Problem 1: Set a cookie with a username on login and retrieve it on the dashboard page.

Task:

- **Create a login form (no authentication, just dummy username).**
- **On successful login, set a cookie named username.**
- **Create a dashboard view that reads the cookie and displays the username**

views.py

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

def login_view(request):
   if request.method == 'POST':
       username = request.POST.get('username')
       response = redirect('dashboard')
       response.set_cookie('username', username)
       return response
   return render(request, 'login.html')

def dashboard(request):
   username = request.COOKIES.get('username')
   return HttpResponse(f"Welcome, {username}”)
```

Problem 2: Store and display visit count using Django sessions.

views.py

```
def visit_count(request):
   count = request.session.get('count', 0)
   count += 1
   request.session['count'] = count
   return HttpResponse(f"You have visited this page {count} times.”)
```

Problem 3: Create a view to register a new user using Django's User model

views.py

```
from django.contrib.auth.models import User
from django.shortcuts import render, redirect

def register_user(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        User.objects.create_user(username=username, password=password)
        return redirect('login')
    return render(request, ‘register.html')
```

register.html

```
<form method="post">
  {% csrf_token %}
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <button type="submit">Register</button>
</form>
```

Problem: Write a Django view that sets a cookie and session for a logged-in user and another view that reads and deletes them.

Views.py

```
from django.http import HttpResponse


def set_cookie_session(request):

    response = HttpResponse("Cookie and session set.")

    response.set_cookie('user_name', 'kamalpreet', max_age=3600)

    request.session['user_id'] = 123

    return response


def get_cookie_session(request):
    username = request.COOKIES.get('user_name')

    user_id = request.session.get('user_id')

    return HttpResponse(f"Username from cookie: {username}, User ID
from session: {user_id}")


def delete_cookie_session(request):

    response = HttpResponse("Cookie and session deleted.")

    response.delete_cookie('user_name')

    try:

        del request.session['user_id']

    except KeyError:

        pass

    return response
```

Problem: Create a view to register users using Django's User model.

```
# views.py

from django.contrib.auth.models import User
from django.http import HttpResponse

def register_user(request):
    if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password']
        email = request.POST['email']
        user = User.objects.create_user(username=username,
password=password, email=email)
        return HttpResponse("User created successfully.")
    return HttpResponse("Send a POST request with username, password,
and email.”)
```

form.py

```
<form method="post">
```

```
  {% csrf_token %}
  <input type="text" name="username" placeholder="Username">
  <input type="email" name="email" placeholder="Email">
  <input type="password" name="password" placeholder="Password">
  <input type="submit" value="Register">
</form>
```

Problem: Configure URLs and views for login and logout.

urls.py:

```
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/',
auth_views.LoginView.as_view(template_name='login.html'),
name='login'),
    path('logout/', auth_views.LogoutView.as_view(next_page='login'),
name='logout'),
]
```

login.html:

```
<form method="post">
  {% csrf_token %}
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <button type="submit">Login</button>
</form>
```

Problem: Create a view that only logged-in users can access, and greet them with their username.

Solution:

```
from django.contrib.auth.decorators import login_required
```

from django.http import HttpResponse

@login_required
def dashboard(request):
return HttpResponse(f"Welcome, {request.user.username}!")
In settings.py, make sure you define:

LOGIN_URL = '/login/'
LOGIN_REDIRECT_URL = '/dashboard/' # Optional
LOGOUT_REDIRECT_URL = '/login/'   # Optional

Problem: Create two models `Author` and `Book`. An author can write multiple books.

Solution:

```
# models.py

from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    published_date = models.DateField()
    author = models.ForeignKey(Author,
on_delete=models.CASCADE)

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

Working with Migrations

Problem: Write the commands to create and apply migrations for the above models.

Solution:

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

Using the Django Shell to Explore Models

a. Insert Records

```
# python manage.py shell

from myapp.models import Author, Book
a = Author.objects.create(name='Kamalpreet Kaur')
Book.objects.create(title='Django for Beginners',
published_date='2023-01-01', author=a)
```

b. Update Records

```
book = Book.objects.get(title='Django for Beginners')
book.title = 'Advanced Django'
book.save()
```

c. Delete Records

```
book = Book.objects.get(title='Advanced Django')
book.delete()
```

Problem: Perform the following ORM queries:

- Get all books
- Filter books by author name
- Count number of books

```
books = Book.objects.all()
books_by_kaur = Book.objects.filter(author__name='Kamalpreet
Kaur')
book_count = Book.objects.count()
```

Models Using Foreign Keys

```
author = models.ForeignKey(Author, on_delete=models.CASCADE)
```

```
author_books = Author.objects.get(name='Kamalpreet
Kaur').book_set.all()
```

Problem: Register models in admin panel.

Solution:

```
# admin.py

from django.contrib import admin
from .models import Author, Book

admin.site.register(Author)
admin.site.register(Book)
```

Problem: Create a new user and add to a group.

Solution (shell):

```
from django.contrib.auth.models import User, Group

user = User.objects.create_user('kamal', 'email@example.com',
'password123')
group = Group.objects.create(name='Editors')
user.groups.add(group)
```

Problem: Give a user permission to change a model.

```
from django.contrib.auth.models import Permission

perm = Permission.objects.get(codename='change_book')
user.user_permissions.add(perm)
```

Problem: Create a simple form with `GET` and `POST` methods and display submitted data.

Solution:

python
Copy code
```
# views.py

from django.http import HttpResponse
from django.shortcuts import render

def basic_form_view(request):
    if request.method == "POST":
        name = request.POST.get('name')
        return HttpResponse(f"Received via POST: {name}")
    elif request.method == "GET":
        return render(request, 'basic_form.html')
```
Template: basic_form.html

html
Copy code
```
<form method="post">
  {% csrf_token %}
  <input type="text" name="name" placeholder="Enter your
name">
  <input type="submit" value="Submit">
</form>
```

Problem: Build a form using Django's `forms.Form` class.

Solution:

```
# forms.py

from django import forms

class NameForm(forms.Form):
    name = forms.CharField(label='Your Name', max_length=100)

# views.py

from .forms import NameForm

def name_form_view(request):
    if request.method == 'POST':
        form = NameForm(request.POST)
        if form.is_valid():
            name = form.cleaned_data['name']
            return HttpResponse(f"Hello, {name}")
    else:
        form = NameForm()
    return render(request, 'name_form.html', {'form': form})
```

Template: name_form.html

```
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Submit</button>
</form>
```

Django automatically protects against CSRF if you use `{% csrf_token %}` in your template.

CSRF Error Example (if missing token):

403 Forbidden — CSRF token missing or incorrect.

Fix: Always include `{% csrf_token %}` inside your `<form method="post">`.

Problem: After successful POST submission, redirect to a success page.

Solution:

```
# views.py

from django.shortcuts import redirect

def post_redirect_view(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        request.session['name'] = name  # temporary storage
        return redirect('thank_you')
    return render(request, 'post_form.html')

def thank_you(request):
    name = request.session.get('name')
    return HttpResponse(f"Thank you, {name}!")
```

urls.py

```
path('submit/', post_redirect_view, name='submit'),
path('thank-you/', thank_you, name='thank_you'),
```

Problem: Add validation to ensure name is longer than 3 characters.

Solution:

```
# forms.py

class NameForm(forms.Form):
    name = forms.CharField(label='Your Name', max_length=100)

    def clean_name(self):
        data = self.cleaned_data['name']
        if len(data) < 4:
            raise forms.ValidationError("Name must be at
least 4 characters long.")
        return data
```

Django will show validation errors automatically in the template if you include:

```
html

{{ form.errors }}
```

Using Django Variables, If-Else, and Loops

python
Copy code
```
# views.py

def student_list(request):
    students = ['Amit', 'Bhavna', 'Charan']
    return render(request, 'students.html', {'students':
students, 'course': 'Python'})
```
html
Copy code
```
<!-- templates/students.html -->
<h2>Course: {{ course }}</h2>

{% if students %}
<ul>
  {% for student in students %}
```

```
    <li>{{ student }}</li>
  {% endfor %}
</ul>
{% else %}
<p>No students enrolled.</p>
{% endif %}
```

Using Template Tags

Common Built-in Tags:

html

```
{% now "Y-m-d H:i" %}
{{ my_list|length }}
{{ title|upper }}
```

Example:

html

```
<h4>Current Time: {% now "d M Y H:i" %}</h4>
```

Dynamic Templates in Django

Problem: Render a blog post with title and body from view.

python

```
# views.py

def blog_post(request):
    context = {'title': 'My First Blog', 'body': 'This is a
blog post content.'}
    return render(request, 'blog_post.html', context)
```

html

```
<!-- templates/blog_post.html -->
<h1>{{ title }}</h1>
<p>{{ body }}</p>
```

Working with Template Inheritance

Problem: Create a `base.html` and extend it in child templates.

html

```
<!-- templates/base.html -->
<html>
<head>
    <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
    <header><h2>Welcome to My Site</h2></header>
    <div>
```

Dynamic Content View

Your blog app needs to show blog details using `/blog/42/`, where `42` is the blog ID. How will you:

- Create a view that fetches blog content by ID?
- Return a 404 if the blog does not exist?

Hint: Use `get_object_or_404()` in the view.

Invalid URL Parameters

A user enters a URL like `/product/abc/` instead of `/product/123/`, and your app crashes.

- How would you handle such cases using regular expressions in URLs?

Hint: Use `re_path()` and validate with `\d+`.

Custom Error Page

You deployed your app, but users complain about an unfriendly 404 page.

- How would you show a custom HTML page instead of the default 404?

Hint: Define a custom `handler404` and return a template.

Login Required

You want only logged-in users to access the `/dashboard/` view.

- How do you enforce authentication in views?

Hint: Use `@login_required`.

Dynamic Navigation Menu

You want the navbar to show "Login" if the user is not authenticated and "Logout" if they are.

- How would you handle this in a Django template?

Hint: Use `{% if user.is_authenticated %}`.

Template Inheritance

You have five pages, all sharing a header/footer layout.

- How will you avoid duplicating code across all HTML files?

Hint: Use `base.html` and `{% extends %}`.

Repeated Content Rendering

You need to list product cards dynamically using loop in a grid.

- How would you loop over a context dictionary in the template?

Hint: Use `{% for %}` loop with dynamic context.

Debugging View Crash

Your view suddenly throws `KeyError: 'name'` when submitted via form.

- What are possible causes and how do you debug this?

Hint: Check if `request.POST.get('name')` is being used properly.

Failing Unit Test

A unit test checking `/hello/` view returns 500 error, but manually it works.

- How do you investigate and fix this?

Hint: Log the response content and status code in test.

Template Not Found

You are getting `TemplateDoesNotExist: home.html`.

- What could be the causes and steps to resolve it?

Hint: Check `TEMPLATES['DIRS']`, folder path, and file extension.

Form Doesn't Save Data

You built a contact form, but it never stores user data.

- What are possible reasons, and how do you debug this?

Hint: Check form validation and `form.save()` call.

CSRF Token Missing

Your POST form keeps failing with `403 Forbidden`.

- What's likely missing, and how do you resolve it?

Hint: Add `{% csrf_token %}` in the template.

: Form Field Error Display

You want to show users which field is incorrect when submitting.

- How do you display individual field errors in the template?

Hint: Use `{{ form.field.errors }}` or `{{ form.non_field_errors }}`.

Prevent Duplicate Submissions

Users are accidentally resubmitting a form on page refresh.

- How would you implement a POST-Redirect-GET pattern to solve this?

Hint: Use `redirect()` after `POST` processing.

Conditional Form Rendering

You want to show additional form fields only if the user selects "Yes" from a dropdown.

- How can you dynamically handle this scenario?

Cookie Management and URL Routing

Your task is to manage session-based preferences using cookies.

Task:
Create views and corresponding URL mappings for:

- Setting a cookie named `preferred_theme` with a user-selected value.
- Getting the value of the cookie.
- Deleting the cookie.

Requirements:

- Explain how you would use the `HttpResponse` object to set and delete cookies.
- Use Django's `path()` and `re_path()` to create appropriate URL patterns and validate inputs using regex.

Building a Book Review Application with Authentication and Ratings Task Description:
Create a Django app that allows users to register/login, browse books, and submit reviews with ratings. Ensure secure voting and one-review-per-user enforcement.

Models:

- **User**: Custom user model with username, email, password, phone, and role.
- **Book**: Title, author, genre, description, published date.
- **Review**: Rating (1–5), review text, user (ForeignKey to User), book (ForeignKey to Book), and timestamp.

Templates:

- Base layout with navbar, footer, and consistent layout.
- Book detail template with review form and existing reviews.
- Auth templates for login/registration.

Forms:

- **RegistrationForm**: Username, email, password, confirm password, phone, role.
- **LoginForm**: Username, password.
- **ReviewForm**: Rating (1-5 dropdown), review text; dynamically tied to the book; ensure user can only review once per book.

Functionality:

- Authenticated users can review; anonymous users are redirected to login.

- Prevent duplicate reviews for the same book by the same user.
- Validate input (e.g., email format, rating range).

— Event Management System with Role-Based Access (CO3, CO4)

Task Description:
Design a system where admins can create events and regular users can register to attend.

Models:

- **User**: Custom user model with roles — `admin` or `attendee`.
- **Event**: Title, location, start/end datetime, created_by (admin).
- **Registration**: Links user to event, with a timestamp.

Views and Access Control:

- Admins can create, update, delete events.
- Attendees can only view and register.
- Unauthorized access to restricted views should redirect to login or 403 page.

Forms:

- EventForm (Admin only).
- RegistrationForm (auto-filled user and event).

Testing:

- Write a test to ensure non-admins cannot access the event creation view.