Criterion C: Development

Table of Contents

TECHNIQUES USED

1. Password Validation:

Password validation is a form of input validation, which refers to the process of checking if the input provided by the user meets the specified requirements.

Two variables 'password' and 'confirm_password' were created in the client registration page(client_reg.html). In the function validatePassword(), a **conditional if-else statement** was used to check if both the password inputs were matching. If the value in the variable 'password'(password.value) is not equal to the value in 'confirm_password'(confirm_password.value), error message "Passwords Don't Match" will be displayed. This prevents inappropriate data from entering the database and allows new clients to secure their account.

```
<script>
    var password = document.getElementById("password")
  , confirm_password = document.getElementById("confirm_password");

function validatePassword(){
  if(password.value != confirm_password.value) {
    confirm_password.setCustomValidity("Passwords Don't Match");
  } else {
    confirm_password.setCustomValidity('');
  }
}

password.onchange = validatePassword;
confirm_password.onkeyup = validatePassword;
</script>
```

Figure 1: Validate Password function

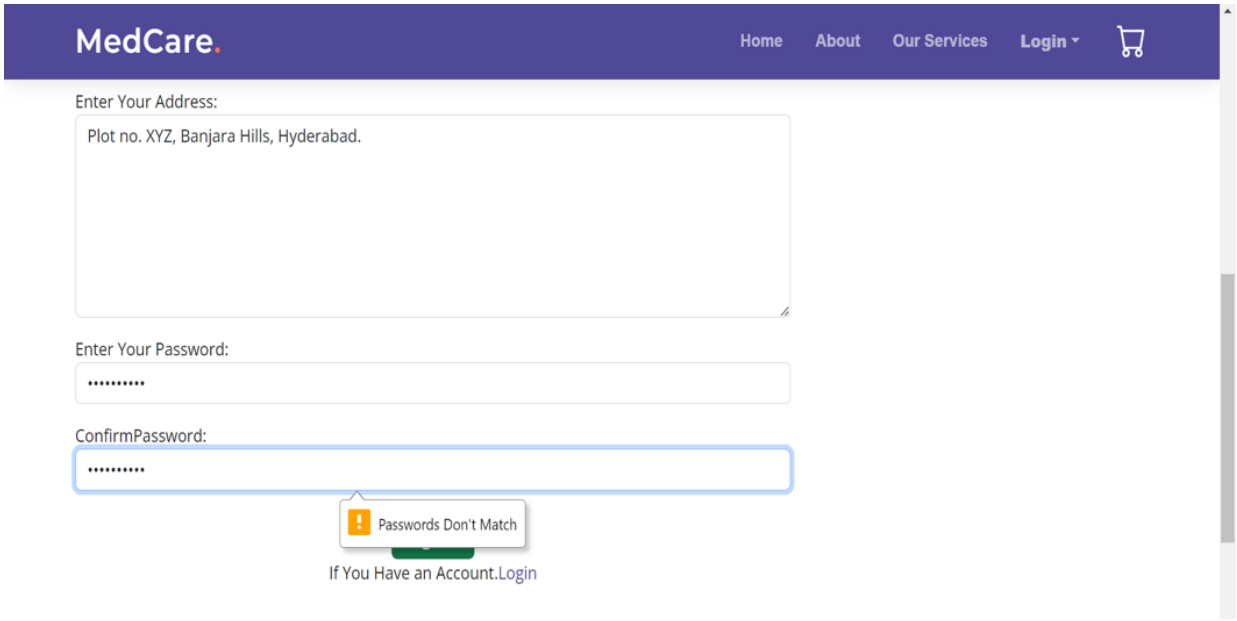


Figure 2: "Passwords Don't Match" error message

2. Exception Handling:

Exception handling is a technique used in programming to deal with errors or unexpected situations that may occur during the execution of the program. In python, exception handling is usually implemented using the **try-except block.**

```
if Client.objects.filter(email=email).exists():
    print("email already taken")
    return render(request, "client_reg.html", {"msg": "email already taken"})
else:
    form = ClientForm(request.POST)
    print("error:", form.errors)
    if form.is_valid():
        try:
            form.save()
            return render(request, "client.html", {"msg": "REGISTER SUCCESS"})
        except Exception as e:
            print(e)

    return render(request, "client_reg.html", {"msg": "REGISTER FAILED"})
```

Figure 3: Try-except block

'ClientForm' is a Django form that has been defined in the forms.py file. When a new client submits a from using a POST request, the code first checks through

the '**Client.objects.filter(email=email).exists()'** query if there is an existing 'Client' object within the given 'email' field value. If so, it returns an "email already taken" message to the user

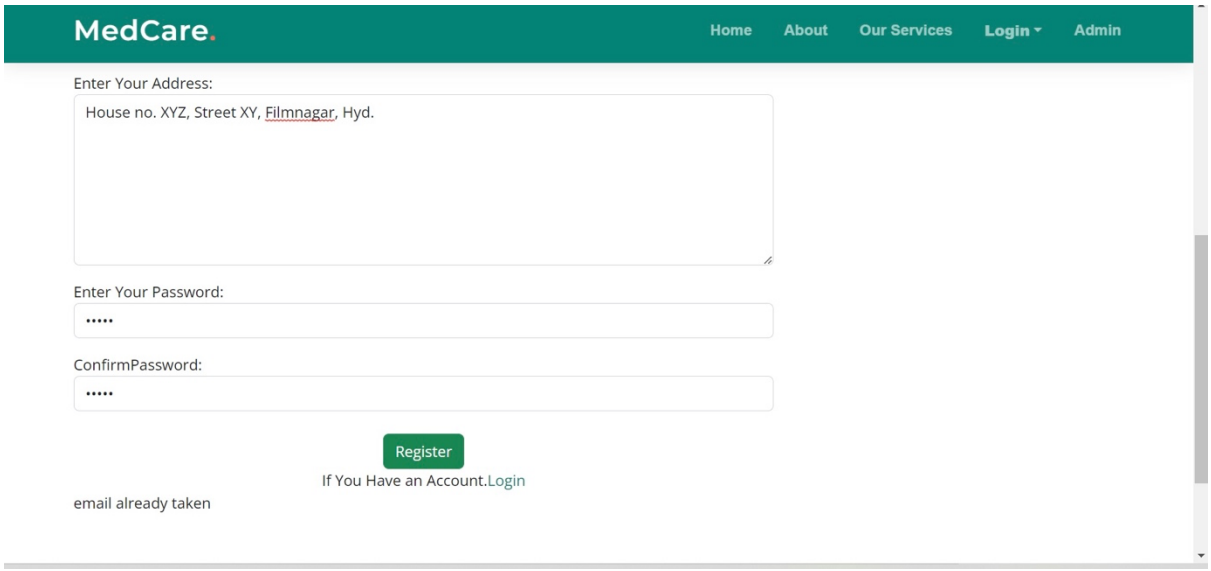


Figure 4: 'Email already taken' error message

'Client.objects.filter' is a method used for retrieving the subset of objects in the 'Client' database table based on certain specifications.
If the email is not taken, it creates an instance of the **'ClientForm'** with the data from the POST request using '**form = ClientForm(request.POST)'.** It then checks if the form is valid using **'form.is_valid()'.**

If the form is valid, it saves the new Client instance to the database using **form.save()** and returns a **"Registration Success"** message to the user. **Form.save()** is a method that is used from the submitted **'ClientForm'** to the database. If the form is valid, the **form.save()** method is called to save the form data to the database.

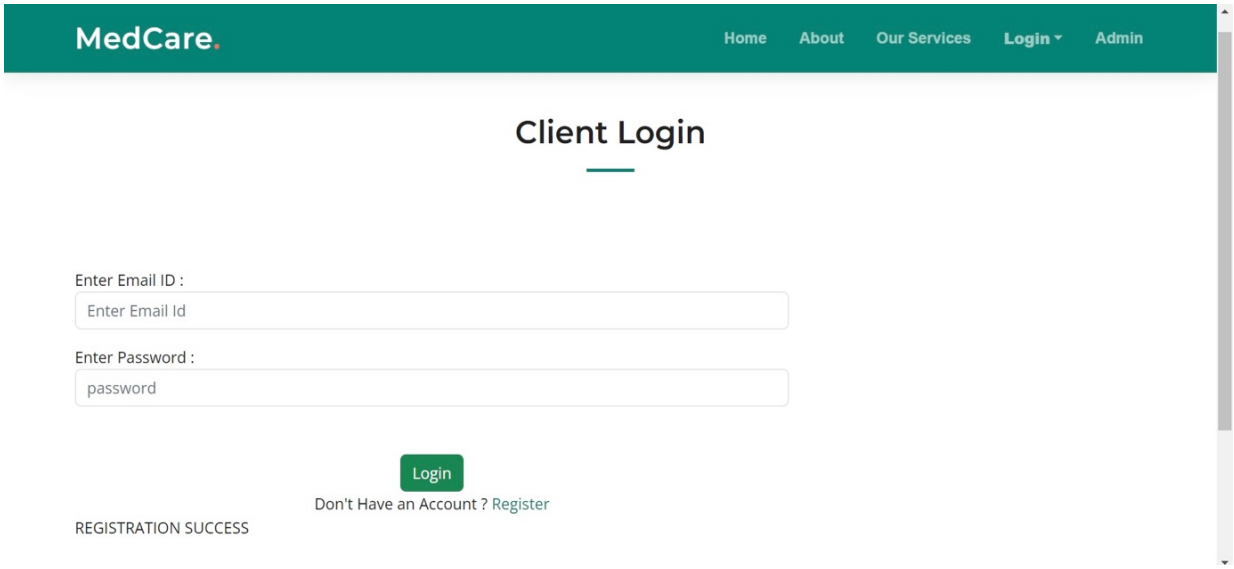


Figure 5: New client 'Registration Success' message

If an exception occurs in the try block, i.e., if the form is not valid, the except block is executed, where it returns a message **"Registration Failed"** to the user. The **'as'** keyword is used to assign the exception object to the variable '**e'.** The except block can then use the **'e'** variable to access information about the exception, such as its type and message.

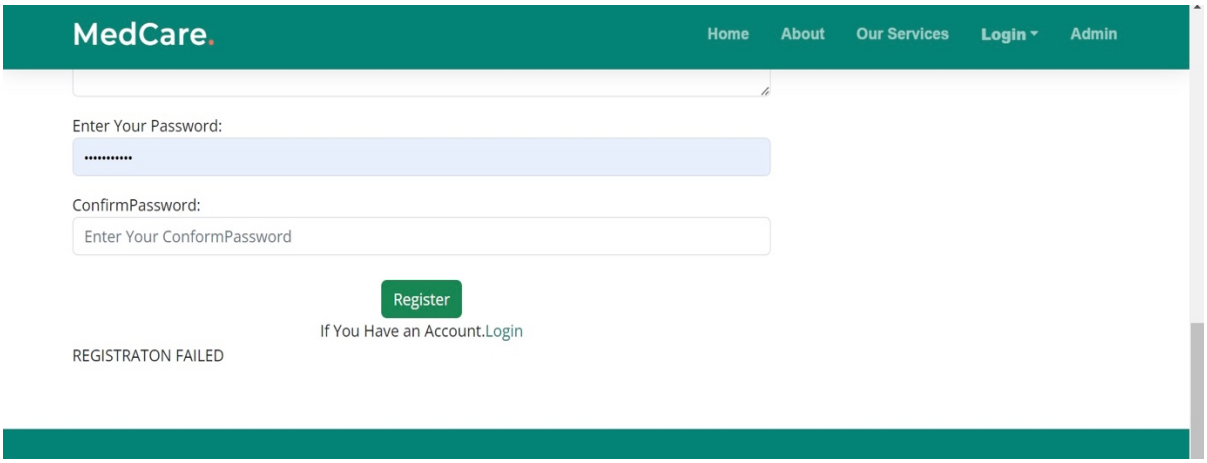


Figure 6: New client 'Registration Failed' message

3. User Authentication

User authentication refers to the process of verifying the user's identity for security purposes during the interactions on a network.

(a) Client & Staff Login

```
41
42  def customer_login(request):
43      if request.method == "POST":
44          email = request.POST["email"]
45          password = request.POST["password"]
46          print(email, "", password)
47          customer = Client.objects.filter(email=email, password=password)
48          if customer.exists():
49              request.session['email'] = email
50              print(email)
51              client = Client.objects.get(email=email)
52              return render(request, "client_profile.html", {"msg": client.firstname})
53          else:
54              return render(request, "client.html", {"msg": "does not exist"})
55      return render(request, "client.html", {})
56
57
```

Figure 7: Customer login function

This authenticates a customer/client login in the web application. The function retrieves the email and password submitted by the user in POST request, and prints them to the console. The function retrieves a 'Client' object form the database that matches the credentials provided by the user. If such an object exists, the email is stored in the session and the user is redirected to the account's profile page. If the 'Client' object does not exist, the user is redirected back to the login page along with an error message.

```
def staff_login(request):
    if request.method == "POST":
        staffid = request.POST["staffid"]
        password = request.POST["password"]
        print(staffid, "", password)
        staff = Staff.objects.filter(staffid=staffid, password=password)
        if staff.exists():
            request.session['staffid'] = staffid
            return render(request, "staff_profile.html", {"msg": staffid})
        else:
            return render(request, "staff.html", {"msg": "not exist"})
    return render(request, "staff.html", {"msg": "not exist"})
```

Figure 8: Staff login function

Similarly, this authenticates a staff member login. The function retrieves the staffid an password submitted by the user in the POST request. It retrieves a 'Staff' object from the staff database which matches the credentials provided by the user. If such as object exists, the user is redirected to the staff account's profile page. If the 'Staff' object does not exist, the user is redirected back to login page with an error.

(b) Account Deletion

```
def client_delete(request, email):
    user = Client.objects.get(email=email)
    user.delete()
    return redirect("/client_reg")
```

Figure 9: Client Account delete function

This function deleted a client form the database based on their email address, the redirects the user to the client registration page. In the 'details' page of the client's account, a confirmation dialog box appears before the user permanently deletes their account – as shown in the figure below for user1.

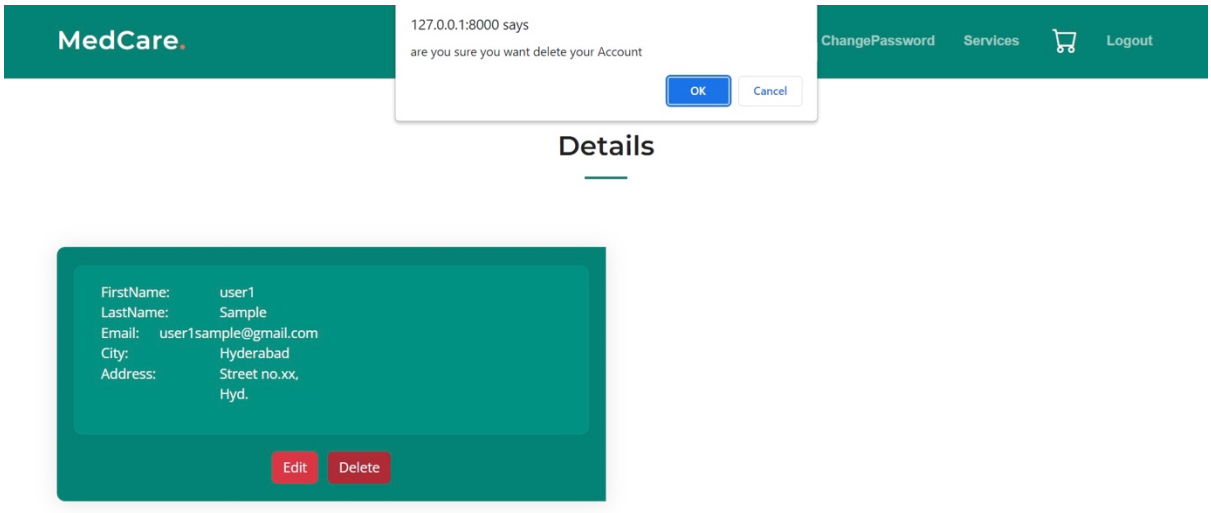


Figure 10: Confirmation dialog box for User1

4. Password Reset

```
106
107   def client_changepassword(request):
108       if is_client(request):
109           if request.method == "POST":
110               email = request.session["email"]
111               password = request.POST["password"]
112               newpassword = request.POST["newpassword"]
113               try:
114                   user = Client.objects.get(email=email, password=password)
115                   user.password = newpassword
116                   user.save()
117                   msg = 'password update successfully'
118                   return render(request, "client.html", {"msg": msg})
119               except:
120                   msg = 'Password is Wrong'
121                   return render(request, "client_password.html", {"msg": msg})
122           return render(request, "client_password.html", {})
123       else:
124           return render(request, "client_password.html", {"msg":"data not valid"})
125
```

Figure 11: Password Reset function

This function allows a client to change their password in their account. It verifies the current password, updates the clients password in the database and displays a “password updated successfully” message to the user.

5. Bootstrap

By making use of Bootstrap CSS library, I could make the design of my web app refined and appealing for the users. Bootstrap CSS Stylesheet and Bootstrap Icons CSS Stylesheet were used which provides styling for HTML elements and components. Bootstrap being a convenient front-end development framework tool helped me save time and effort in creating visually appealing designs. It enhances the overall look if the website makes the development faster by providing pre-built templates and CSS Styles which can be customized.

```
<!-- Vendor CSS Files -->
<link href="{% static 'assets/vendor/bootstrap/css/bootstrap.min.css' %}"
 rel="stylesheet">
<link href="{% static 'assets/vendor/bootstrap-icons/bootstrap-icons.css' %}"
rel="stylesheet">
```

```
<!-- Vendor JS Files -->
<script src="{% static 'assets/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
```

Figure 12: Bootstrap CSS & JS files

6. POST requests

POST requests are used to submit data from the browser to the server, as the information of the users is majorly inputted through text, in forms such as login, register, contact and servicebookings page. “POST” request send the data from the client to the server, which then validates the data and stores it in the database.

```
def staff_login(request):
    if request.method == "POST":
        staffid = request.POST["staffid"]
        password = request.POST["password"]
        print(staffid, "", password)
```

```
def customer_login(request):
    if request.method == "POST":
        email = request.POST["email"]
        password = request.POST["password"]
        print(email, "", password)
        customer = Client.objects.filter(email=email, password=password)
```

```
def client_changepassword(request):
    if is_client(request):
        if request.method == "POST":
            email = request.session["email"]
            password = request.POST["password"]
            newpassword = request.POST["newpassword"]
```

```
def contact_page(request):
    if request.method == 'POST':
        contact = ContactForm(request.POST)
        print(contact.errors)
```

Figure 13: POST request methods

7. CRUD

CRUD Operations are used to create, read, update and delete data from databases. This application contains methods corresponding to these operations.

Create

```
operations = [
    migrations.CreateModel(
        name='Client',
        fields=[
            ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
            ('firstname', models.CharField(max_length=50)),
            ('lastname', models.CharField(max_length=50)),
            ('city', models.CharField(max_length=50)),
            ('email', models.EmailField(max_length=50)),
            ('address', models.CharField(max_length=500)),
            ('password', models.CharField(max_length=100)),
            ('confirm_password', models.CharField(max_length=100)),
        ],
        options={
            'db_table': 'client',
        },
    ),
    migrations.CreateModel(
        name='Contact',
        fields=[
            ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
            ('name', models.CharField(max_length=100)),
            ('email', models.CharField(max_length=100)),
            ('subject', models.CharField(max_length=100)),
            ('message', models.CharField(max_length=100)),
        ],
        options={
            'db_table': 'contact',
```

Figure 14: Create operation

In the above code, **'migrations.CreateModel()'** represents the create operation, which creates new databases along with specified name and fields. The two models created are 'Client' and 'Contact'.

Read

```
        email = request.POST["email"]
        password = request.POST["password"]
        print(email, "", password)
        customer = Client.objects.filter(email=email, password=password)
        if customer.exists():
            request.session['email'] = email
            print(email)
            client = Client.objects.get(email=email)
            return render(request, "client_profile.html", {"msg": client.firstname})
```

Figure 15: Read operation

The read operation is being performed in the 'Client' model using the 'object.filter()' method to retrieve he required client information from the database.

Update

```
    user = Client.objects.get(email=email, password=password)
    user.password = newpassword
    user.save()
    msg = 'password update successfully'
    return render(request, "client.html", {"msg": msg})
```

Figure 16: Update operation

The update command is ‘user.save()’, which saves the updates password from the change password page to the database. This command updates the corresponding row in the ‘Client’ table with the new password for the client whose email and old password match the values provided in the request.

Delete

```
def client_delete(request, email):
    user = Client.objects.get(email=email)
    user.delete()
    return redirect("/client_reg")
```

Figure 17: Delete operation

The delete operation which is performed using the command ‘user.delete()’ which deletes the Client object from the database.

8. Inheritance

Inheritance is an OOP principle, where a new class is created by inheriting the properties and method of an existing class.

Figure 18: Inheritance in forms.py

In forms.py, each form class is a subclass of Django's '**ModelForm',** which is the parent class from where these classes inherit all the properties and methods, used to create the fields and validation rules.
'forms.ModelForm' specifies a **'model'** attribute, which determines the model that the form is based on.

For example, '**ContactForm'** is a subclass of **'ModelForm'** and specifies its respective '**Contact**' model as its **'Meta'** class. This means **'ContactForm'** class will inherit all the attributes and methods of **'ModelForm'** based on the fields of the **'Contact'** model.

9. jQuery

```
<!-- jQuery -->
<script
src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous">
</script>

<!-- XDSoft DateTimePicker -->
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/jquery-datetimepicker/2.5.20/jquery.datetimepicker.min.css"
integrity="sha256-DOS9W6NR+NFe1fUhEE0PGKY/fubbUCnOfTje2JMDw3Y="
crossorigin="anonymous"
/>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery-datetimepicker/2.5.20/jquery.datetimepicker.full.min.js"
integrity="sha256-FEqEelWI3WouFOo2VWP/uJfs1y8KJ++FLh2Lbqc8SJk="
crossorigin="anonymous"></script>
```

Figure 19: jQuery & DateTimePicker

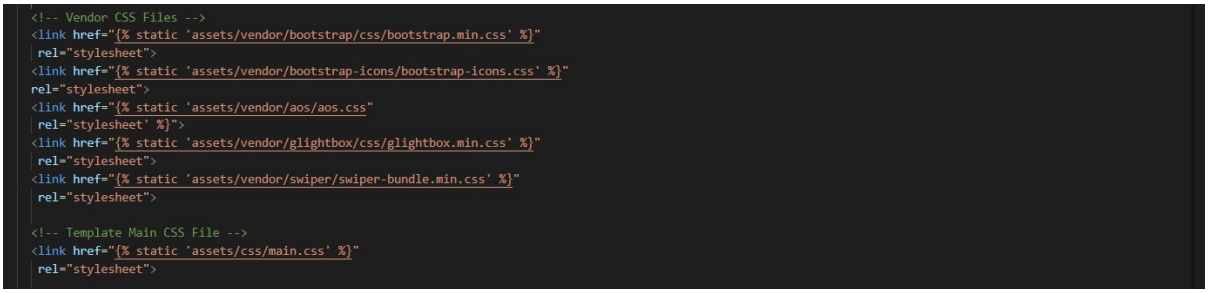
jQuery, a JavaScript library, was used to simplify JavaScript programming and add interactivity to the web page, specifically for handling events such as clicking on elements and animating elements. The script tag is linking to a slim version of the jQuery library, which is a lightweight version of the library that includes the commonly used features and helps reduce the size of webpage and improve its loading speed.

A jQuery plugin was used to provide a user-friendly date and time picker widget for various forms on the website. The plug consists of two parts, a CSS file and a JavaScript file adding to the appearance and functionality of the widget respectively. The XDSoftDateTimePicker can be initialized on any form input using JS. This enables the user to select a date and time from a pop-up calendar.

10. Additional JavaScript libraries

```
<!-- Vendor JS Files -->
<script src="{% static 'assets/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
<script src="{% static 'assets/vendor/aos/aos.js' %}"></script>
<script src="{% static 'assets/vendor/glightbox/js/glightbox.min.js' %}"></script>
<script src="{% static 'assets/vendor/purecounter/purecounter_vanilla.js' %}"></script>
<script src="{% static 'assets/vendor/swiper/swiper-bundle.min.js' %}"></script>
<script src="{% static 'assets/vendor/isotope-layout/isotope.pkgd.min.js' %}"></script>
<script src="{% static 'assets/vendor/php-email-form/validate.js' %}"></script>

<!-- Template Main JS File -->
<script src="{% static 'assets/js/main.js' %}"></script>
```



```
<!-- Vendor CSS Files -->
<link href="{% static 'assets/vendor/bootstrap/css/bootstrap.min.css' %}"
 rel="stylesheet">
<link href="{% static 'assets/vendor/bootstrap-icons/bootstrap-icons.css' %}"
rel="stylesheet">
<link href="{% static 'assets/vendor/aos/aos.css"
 rel="stylesheet' %}">
<link href="{% static 'assets/vendor/glightbox/css/glightbox.min.css' %}"
 rel="stylesheet">
<link href="{% static 'assets/vendor/swiper/swiper-bundle.min.css' %}"
 rel="stylesheet">

<!-- Template Main CSS File -->
<link href="{% static 'assets/css/main.css' %}"
 rel="stylesheet">
```

Figure 20: Additional JS libraries used

Besides Bootstrap, various JavaScript libraries were included in the web application to ass different functionalities and features for users.

'aos.js': AOS(Animate On Scroll) library that allows flexible animations for elements on scroll.

'glightbox.min.js': GLightbox is a lightweight JS library that provides customisable lightbox for displaying images, videos, and other content.

'purecounter_vanilla.js': Simple and lightweight JS library for adding animated counters to the web application.

'swiper-bundle.min.js': Modern and mobile-friendly slider plugin library enabling touch and mouse support.

'isotope.pkgd.min.js': Library which allows easy filtering and sorting of HTML elements on the website.

'validate.js': This library provides client-side validation for web forms.

'main.js': Custom JS file that contains the main script for the website, including its custom functionality features and interactivity.

'main.css': Custom CSS file containing the main styles for the website, with extended features of HTML elements.

Bibliography:

Aryan, Raj. "CRUD Operation in Database in Python Using SQLite." Python Lobby, 30 Nov. 2020, pythonlobby.com/crud-operation-in-database-in-python-using-sqlite.

Chris, Kolade. "How to Link CSS to HTML – Stylesheet File Linking." freeCodeCamp.org, 15 June 2022, www.freecodecamp.org/news/how-to-link-css-to-html.

DateTimePicker. 3 Apr. 2022, xdsoft.net/jqplugins/datetimepicker.

GeeksforGeeks. "Inheritance in Python." GeeksforGeeks, 21 Nov. 2022, www.geeksforgeeks.org/inheritance-in-python.

JavaScripting.com - the Database of JavaScript Libraries. www.javascripting.com.

Nik. "Python Requests: POST Request Explained." Datagy, 30 Dec. 2022, datagy.io/python-requests-post.

Vishal. "Inheritance in Python." PYnative, 28 Aug. 2021, https://pynative.com/python-inheritance/

W3 Schools, www.w3schools.com/python/python_try_except.asp.

Web Dev Simplified. "Bootstrap 5 Crash Course." YouTube, 20 Aug. 2022, www.youtube.com/watch?v=Jyvffr3aCp0. [Original source: https://studycrumb.com/alphabetizer]