USN | | | | | | | | | |

Internal Assessment Test 3 – July 2024

Sub	**Full Stack Development**					Sub code	**21CS62**	Branch	**CSE**
Date	**27.07.2024**	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	**VI Sem (B, C)**		**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1(a)	How can developers ensure security and privacy when implementing cookies and sessions?	[5]	CO4	L2
1(b)	Implement a basic login system in Python using Django that checks a username and password against a predefined list of users.	[5]	CO4	L3
2(a)	What are MIME types.How can you set the MIME type of a response in a Django view? Provide a code example.	[5]	CO4	L2,L3
2(b)	Write a django Application to generate a pdf file from the given model.	[5]	CO4	L3
3(a)	Define the following terms: 1. XHTMLHttpRequest and Response 2. JSON 3. iFrames	[3]	CO5	L2
3(b)	How do you integrate jQuery UI Autocomplete with a Django form to provide suggestions based on user input?	[7]	CO5	L3

4(a)	How do you create a simple RSS feed in Django using the Syndication Feed Framework? Provide a step-by-step example including necessary code snippets.	[5]	CO4	L3
4(b)	How do you create a basic XML sitemap in Django using the Sitemap Framework? Provide an example including necessary code snippets and configurations.	[5]	CO4	L3
5(a)	How can you use jQuery to send an AJAX GET request to a Django view and display the returned data in an HTML element?	[5]	CO5	L2
5(b)	How can you use the `{% static %}` template tag in Django to include static files in your HTML templates?	[5]	CO5	L2
6	Write an Application in django to generate a CSV file with all files ie. models.py, views.py, urls.py, index.py. OR How do you create a simple user profile page in Django that displays information about the currently logged-in user?	[10]	CO4	L3

CI CCI HOD

Internal Assessment Test 3

Solution

Sub:	**Full Stack Development**					Sub Code:	**21CS62**	Branch :	CSE
Date:	30/7/2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VI- B & C		

Question number	Question and solution
1	**a)How can developers ensure security and privacy when implementing cookies and sessions?** **Solution**: To ensure security and privacy when implementing cookies and sessions, developers should follow these best practices: 1. Use Secure Cookies: ○ Secure Flag: Set the Secure flag to ensure cookies are only sent over HTTPS. ○ HttpOnly Flag: Use the HttpOnly flag to prevent JavaScript access, reducing XSS attack risks. ○ SameSite Attribute: Implement the SameSite attribute to prevent CSRF attacks by restricting cross-site requests. 2. Session Management: ○ Session Expiry: Set short expiration times for sessions and renew them regularly to minimize hijacking risks. ○ Regenerate Session IDs: Regenerate session IDs upon login and periodically to prevent session fixation. ○ Server-Side Storage: Store sessions securely on the server side, avoiding sensitive data in cookies. 3. Encrypt Sensitive Data and Use HTTPS: ○ Encryption: Encrypt cookies containing sensitive data using strong encryption algorithms. ○ HTTPS: Always use HTTPS to encrypt data in transit, including cookies. By implementing these measures, developers can significantly enhance the security and privacy of their web applications. **b) Implement a basic login system in Python using Django that checks a username and password**

against a predefined list of users.

Solution:

To implement a basic login system in Python using Django that checks a username and password against a predefined list of users, follow these steps:

1. Set Up Django Project and Application

First, create a Django project and application:

2. Define Predefined Users

In your `myapp` application, define a list of predefined users. This can be done in a separate file like `users.py`:

```
# myapp/users.py

USERS = {

    'john_doe': 'password123',

    'jane_smith': 'mypassword',

}
```

3. Create a Login Form

Create a simple form for the login in `forms.py`:

```
# myapp/forms.py

from django import forms


class LoginForm(forms.Form):

    username = forms.CharField(max_length=100)

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

4. Create Views for Login

Define views to handle the login form submission in `views.py`:

```
# myapp/views.py

from django.shortcuts import render, redirect

from django.http import HttpResponse

from .forms import LoginForm

from .users.py import USERS


def login_view(request):

    if request.method == 'POST':

        form = LoginForm(request.POST)

        if form.is_valid():

            username = form.cleaned_data['username']

            password = form.cleaned_data['password']

            if username in USERS and USERS[username] == password:

                request.session['username'] = username

                return redirect('home')

            else:

                return HttpResponse("Invalid login credentials")

    else:

        form = LoginForm()

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


def home_view(request):

    if 'username' in request.session:

        username = request.session['username']

        return HttpResponse(f'Welcome, {username}!')

    return redirect('login')
```

2

a) What are MIME types.How can you set the MIME type of a response in a Django view? Provide a code example.

Solution:

MIME (Multipurpose Internet Mail Extensions) types are standardized ways to indicate the nature and format of a file. They are used to specify the type of data being sent over the internet, ensuring that both the sender and receiver interpret the file correctly. For example, a MIME type might specify whether a file is a text document, an image, a video, etc. Common MIME types include:

- `text/html` for HTML documents
- `image/jpeg` for JPEG images
- `application/json` for JSON data
- `text/css` for CSS files

Setting the MIME Type in a Django View

In Django, you can set the MIME type of a response using the `HttpResponse` object by specifying the `content_type` parameter. This parameter determines the MIME type of the HTTP response.

Example Code

Here's an example of how you can set different MIME types in Django views

```
from django.http import HttpResponse

def html_response_view(request):
    html_content = "<html><body><h1>Hello, World!</h1></body></html>"
    return HttpResponse(html_content, content_type="text/html")

def json_response_view(request):
    json_content = '{"message": "Hello, World!"}'
    return HttpResponse(json_content, content_type="application/json")

def text_response_view(request):
    text_content = "Hello, World!"
    return HttpResponse(text_content, content_type="text/plain")
```

```
def csv_response_view(request):
    csv_content = "Name,Age\nAlice,30\nBob,25"
    return HttpResponse(csv_content, content_type="text/csv")
```

b) Write a django Application to generate a pdf file from the given model.

Solution:

Create a Django Model

```
# models.py

from django.db import models

class ResearchProposal(models.Model):
    title = models.CharField(max_length=200)
    abstract = models.TextField()
    keywords = models.CharField(max_length=200)
    area = models.CharField(max_length=100)
    full_paper = models.FileField(upload_to='papers/', null=True, blank=True)

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

Create a Django View to Generate PDF

```
# views.py

from django.shortcuts import get_object_or_404
from django.http import HttpResponse
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from .models import ResearchProposal

def generate_pdf(request, proposal_id):
    proposal = get_object_or_404(ResearchProposal, id=proposal_id)

    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = f'attachment; filename="{proposal.title}.pdf"'

    p = canvas.Canvas(response, pagesize=letter)
    p.drawString(100, 750, f'Title: {proposal.title}')
    p.drawString(100, 725, f'Abstract: {proposal.abstract}')
    p.drawString(100, 700, f'Keywords: {proposal.keywords}')
    p.drawString(100, 675, f'Area of Research: {proposal.area}')

    p.showPage()
    p.save()
```

	return response
3	**a)** Define the following terms: 1. XHTMLHttpRequest and Response 2. JSON 3. iFrames **Solution:** 1. **XMLHttpRequest:** XMLHttpRequest (XHR) is a JavaScript object that allows web pages to make HTTP requests to servers and load data asynchronously without reloading the entire page. It's fundamental for making AJAX (Asynchronous JavaScript and XML) requests. 2. **JSON (JavaScript Object Notation):** ● Definition: A lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. ● Explanation: JSON is used to represent data structures and objects in a text format, and it is commonly used for transmitting data in web applications. 3. **iFrames (Inline Frames):** ● **Definition:** HTML elements that allow you to embed another HTML document within the current document. ● **Explanation:** iFrames are used to embed content from other sources, such as videos, maps, or other web pages, within a web page. Example: **<iframe src="https://www.example.com" width="600" height="400"></iframe>** **b)** How do you integrate jQuery UI Autocomplete with a Django form to provide suggestions based on user input? **Solution:** **1. Install and Configure jQuery and jQuery UI** First, ensure you have jQuery and jQuery UI included in your template. You can use CDN links for simplicity: <!DOCTYPE html> <html lang="en"> <head>

```
    <meta charset="UTF-8">
    <title>Autocomplete Example</title>
    <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
</head>
<body>
```

2. Create a Django Form

Create a Django form with a CharField for the input field you want to add autocomplete to:

```
# forms.py
from django import forms


class MyForm(forms.Form):
    my_field = forms.CharField(label='My Field', max_length=100)
```

3. Create a View to Serve Autocomplete Suggestions

Create a view in Django that will return JSON data based on the user's input:

```
# views.py
from django.http import JsonResponse
from django.shortcuts import render
from .forms import MyForm


def autocomplete_view(request):
    if 'term' in request.GET:
        qs = MyModel.objects.filter(name__icontains=request.GET.get('term'))
        names = list()
        for product in qs:
            names.append(product.name)
        return JsonResponse(names, safe=False)
    return render(request, 'my_template.html', {'form': MyForm()})
```

4. Configure URLs

Add URLs for your autocomplete view and form view

```
# urls.py
from django.urls import path
from .views import autocomplete_view


urlpatterns = [
    path('autocomplete/', autocomplete_view, name='autocomplete'),
    path('my_form/', autocomplete_view, name='my_form'),
]
```

5. Update Your Template

Update your template to include the form and the jQuery UI Autocomplete initialization:

```
<!-- my_template.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Autocomplete Example</title>
    <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

	```
	    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
	</head>
	<body>
	    <form method="post">
	        {% csrf_token %}
	        {{ form.as_p }}
	        <input type="submit" value="Submit">
	    </form>

	    <script>
	        $(document).ready(function() {
	            $('#id_my_field').autocomplete({
	                source: "{% url 'autocomplete' %}",
	                minLength: 2,
	            });
	        });
	    </script>
	</body>
	</html>
	```
4	**a)** How do you create a simple RSS feed in Django using the Syndication Feed Framework? Provide a step-by-step example including necessary code snippets. **Solution**: Step - 1: Create project and App. **Step 2: Define Your Models**

Define a model for your blog posts in `blog/models.py`

```
# blog/models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField('date published')

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

Step 3: Create a Feed Class

Create a feed class in `blog/feeds.py` that will generate the RSS feed:

```
# blog/feeds.py
from django.contrib.syndication.views import Feed
from django.urls import reverse
from .models import Post

class LatestPostsFeed(Feed):
    title = "My Blog"
    link = "/blog/"
    description = "Updates on new blog posts."

    def items(self):
        return Post.objects.order_by('-pub_date')[:5]

    def item_title(self, item):
        return item.title

    def item_description(self, item):
        return item.content

    def item_link(self, item):
        return reverse('blog:post_detail', args=[item.pk])
```

Step 4: Configure URLs

Add a URL pattern for the feed in your `blog/urls.py`

```
# blog/urls.py
from django.urls import path
from . import views
from .feeds import LatestPostsFeed

app_name = 'blog'

urlpatterns = [
    path('', views.index, name='index'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('feed/', LatestPostsFeed(), name='post_feed'),
]
```

Step 5: Create Views and Templates

Create basic views and templates to display your blog posts:

```
# blog/views.py
from django.shortcuts import render, get_object_or_404
from .models import Post

def index(request):
    latest_posts = Post.objects.order_by('-pub_date')[:5]
    context = {'latest_posts': latest_posts}
    return render(request, 'blog/index.html', context)

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

Create templates `index.html` and `post_detail.html` in `blog/templates/blog/`:

b) How do you create a basic XML sitemap in Django using the Sitemap Framework? Provide an example including necessary code snippets and configurations.

Solution:

```
# blog/models.py

from django.db import models
```

```
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField('date published')
    def __str__(self):
        return self.title
# blog/sitemaps.py
from django.contrib.sitemaps import Sitemap
from .models import Post
class PostSitemap(Sitemap):
    changefreq = "daily"
    priority = 0.8
    def items(self):
        return Post.objects.all()
    def lastmod(self, obj):
        return obj.pub_date
```

Configure URLs

Add a URL pattern for the sitemap in your `blog/urls.py`

```
# blog/urls.py
from django.urls import path
from . import views
from .sitemaps import PostSitemap
from django.contrib.sitemaps.views import sitemap
```

```
sitemaps = {
    'posts': PostSitemap,
}


app_name = 'blog'


urlpatterns = [
    path('', views.index, name='index'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps}, name='django.contrib.sitemaps.views.sitemap'),
]


# blog/views.py
from django.shortcuts import render, get_object_or_404
from .models import Post
def index(request):
    latest_posts = Post.objects.order_by('-pub_date')[:5]
    context = {'latest_posts': latest_posts}
    return render(request, 'blog/index.html', context)


def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

5	

a) How can you use jQuery to send an AJAX GET request to a Django view and display the returned data in an HTML element?

Solution:

First, create a Django view that will handle the GET request and return data. For this example, let's assume we have a simple view that returns a JSON response.

# views.py

```
from django.http import JsonResponse

def my_view(request):

    data = {

        'message': 'Hello, this is the data from the server!'

    }

    return JsonResponse(data)
```

<!-- templates/index.html -->

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>AJAX GET Request Example</title>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

</head>

<body>

    <h1>AJAX GET Request Example</h1>

    <button id="fetch-data">Fetch Data</button>
    <div id="data-container"></div>


    <script>
        $(document).ready(function(){
            $('#fetch-data').click(function(){
                $.ajax({
                    url: "{% url 'my_view' %}",
                    type: 'GET',
                    success: function(response) {
                        $('#data-container').html('<p>' + response.message + '</p>');
                    },
                    error: function(xhr, status, error) {
                        $('#data-container').html('<p>Error: ' + error + '</p>');
                    }
                });
            });
        });
    </script>
</body>
</html>
```

# urls.py

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

urlpatterns = [

    path('my-view/', my_view, name='my_view'),

]
```

# views.py

```
from django.shortcuts import render

def index(request):

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

b) How can you use the **`{% static %}`** template tag in Django to include static files in your HTML templates?

Solution:

The `{% static %}` template tag in Django is used to include static files in your HTML templates. Static files include CSS, JavaScript, images, and other files that don't change dynamically and are used across your site.

Here's a step-by-step guide on how to use the `{% static %}` template tag in Django:

Step 1: Configure Static Files in `settings.py`

Ensure you have the static files settings configured in your `settings.py` file:

```
# settings.py
import os

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

Step 2: Create a Static Directory

Create a `static` directory in your project or app directory. Inside this `static` directory, you can organize your static files into subdirectories such as `css`, `js`, `images`, etc.

Step 3: Load the Static Template Tag in Your Template

In your HTML template, you need to load the static template tag library at the top of the file. Then you can use the `{% static %}` tag to include your static files.

```html
<!-- templates/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Static Files Example</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'my_app/css/style.css' %}">
    <script src="{% static 'my_app/js/script.js' %}"></script>
</head>
<body>
    <h1>Welcome to the Static Files Example</h1>
    <img src="{% static 'my_app/images/logo.png' %}" alt="Logo">
</body>
</html>
```

6

Write an Application in django to generate a CSV file with all files ie. models.py, views.py, urls.py, index.py.

Solution:

```python
# csvapp/models.py
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

# csvapp/views.py
import csv
from django.http import HttpResponse
from .models import Person

def generate_csv(request):
    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="people.csv"'

    writer = csv.writer(response)
    writer.writerow(['First Name', 'Last Name', 'Email'])

    people = Person.objects.all().values_list('first_name', 'last_name', 'email')
    for person in people:
        writer.writerow(person)

    return response

# csvapp/urls.py

from django.urls import path
from .views import generate_csv

urlpatterns = [
    path('generate_csv/', generate_csv, name='generate_csv'),
]

<!-- csvapp/templates/index.html →

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>CSV Generator</title>
</head>
<body>
    <h1>Generate CSV</h1>
    <a href="{% url 'generate_csv' %}">Download CSV</a>
</body>
</html>
```

OR

How do you create a simple user profile page in Django that displays information about the currently logged-in user?

Solution:

```
# myapp/models.py

from django.db import models
from django.contrib.auth.models import User

class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(blank=True, null=True)
    location = models.CharField(max_length=100, blank=True, null=True)

    def __str__(self):
        return self.user.username

# myapp/views.py
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User

@login_required
def profile(request):
    user = request.user
    return render(request, 'myapp/profile.html', {'user': user})
```

<!-- myapp/templates/myapp/profile.html →

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p><strong>Username:</strong> {{ user.username }}</p>
    <p><strong>Email:</strong> {{ user.email }}</p>
    {% if user.userprofile %}
        <p><strong>Bio:</strong> {{ user.userprofile.bio }}</p>
        <p><strong>Location:</strong> {{ user.userprofile.location }}</p>
    {% endif %}
</body>
</html>
```

# myapp/urls.py

```
from django.urls import path
from . import views
urlpatterns = [
    path('profile/', views.profile, name='profile'),
]
```

# project/urls.py

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```