Handling Forms in Django:

- Forms are used for handling user inputs in web applications.

Purpose of forms:

- **Primary Goal:** Collect User input (e.g., login-forms, enquiry-forms, registration-forms)
- **Uses:** Read and process user-provided data based on application requirements.

Types of form handling in Django applications:

- Inside Django, we can create and handle forms in 3 different ways:

1. **HTML based form:** Manual HTML in templates with custom view logic for processing and validation.
2. **Django forms:** Python-based form classes with built-in rendering, validation, and error handling.
3. **Django Model-Based Forms**: Forms tied to models, automating field generation and database interaction.

HTML based Forms:

- **Overview:** HTML forms involve writing raw HTML in templates and processing request.POST data in views. This method provides full control over the form's structure and styling but requires manual validation at server side.

Steps

1. **Define the HTML Form**: Use <form> tags with inputs (e.g., text, radio, checkboxes, select) in a template.
2. **Handle Submission in View**: Extract data by using request.POST.get(), validate it, and process it (e.g., save to database).
3. **Add CSRF Token**: Include {% csrf_token %} for security (explained below).

Use of csrf_token

- **Purpose**: Protects against **Cross-Site Request Forgery (CSRF)** attacks, where a malicious site tricks a user into submitting a form to your site without their knowledge.
- **How It Works**: Django generates a unique token for each user session and includes it in forms as a ***hidden field***. When the form is submitted, Django verifies the token matches the session's token, ensuring the request originates from your site.
- **Real-Time Analogy**: Imagine you're a bank teller (Django app) and a customer (user) wants to withdraw money. You give them a special ticket (CSRF token) with their account number when they enter the bank. When they submit a withdrawal request (form submission), you check the ticket matches their account (session). If a thief (malicious site) tries to forge a request without the ticket, you reject it because it doesn't match.

Example1: Product Registration form using HTML based Form:

- Create a new project called **HTMLFormProject**

  django-admin startproject HTMLFormProject

- Move inside the Project folder

  cd HTMLFormProject

- Create a new application called **ProductApp** inside the above project.

python manage.py startapp ProductApp

- Register this **ProductApp** inside the settings.py file

- Define the following **Product** model class inside the **ProductApp/models.py** file.

ProductApp/models.py:

```
from django.db import models

# Create your models here.


class Product(models.Model):
    product_id = models.AutoField(primary_key=True)
    product_name = models.CharField(max_length=20)
    price = models.IntegerField()
    quantity = models.IntegerField()

    # Define choices for the category field
    CATEGORY_CHOICES = (
        ('Electronics', 'Electronics'),
        ('Stationary', 'Stationary'),
```

```
    ('HomeAppliances', 'HomeAppliances'),
)

category = models.CharField(max_length=20, choices=CATEGORY_CHOICES)
created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

def __str__(self):
    return self.product_name
```

- We added a CATEGORY_CHOICES tuple that defines the options for the category dropdown. Each tuple inside CATEGORY_CHOICES contains two values:
    - The first value (e.g., 'Electronics') is the value that will be stored in the database.
    - The second value (e.g., 'Electronics') is the human-readable name that will appear in the dropdown.
- The **category** field now uses models.CharField with the choices=CATEGORY_CHOICES parameter. This ensures that only the values defined in CATEGORY_CHOICES can be selected, and the admin interface will automatically render it as a dropdown.

```
CATEGORY_CHOICES = (
    ('ELECT', 'Electronics'),
    ('STAT', 'Stationary'),
    ('HOME', 'HomeAppliances'),
)
```

- In this case, 'ELECT', 'STAT', and 'HOME' will be stored in the database, but "Electronics," "Stationary," and "Home Appliances" will appear in the dropdown.
- auto_now_add=True sets the field to the current timestamp when the object is first created and cannot be edited afterward.
- auto_now=True updates the field to the current timestamp every time the object is saved (created or updated).

- **Perform the migrations:**

  python manage.py makemigrations

  python manage.py migrate

- Register the above **Product** class with **ProductAdmin** classes inside the **ProductApp/admin.py** file to access it inside the admin interface.

ProductApp/admin.py:

```
from django.contrib import admin
from ProductApp.models import Product
# Register your models here.
class ProductAdmin(admin.ModelAdmin):
    list_display = ['product_id', 'product_name', 'price',
                    'quantity', 'category', 'created_at', 'updated_at']
```

```
admin.site.register(Product, ProductAdmin)
```

- Create a super-user to access the admin interface.
- Run the server and access the admin interface

  python manage.py createsuperuser

  http://127.0.0.1:8000/admin

- Add a few products inside the database using the admin interface.

Creating the Product registration HTML form:

- Define a view function to render the home page inside **ProductApp/views.py** file

ProductApp/views.py:

```
from django.shortcuts import render

# Create your views here.

def home_view(request):

    return render(request, 'home.html')
```

- Define the url pattern for the above view function inside the **ProductApp/urls.py** file:

ProductApp/urls.py:

```
from django.urls import path

from . import views

urlpatterns = [
```

```
    path('', views.home_view, name='home')
]
```

- Include the above ProductApp.urls.py file inside the Project level urls.py file:

Project level urls.py file:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('ProductApp.urls'))
]
```

- Create home.html file inside the **ProductApp/templates** folder

home.html:

```
<!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 bgcolor="wheat">

    <h1 style="text-align: center;">Welcome to the Product Management Application</h1>

    <a href="#">Add New Product</a>

    <br><br>

    <a href="#">Get All Products</a>

</body>

</html>
```

Implementing Add New Product functionality:

- Define the following view function inside the **ProductApp/view.py** file to render the Product registration page.

```
def add_product_view(request):

    return render(request, 'addproduct.html')
```

- Create **addproduct.html** file inside the **ProductApp/templates** folder

addproduct.html:

```
<!DOCTYPE html>

<html lang="en">


<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add New Product</title>
</head>

<body bgcolor="lightblue">
    <h1 style="text-align: center;">Add a New Product</h1>
    <form method="POST">
        {% csrf_token %}
        <label for="product_name">Product Name:</label>
        <input type="text" id="product_name" name="product_name"
required><br><br>

        <label for="price">Price:</label>
        <input type="number" id="price" name="price" required><br><br>

        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" name="quantity"
required><br><br>

       <label for="category">Category:</label>
        <select id="category" name="category" required>
            <option value="Electronics">Electronics</option>
            <option value="Stationary">Stationary</option>
```

```
            <option value="HomeAppliances">HomeAppliances</option>
        </select><br><br>



        <input type="submit" value="AddProduct">
    </form>
    <br>
    <a href="{% url 'home' %}">Back to Home</a>


    <!-- Display Error Message -->
    {% if error_msg %}
    <div style="color: red;">
        <strong>{{ error_msg }}</strong>
    </div>
    <br>
    {% endif %}
</body>
</html>
```

Note: In the <form> since we have not defined the **action** attribute, so once we submit the form it uses the POST method and the form submission will be there inside the same view function. So to handle the form submission data we need to modify the same view function inside the **ProductApp/views.py** file.

Modify the same view function inside the ProductApp/views.py file:

```python
def add_product_view(request):
    error_msg = ''  # Initialize error message

    if request.method == 'POST':
        product_name = request.POST.get('product_name')
        price = request.POST.get('price')
        quantity = request.POST.get('quantity')
        category = request.POST.get('category')

        # Validation
        if not product_name or not price or not quantity or not category:
            error_msg = 'All fields are required.'
        elif int(price) <= 0:
            error_msg = 'Price must be greater than zero.'
        # Check if product name already exists
        elif Product.objects.filter(product_name=product_name).exists():
            error_msg = 'A product with this name already exists.'
        else:
            # If validation passes, create the product
            Product.objects.create(
                product_name=product_name,
                price=int(price),
```

```
            quantity=int(quantity),
            category=category
        )
        # Redirect to home page after successful submission
        return redirect('home')

return render(request, 'addproduct.html', {'error_msg': error_msg})
```

- Define the url pattern for the above view function inside the **ProductApp/urls.py** file:

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.home_view, name='home'),
    path('addproduct/', views.add_product_view, name='addproduct'),
]
```

- Define the name of this url pattern to the <a> tag of **home.html** file

```
<a href="{% url 'addproduct' %}">Add New Product</a>
```

- Set the time zone inside the settings.py:

```
TIME_ZONE = 'Asia/Kolkata'
```

Assignment:

- Implement the Get All Product functionality where display all product lists inside a simple HTML table, and no products are there then show a proper message.

Hint:

```
<thead>
    <tr>
       <th>Product ID</th>
        <th>Product Name</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Category</th>
        <th>Created At</th>
        <th>Updated At</th>
    </tr>
</thead>
```

Example2: Product Registration using Django Forms:(Python based Form classes)

- Django Forms allow you to define forms as Python classes, providing built-in validation, rendering, and error handling. This approach is more efficient than HTML-based forms because Django handles much of the heavy lifting.

Step 1: Create a Form Class

Create a new file forms.py in the ProductApp directory to define the form.

ProductApp/forms.py:

```
from django import forms

from django.core.exceptions import ValidationError

from ProductApp.models import Product




class ProductForm(forms.Form):

    product_name = forms.CharField(

        max_length=20, required=True, label='Product Name')

    price = forms.IntegerField(min_value=1, required=True, label='Price')

    quantity = forms.IntegerField(min_value=0, required=True,
label='Quantity')



    # Define choices for the category field

    CATEGORY_CHOICES = (

        ('Electronics', 'Electronics'),

        ('Stationary', 'Stationary'),

        ('Home Appliances', 'Home Appliances'),

    )



    category = forms.ChoiceField(

        choices=CATEGORY_CHOICES, required=True, label='Category')
```

```
    def clean_product_name(self):
        product_name = self.cleaned_data.get('product_name')
        # Check if product already exists
        if Product.objects.filter(product_name=product_name).exists():
            raise ValidationError(
                f"A product with the name '{product_name}' already
exists.")
        return product_name
```

- We can implement the custom validation inside the form class by using **clean_<field_name>**
- This method is automatically called by Django when the form is being validated.

Step 2: Update Views

Modify the **ProductApp/views.py** to use the form for rendering and processing.

ProductApp/views.py:

```
from ProductApp.forms import ProductForm
def add_product_view(request):
    form = ProductForm()
    if request.method == 'POST':
```

```
        form = ProductForm(request.POST)
        if form.is_valid():
            # Extract cleaned data
            product_name = form.cleaned_data['product_name']
            price = form.cleaned_data['price']
            quantity = form.cleaned_data['quantity']
            category = form.cleaned_data['category']

            # Save to database
            Product.objects.create(
                product_name=product_name,
                price=price,
                quantity=quantity,
                category=category
            )
            return redirect('home')

    return render(request, 'addproduct.html', {'form': form})
```

- If the request method is **GET**, we create an empty form to display.
- In add_product_view, if the request method is POST, we instantiate the form with request.POST to validate the submitted data. If form.is_valid(), we use form.cleaned_data to get validated data safely.

Step 3: Update Templates

Modify addproduct.html to render the Django Form.

ProductApp/templates/addproduct.html:

```
<!DOCTYPE html>

<html lang="en">


<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Add New Product</title>

<style>

        .errorlist {

            color: red;

            font-size: 20px;

            margin-top: 5px;

        }

    </style>


</head>


<body bgcolor="lightblue">

    <h1 style="text-align: center;">Add a New Product</h1>
```

```
    <form method="POST">
        {% csrf_token %}
        {{form.as_p}}
        <input type="submit" value="Add Product">
    </form>
    <br>
    <a href="{% url 'home' %}">Back to Home</a>



</body>
</html>
```

- Instead of manually writing HTML input fields, we use {{ form.as_p }} to render all form fields as paragraphs. Django automatically includes labels, inputs, and validation errors.
- form.errors allows displaying validation errors if the form data is invalid.
- Django renders error messages inside `<ul class="errorlist">`

Example3: Product Registration using Django Model based Form:

- Django Model Forms are tied to models, automatically generating form fields based on the model fields and handling database interactions. This is ideal for CRUD operations.

Step 1: Create a Model Form

Update or create forms.py to include a ModelForm.

ProductApp/forms.py (replace or add to the previous content):

```
from django import forms

from .models import Product

class ProductModelForm(forms.ModelForm):

    class Meta:

        model = Product

        fields = ['product_name', 'price', 'quantity', 'category']

        labels = {

            'product_name': 'Product Name',

            'price': 'Price',

            'quantity': 'Quantity',

            'category': 'Category',

        }

        widgets = {

            'category': forms.Select(),

        }


    # Custom validation for the product_name field

    def clean_product_name(self):

        product_name = self.cleaned_data['product_name']

        # Check if the product name already exists in the database
```

```
        if Product.objects.filter(product_name=product_name).exists():
            raise forms.ValidationError(f"A product with the name '{product_name}' already exists.")

        return product_name
```

Explanation:

- ProductModelForm inherits from forms.ModelForm.
- The Meta class specifies the model (Product) and the fields to include.
- labels customizes the field labels for better readability.
- widgets overrides the default widget for category to use a Select dropdown with the CATEGORY_CHOICES defined in the model.

Notes on Django Widgets

Widgets in Django are responsible for rendering and handling form elements (fields) in the HTML template. They control how the form field is displayed in the HTML and how data is presented and validated.

- Widgets are classes in Django that control the HTML output of a form field.
- A widget is responsible for rendering the form field and processing the user input when the form is submitted.
- Widgets allow you to control the input field's appearance and behavior on the frontend (such as choosing the input type, adding CSS classes, etc.).

2. Default Widgets in Django

- Every field in a Django form is rendered using a default widget. For instance:
    - `CharField`: Uses a `TextInput` widget.
    - `IntegerField`: Uses a `NumberInput` widget.
    - `ChoiceField`: Uses a `Select` widget.
    - `DateField`: Uses a `DateInput` widget.
- These default widgets can be customized by specifying the `widget` attribute in the form field definition.

3. Commonly Used Widgets:

- Django provides several built-in widgets that you can use to control how the form fields are rendered:

a. **PasswordInput**: Renders a password input field (similar to `TextInput`, but with password hiding).

```
password = forms.CharField(widget=forms.PasswordInput)
```

b. **Select**: Renders a dropdown menu (for `ChoiceField)`

```
category = forms.Select(choices=[('Electronics', 'Electronics'), ('Stationary', 'Stationary')])
```

c. **RadioSelect**: Renders radio buttons.

```
gender = forms.ChoiceField(choices=[('M', 'Male'), ('F', 'Female')], widget=forms.RadioSelect)
```

d. **CheckboxInput**: Renders a checkbox (for boolean values).

subscribe = forms.BooleanField(widget=forms.CheckboxInput)

4. **Customizing Widgets**

- Widgets can be customized by modifying their `attrs` argument. The `attrs` dictionary allows you to add custom HTML attributes to the input element, like `class`, `id`, `style`, etc.

Example:

name = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Enter your name'}))

Step 2: Update Views

Modify **ProductApp/views.py** to use the Model Form.

```
def add_product_view(request):
    form = ProductModelForm()
    if request.method == 'POST':
        form = ProductModelForm(request.POST)
        if form.is_valid():
            form.save()  # Automatically saves to the database
            return redirect('home')



    return render(request, 'addproduct.html', {'form': form})
```

Explanation:

- We use ProductModelForm instead of ProductForm.
- When the form is valid, form.save() automatically creates a new Product instance in the database, mapping the form fields to the model fields.
- No need to manually extract and save data, as the Model Form handles this.

Step 3: Update Templates

- The addproduct.html template remains similar but uses the Model Form.

Example: Student Registration Form with Course Enrollment

- Create a new project called **HTMLFormProject:**

  django-admin startproject HTMLFormProject2

- Move inside the project directory:

  cd HTMLFormProject2

- Create a StudentApp inside the HTMLFormProject

  python manage.py startapp StudentApp

- Register the **StudentApp** inside the settings.py file.

- Define the following model classes inside the models.py file of the **StudentApp**

```
from django.db import models

# Create your models here.



class Course(models.Model):
    course_id = models.AutoField(primary_key=True)
    cname = models.CharField(max_length=100)
    fee = models.IntegerField()
    duration = models.CharField(max_length=20)

    def __str__(self):
        return self.cname



class Student(models.Model):
    GENDER_CHOICES = (
        ('Male', 'Male'),
        ('Female', 'Female'),
        ('Other', 'Other'),
    )

    QUALIFICATION_CHOICES = (
```

```
            ('B.Tech', 'B.Tech'),
            ('M.Tech', 'M.Tech'),
            ('BCA', 'BCA'),
            ('MCA', 'MCA'),
            ('Other', 'Other'),
        )

        roll = models.AutoField(primary_key=True)
        name = models.CharField(max_length=100)
        email = models.EmailField(unique=True)
        address = models.CharField(max_length=50)

        # e.g., Male, Female, Other
        gender = models.CharField(max_length=10, choices=GENDER_CHOICES)
        courses = models.ManyToManyField(Course)  # many to many
    relationship
        qualification = models.CharField(
            max_length=50, choices=QUALIFICATION_CHOICES)
        # e.g., B.Tech, M.Tech, BCA, MCA
```

- **Apply the migrations:** to create tables for the above model classes.

  ```
  python manage.py makemigrations
  python manage.py migrate
  ```

- Register both classes inside the **StudentApp/admin.py** file to access in the admin interface.

StudentApp/admin.py:

```
from django.contrib import admin
from django.forms import RadioSelect
from .models import Course, Student
class CourseAdmin(admin.ModelAdmin):
    list_display = ('course_id', 'cname', 'fee', 'duration')  # Display
columns
class StudentAdmin(admin.ModelAdmin):
    list_display = ('name', 'email', 'gender',
                    'qualification', 'enrolled_courses')

#   To display the gender as radio button
    radio_fields = {'gender': admin.VERTICAL}

    def enrolled_courses(self, obj):
        course_names = []
        for course in obj.courses.all():
            course_names.append(course.cname)
        return ", ".join(course_names)
admin.site.register(Course, CourseAdmin)  # Register Course with
CourseAdmin
```

```
# Register Student with StudentAdmin

admin.site.register(Student, StudentAdmin)
```

- **Create the superuser and access the admin interface by running the server**

  python manage.py createsuperuser

  python manage.py runserver

  http://127.0.0.1:8000/admin

- Define the following view function inside the **StudentApp/views.py** file to render the Home page

```
from django.shortcuts import render

# Create your views here.

def home_view(request):

    return render(request, 'home.html')
```

- Create a base.html file inside the **StudentApp/templates** folder by adding the bootstrap links

base.html:

```
<!DOCTYPE html>
```

```
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title_block %} HTMLFormProject {% endblock %}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">
    <style>
        body {
            margin: 0;
            padding: 0;
        }

        header {
            height: 10vh;
            background-color: rosybrown
        }

        main {
            height: 80vh;
            background-color: aquamarine;
```

```
        }

        footer {
            height: 10vh;
            background-color: powderblue;
        }
    </style>
</head>

<body>
    <header>

        <nav class="nav justify-content-end">

            <a class="nav-link" href="#">Student App</a>
            <a class="nav-link" href="#">Add New Course</a>
            <a class="nav-link" href="#">Register New Student</a>
        </nav>
    </header>
    <main>
        {% block main_block %}
        {% endblock %}
    </main>
```

```
    <footer>
        <p class="text-center">&copy; This is the footer section</p>
    </footer>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN
7N6jIeHz" crossorigin="anonymous"></script>

</body>


</html>
```

home.html:

```
{% extends "base.html" %}

{% block title_block %} Home Page {% endblock %}

{% block main_block %}

<h3 class="text-center">Welcome to Home page</h3>

{% endblock %}
```

- Define the url pattern for the home_view function inside the **StudentApp/urls.py** file.

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

```
from . import views

urlpatterns = [

    path('', views.home_view, name='home')

]
```

- Include this urls.py file inside the Project level urls.py file.

```
from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]
```

- Run the server:

  python manage.py runserver

- Access the application:

  http://127.0.0.1:8000/

Implement the *Add New Course* functionality:

- Define the following view function inside the **StudentApp/views.py** file:

```
from django.shortcuts import render, redirect
```

```
from StudentApp.models import Course
from django.contrib import messages


def add_course_view(request):
    # Getting all courses from the database
    courses = Course.objects.all()

    if request.method == "POST":
        cname = request.POST.get('cname')
        fee = int(request.POST.get('fee'))
        duration = request.POST.get('duration')

        # Validation
        if not cname:
            messages.error(request, "Course name is mandatory")
        elif Course.objects.filter(cname=cname).exists():
            messages.error(request, "Course Name is already
registered")
        elif not fee or fee <= 0:
            messages.error(request, "Fee must be positive Integer")
        else:
            Course.objects.create(cname=cname, fee=fee,
duration=duration)
```

```
            messages.success(request, f"Course {cname} is added
successfully ")
            return redirect('home')

    return render(request, 'addcourse.html', context={"courses":
courses})
```

- **Define the url pattern for the above view function**

```
path('addcourse/', views.add_course_view, name='addcourse'),
```

- **Create addcourse.html file inside the StudentApp/templates folder**

addcourse.html:

```
{% extends "base.html" %}

{% block title_block %} Add Course {% endblock %}

{% block main_block %}

<h3 class="text-center">New Course Addition Screen</h3>

<div class="container">
```

```
    <form method="POST">
        {% csrf_token %}
        <div class="mb-3">
            <label for="cname" class="form-label">Enter Course
Name:</label>
            <input type="text" id="cname" name="cname" required
class="form-control">
        </div>

        <div class="mb-3">
            <label for="fee" class="form-label">Enter Course Fee:</label>
            <input type="number" id="fee" name="fee" required
class="form-control">
        </div>

        <div class="mb-3">
            <label for="duration" class="form-label">Enter Course
Duration:</label>
            <input type="text" id="duration" name="duration" required
class="form-control">
        </div>

        <input type="submit" value="AddCourse" class="btn btn-success">
    </form>
```

```
</div>
<hr>

{% if courses %}

<div class="container">

    <table class="table table-dark">

        <thead>
            <tr>
                <th class="col-2">CourseId</th>
                <th class="col-2">CourseName</th>
                <th class="col-2">Fee</th>
                <th class="col-2">Duration</th>
                <th class="col-4">Action</th>
            </tr>

        </thead>

        <tbody>

            {% for course in courses %}
```

```
            <tr>
                <td>{{course.course_id}}</td>
                <td>{{course.cname}}</td>
                <td>{{course.fee}}</td>
                <td>{{course.duration}}</td>
                <td>
                    <a href="#" class="btn btn-primary">UPDATE</a>
                    <a href="#" class="btn btn-success">DELETE</a>
                    <a href="#" class="btn btn-danger">STUDENT LIST</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endif %}
{% endblock %}
```

- **Modify the base.html to display the generated the messages from the view function:**

base.html:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

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

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">

    <style>

        body {

            margin: 0;

            padding: 0;

        }



        header {

            height: 10vh;

            background-color: rosybrown

        }

        main {

            height: 80vh;

            background-color: aquamarine;

        }
```

```
        footer {
            height: 10vh;
            background-color: powderblue;
        }
    </style>
</head>

<body>
    <header>
        <nav class="nav justify-content-end">
            <a class="nav-link" href="#">Student App</a>
            <a class="nav-link" href="{% url 'addcourse' %}">Add New Course</a>
            <a class="nav-link" href="#">Register New Student</a>
        </nav>
    </header>
    <main class="overflow-auto">
        {% if messages %}
        {% for message in messages %}
        <div class="alert alert-warning  alert-dismissible fade show" role="alert">
            {{message}}
            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
```

```
        </div>


        {% endfor %}

        {% endif %}


        {% block main_block %}

        {% endblock %}

    </main>


    <footer>

        <p class="text-center">&copy; This is the footer section</p>

    </footer>


    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN
7N6jIeHz" crossorigin="anonymous"></script>

</body>


</html>
```

Implementing Register New Student:

- Define the following view function inside **StudentApp/views.py** file

```python
def register_student_view(request):
    courses = Course.objects.all()
    students = Student.objects.all()

    if request.method == "POST":
        name = request.POST.get('name')
        email = request.POST.get('email')
        address = request.POST.get('address')
        gender = request.POST.get('gender')
        # Get multiple selected courses
        course_ids = request.POST.getlist('courses')
        qualification = request.POST.get('qualification')

        # Validation
        if not name:
            messages.error(request, "Name is mandatory")
        elif not email:
            messages.error(request, "Email is mandatory")
        elif Student.objects.filter(email=email).exists():
            messages.error(request, "Email is already registered")
        elif not course_ids:
            messages.error(request, "Please select at least one course")
```

```
        else:
            # Create student
            student = Student.objects.create(
                name=name,
                email=email,
                address=address,
                gender=gender,
                qualification=qualification
            )
            # Add selected courses
            student.courses.set(course_ids)
            messages.success(
                request, f"Student {name} registered successfully with
Roll No: {student.roll}")
            return redirect('home')


    return render(request, 'registerstudent.html', context={'courses':
courses, 'students': students})
```

- Define the url path for the above view function inside **StudentApp/urls.py** file.

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

urlpatterns = [
```

```
    path('', views.home_view, name='home'),

    path('addcourse/', views.add_course_view, name='addcourse'),

    path('registerstudent/', views.register_student_view,
name='registerstudent')



]
```

- Modify the base.html and apply the link to **Register New Student**:

```
    <nav class="nav justify-content-end">

            <a class="nav-link" href="{% url 'home' %}">Student
App</a>

            <a class="nav-link" href="{% url 'addcourse' %}">Add New
Course</a>

            <a class="nav-link" href="{% url 'registerstudent'
%}">Register New Student</a>

        </nav>
```

- Create the **registerstudent.html** file inside the **StudentApp/templates** folder.

registerstudent.html:

```
{% extends "base.html" %}
```

```
{% block title_block %} Register Student {% endblock %}


{% block main_block %}

<h3 class="text-center">New Student Registration Screen</h3>


<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label for="name" class="form-label">Enter Student
Name:</label>

            <input type="text" id="name" name="name" required
class="form-control">

        </div>


        <div class="mb-3">

            <label for="email" class="form-label">Enter Email:</label>

            <input type="email" id="email" name="email" required
class="form-control">

        </div>


        <div class="mb-3">

            <label for="address" class="form-label">Enter Address:</label>

            <input type="text" id="address" name="address" required
class="form-control">
```

```
        </div>


        <div class="mb-3">
            <label for="gender" class="form-label">Select Gender:</label>
            <select id="gender" name="gender" required
class="form-control">
                <option value="">Select Gender</option>
                <option value="Male">Male</option>
                <option value="Female">Female</option>


            </select>
        </div>


        <div class="mb-3">
            <label class="form-label">Select Courses:</label>


            {% for course in courses %}


            <input class="form-check-input" type="checkbox" name="courses"
id="course_{{course.course_id}}"
                value="{{course.course_id}}">
            <label class="form-check-label"
for="course_{{course.course_id}}">
                {{course.cname}}
```

```
            </label>


            {% empty %}
            <p>No courses available. Please add courses first.</p>
            {% endfor %}
        </div>


        <div class="mb-3">
            <label for="qualification" class="form-label">Select
Qualification:</label>
            <select id="qualification" name="qualification" required
class="form-control">
                <option value="">Select Qualification</option>
                <option value="B.Tech">B.Tech</option>
                <option value="M.Tech">M.Tech</option>
                <option value="BCA">BCA</option>
                <option value="MCA">MCA</option>
            </select>
        </div>


        <input type="submit" value="Register Student" class="btn
btn-success">
    </form>
```

```
<hr>

{% if students %}

<table class="table table-dark">

    <thead>

        <tr>
            <td>Roll</td>
            <td>Student Name</td>
            <td>Email</td>
            <td>Address</td>
            <td>Gender</td>
            <td>Qualification</td>
            <td>Courses</td>

        </tr>

    </thead>

    <tbody>
        {% for student in students %}
```

```
            <tr>
                <td>{{student.roll}}</td>
                <td>{{student.name}}</td>
                <td>{{student.email}}</td>
                <td>{{student.address}}</td>
                <td>{{student.gender}}</td>
                <td>{{student.qualification}}</td>
                <td>{{student.courses.all | join:", "}}</td>

            </tr>
            {% endfor %}
        </tbody>
    </table>
    {% endif %}
</div>
{% endblock %}
```

Implementing the **STUDENT LIST** functionality in **addcourse.html**:

- Add the following view function inside **StudentApp/views.py** file:

```
def course_student_list_view(request, course_id):
    course = Course.objects.get(course_id=course_id)
```

```
    students = course.student_set.all()  # Get all students enrolled
in this course

    return render(request, 'coursestudentlist.html',
context={'course': course, 'students': students})
```

- **Update urls.py**
    - Add a URL pattern for the student list view:

```
from django.urls import path

from . import views

urlpatterns = [

    path('', views.home_view, name='home'),

    path('addcourse/', views.add_course_view, name='addcourse'),

    path('registerstudent/', views.register_student_view,
name='registerstudent'),

    path('course/<int:course_id>/students/',

         views.course_student_list_view, name='course_student_list')

]
```

- Create **coursestudentlist.html** inside **StudentApp/templates** folder

coursestudentlist.html

```
{% extends "base.html" %}


{% block title_block %} Students in {{ course.cname }} {% endblock
%}


{% block main_block %}
```

```
<h3 class="text-center">Students Enrolled in {{ course.cname }}</h3>


<div class="container">
    {% if students %}
    <table class="table table-dark">
        <thead>
            <tr>
                <th>Roll</th>
                <th>Name</th>
                <th>Email</th>
                <th>Address</th>
                <th>Gender</th>
                <th>Qualification</th>
            </tr>
        </thead>
        <tbody>
            {% for student in students %}
            <tr>
                <td>{{ student.roll }}</td>
                <td>{{ student.name }}</td>
                <td>{{ student.email }}</td>
                <td>{{ student.address }}</td>
                <td>{{ student.gender }}</td>
```

```
                <td>{{ student.qualification }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
    {% else %}
    <p class="text-center">No students enrolled in this course yet.</p>
    {% endif %}
    <a href="{% url 'addcourse' %}" class="btn btn-primary">Back to Courses</a>
</div>
{% endblock %}
```

- **Update addcourse.html**
  - Modify the "STUDENT LIST" button to link to the new view:

```
<td>
     <a href="#" class="btn btn-primary">UPDATE</a>
     <a href="#" class="btn btn-success">DELETE</a>
     <a href="{% url 'course_student_list' course.course_id %}" class="btn btn-danger">STUDENT LIST</a>
</td>
```

Implementing the **DELETE** functionality in **addcourse.html:**

- This will allow deleting a course from the database.

- **Update views.py**
- Add a view function to handle course deletion:

```
def delete_course_view(request, course_id):
    course = Course.objects.get(course_id=course_id)
    if request.method == "POST":
        course.delete()
        messages.success(
            request, f"Course {course.cname} deleted successfully")
        return redirect('addcourse')
    return render(request, 'deletecourse.html', context={'course':
course})
```

- **Update urls.py**
- Add a URL pattern for the delete view:

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

urlpatterns = [

    path('', views.home_view, name='home'),
    path('addcourse/', views.add_course_view, name='addcourse'),
    path('registerstudent/', views.register_student_view,
name='registerstudent'),
    path('course/<int:course_id>/students/',
```

```
            views.course_student_list_view,
        name='course_student_list'),

            path('course/<int:course_id>/delete/',

                views.delete_course_view, name='delete_course'),

        ]
```

- Inside **StudentApp/templates/**, create a new file called deletecourse.html:

deletecourse.html

```
{% extends "base.html" %}


{% block title_block %} Delete {{ course.cname }} {% endblock %}


{% block main_block %}

<h3 class="text-center">Confirm Deletion</h3>


<div class="container">

    <p>Are you sure you want to delete the course <strong>{{ course.cname
}}</strong>?</p>

    <form method="POST">

        {% csrf_token %}

        <input type="submit" value="Confirm Delete" class="btn
btn-danger">
```

```
        <a href="{% url 'addcourse' %}" class="btn
btn-secondary">Cancel</a>

    </form>

</div>

{% endblock %}
```

- Update **addcourse.html**
    - Modify the "**DELETE**" button to link to the delete view:

```
<td>

        <a href="#" class="btn btn-primary">UPDATE</a>

        <a href="{% url 'delete_course' course.course_id %}" class="btn
    btn-success">DELETE</a>

        <a href="{% url 'course_student_list' course.course_id %}"
    class="btn btn-danger">STUDENT LIST</a>

</td>
```

Implementing the **UPDATE** functionality in **addcourse.html:**

- This will allow updating the details of an existing course.
    - **Update views.py**
    - Add a view function to handle course updates:

```
def update_course_view(request, course_id):

    course = Course.objects.get(course_id=course_id)

    if request.method == "POST":

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

        fee = int(request.POST.get('fee'))

        duration = request.POST.get('duration')
```

```
        # Validation
        if not cname:
            messages.error(request, "Course name is mandatory")
        elif
Course.objects.filter(cname=cname).exclude(course_id=course_id).exists():
            messages.error(request, "Course Name is already registered")
        elif not fee or fee <= 0:
            messages.error(request, "Fee must be a positive integer")
        else:
            course.cname = cname
            course.fee = fee
            course.duration = duration
            course.save()
            messages.success(request, f"Course {cname} updated
successfully")
            return redirect('addcourse')


    return render(request, 'updatecourse.html', context={'course':
course})
```

- **Update urls.py**
    - Add a URL pattern for the update view:

```
from django.urls import path

from . import views


urlpatterns = [


    path('', views.home_view, name='home'),

    path('addcourse/', views.add_course_view, name='addcourse'),

    path('registerstudent/', views.register_student_view,
name='registerstudent'),

    path('course/<int:course_id>/students/',

         views.course_student_list_view, name='course_student_list'),

    path('course/<int:course_id>/delete/',

         views.delete_course_view, name='delete_course'),

    path('course/<int:course_id>/update/',

         views.update_course_view, name='update_course'),

]
```

- Create **updatecourse.html**
    - Inside **StudentApp/templates/**, create a new file called updatecourse.html:

updatecourse.html

```
{% extends "base.html" %}


{% block title_block %} Update {{ course.cname }} {% endblock %}
```

```
{% block main_block %}

<h3 class="text-center">Update Course Details</h3>


<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label for="cname" class="form-label">Course Name:</label>

            <input type="text" id="cname" name="cname" value="{{
course.cname }}" required class="form-control">

        </div>

        <div class="mb-3">

            <label for="fee" class="form-label">Course Fee:</label>

            <input type="number" id="fee" name="fee" value="{{ course.fee
}}" required class="form-control">

        </div>

        <div class="mb-3">

            <label for="duration" class="form-label">Course
Duration:</label>

            <input type="text" id="duration" name="duration" value="{{
course.duration }}" required

                class="form-control">

        </div>
```

```
        <input type="submit" value="Update Course" class="btn
btn-primary">

        <a href="{% url 'addcourse' %}" class="btn
btn-secondary">Cancel</a>

    </form>

</div>

{% endblock %}
```

- Update **addcourse.html**
    - Modify the "**UPDATE**" button to link to the update view:

```
<td>

   <a href="{% url 'update_course' course.course_id %}" class="btn
btn-primary">UPDATE</a>

  <a href="{% url 'delete_course' course.course_id %}" class="btn
btn-success">DELETE</a>

  <a href="{% url 'course_student_list' course.course_id %}"
class="btn btn-danger">STUDENT LIST</a>

</td>
```