FULLSTACK DEVELOPMENT MODEL QUESTION PAPER WITH ANSWERS

VTUCSE21

Module-01

Q.01

a. Define: (4 Marks)

a) **Web Frameworks with example.**
b) **URLs.**

b. Explain features of Django. (8 Marks)

Ans:

1a. Define: (4 Marks)

Web Frameworks with example:

- Web frameworks are software tools designed to support the development of web applications, including web services, web resources, and web APIs.
- Examples include Django for Python, Ruby on Rails for Ruby, and Laravel for PHP.

URLs:

- URLs (Uniform Resource Locators) are the addresses used to access resources on the internet.
- A URL consists of the protocol (e.g., http), domain name (e.g., www.example.com), and the path to the resource (e.g., /path/to/resource).

1b. Explain features of Django: (8 Marks)

1. Admin Interface:

- Provides a built-in admin interface to manage application data.
- Automatically generated and customizable.

- **Example:** Add, update, and delete records via a web interface.

2. ORM (Object-Relational Mapping):

- Allows interaction with databases using Python code instead of SQL.
- Simplifies database operations and increases productivity.
- **Example:** Use Model.objects.all() to retrieve all records from a table.

3. MTV Architecture:

- Model-Template-View architecture separates concerns within the application.
- **Model:** Manages data and business logic.
- **Template:** Handles presentation and user interface.
- **View:** Manages the control flow and processes user input.

4. Security:

- Includes built-in protections against common attacks like SQL injection, XSS, and CSRF.
- Regular security updates and best practices.
- **Example:** CSRF tokens are automatically included in forms.

5. Scalability:

- Designed for scalability and rapid development.
- Supports high-traffic sites with ease.
- **Example:** Can handle increased load by scaling out with additional servers.

1c. Explain MVC Architecture: (8 Marks)

1. **Model:**
    - Represents the data and business logic.
    - Manages data storage and retrieval.
    - **Example:** In a blogging application, the Post model represents blog posts.

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
```

2. **View:**

- Displays the data (user interface).
- Presents data to the user and handles user interaction.
- **Example:** The template that displays the list of blog posts.

```
def show_posts(request):
    posts = Post.objects.all()
    return render(request, 'posts.html', {'posts': posts})
```

3. **Controller:**
    - Handles user input and updates the Model and View accordingly.
    - Acts as an intermediary between Model and View.
    - **Example:** A function that processes form submissions and updates the blog post data.

```
def add_post(request):
    if request.method == 'POST':
        title = request.POST['title']
        content = request.POST['content']
        Post.objects.create(title=title, content=content)
        return redirect('show_posts')
```

Q.02

a. Define: (4 Marks)

a) **Django URL Configuration.**
b) **Loose Coupling.**

b. List out Django Exceptions & Errors with neat diagram. (8 Marks)

c. Develop a Django app that displays current date & time. (8 Marks)

2a. Define: (4 Marks)

Django URL Configuration:

- Django uses urls.py to route URLs to the appropriate views.
- Each URL pattern is associated with a view function or class.
- This mapping allows the application to handle different URL paths and execute the corresponding logic.
- **Example:** In urls.py, you might map the URL pattern path('current_time/', views.current_datetime) to the current_datetime view.

Loose Coupling:

- A design principle aimed at reducing dependencies between components.
- Makes the system more modular and easier to maintain.
- Each component can be modified or replaced without affecting others significantly.
- **Example:** In Django, views and models are loosely coupled. Changes in the model do not directly affect the view.

2b. List out Django Exceptions & Errors with neat diagram: (8 Marks)

Common Django Exceptions:

1. **ObjectDoesNotExist:**
    - Raised when an object is not found in the database.
    - **Example:** MyModel.objects.get(pk=1) raises ObjectDoesNotExist if no record with primary key 1 exists.

2. **MultipleObjectsReturned:**
    - Raised when a query returns multiple objects but only one was expected.
    - **Example:** MyModel.objects.get(name='example') raises MultipleObjectsReturned if more than one record with the name 'example' exists.

3. **FieldError:**
    - Raised for problems with model fields.
    - **Example:**Using a non-existent field in a query: MyModel.objects.filter(nonexistent_field='value').

4. **ValidationError:**
    - Raised when data validation fails.
    - **Example:** MyForm({'name': ''}) raises ValidationError if the name field is required but not provided.
5. **PermissionDenied:**
    - Raised when a user does not have the necessary permissions to access a resource.
    - **Example:** Manually raised in a view: from django.core.exceptions import PermissionDenied.

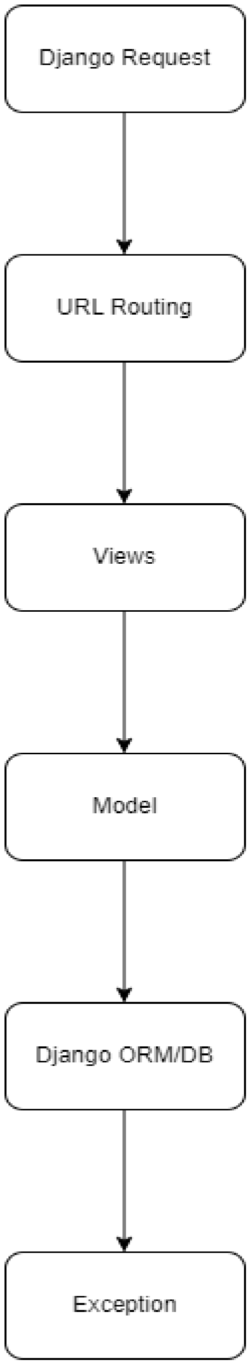


Fig: Django Request-Response Cycle

2c. Develop a Django app that displays current date & time: (8 Marks)

Step-by-Step Guide:

1. **Create a Django Project:**
    - django-admin startproject current_date
    - cd myproject

2. **Create a Django App:**
    - python manage.py startapp date_app

3. **Configure myapp in settings.py:**

```
INSTALLED_APPS = [

# ...

'date_app',

]
```

4. **Define the View in date_app/views.py:**

```
from django.http import HttpResponse

from django.utils import timezone

def current_datetime(request):

    now = timezone.now()

    html = f"<html><body>Current date and time: {now}</body></html>"

    return HttpResponse(html)
```

5. **Configure URL in myapp/urls.py:**

```
from django.urls import path

from .views import current_datetime

urlpatterns = [

    path('current_time/', current_datetime, name='current_datetime'),

]
```

6. **Include App URLs in current_date/urls.py:**

```
from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]
```

7. **Run the Server:**
    - python manage.py runserver

8. **Access the URL:**
    - Navigate to **http://localhost:8000/current_time/** to see the current date and time displayed.

Module 2

Q.03

a. Explain MVT Architecture. (10 Marks)

b. Explain Template Inheritance with an example. (10 Marks)

3a. Explain MVT Architecture (10 Marks)

Model-View-Template (MVT) Architecture:

Model:

- Manages the data and business logic of the application.
- Defines the structure of the data, including fields and behaviors.
- Interacts with the database via Django's ORM.
- **Example:**

```
from django.db import models
class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    published_date = models.DateField()
```

View:

- Handles the user interface logic.
- Receives HTTP requests and returns HTTP responses.
- Retrieves data from the model, processes it, and passes it to the template.
- **Example:**

```
from django.shortcuts import render

from .models import Book
```

```
def book_list(request):
    books = Book.objects.all()
    return render(request, 'book_list.html', {'books': books})
```

Template:

- Contains the HTML and presentation logic.
- Responsible for rendering the data passed from the view.
- Uses Django's templating language to include dynamic content.
- **Example:**

```
<!-- book_list.html -->
<!DOCTYPE html>
<html>
<head>
   <title>Book List</title>
</head>
<body>
   <h1>Books</h1>
   <ul>
      {% for book in books %}
         <li>{{ book.title }} by {{ book.author }} ({{ book.published_date }})</li>
      {% endfor %}
   </ul>
</body>
</html>
```

3b. Explain Template Inheritance with an example (10 Marks)

Template Inheritance:

- Template Inheritance allows you to define a base template and extend it in other templates.
- This helps in maintaining a consistent layout across different pages and avoids code duplication.

Example: base.html

```
<!DOCTYPE html>

<html>

<head>

   <title>{% block title %}My Site{% endblock %}</title>

</head>

<body>

   {% block content %}{% endblock %}

</body>

</html>
```

home.html:

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

{% block title %}Home{% endblock %}

{% block content %}

<h1>Welcome to my site</h1>

{% endblock %}
```

In this example:

- base.html is the base template that defines the common structure of the HTML document.
- home.html extends base.html and overrides the title and content blocks to provide specific content for the home page.

Key Points of Template Inheritance

1. **Reusability:**
    - **Base Template (base.html):** Contains the common layout and structure.
    - **Child Templates:** Extend the base template and override specific blocks to customize content.
2. **Maintaining Consistency:**
    - Changes to the base template automatically reflect across all child templates, ensuring a consistent layout.
3. **Avoiding Duplication:**
    - Common elements (like headers, footers) are defined once in the base template, avoiding redundancy.

Benefits:

- ❖ **Easy Updates:** Modify the base template to update the layout across all pages.
- ❖ **Clean Code:** Keeps template files clean and manageable by separating common structure from page-specific content.
- ❖ **Flexibility:** Allows specific sections of the layout to be customized as needed.

Q.04

a. Explain Django and Python Templates with an example. (10 Marks)

b. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event. (10 Marks)

4a. Explain Django and Python Templates with an example (10 Marks)

Django Templates:

- Django's templating engine allows you to create dynamic HTML by embedding Django template tags and filters within your HTML code.

Example:

```
<!-- template.html -->

<h1>Hello, {{ name }}!</h1>
```

```
# views.py

from django.shortcuts import render

def greet(request):

    context = {'name': 'John'}

    return render(request, 'template.html', context)
```

- The HTML template uses {{ name }} as a placeholder.
- The greet function in views.py creates a context dictionary ({'name': 'John'}).
- The render function combines the template with the context to generate the final HTML.

Python Templates:

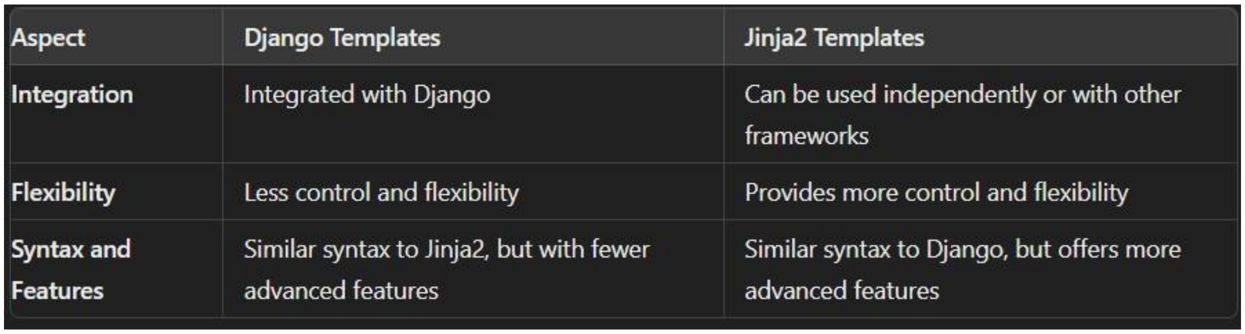
- Python templates like Jinja2 can also be used for rendering HTML.
- Jinja2 is similar to Django's templating engine but offers more control and flexibility.
- **Example:**

```
<!-- template.html (Jinja2) -->
<h1>Hello, {{ name }}!</h1>

# example.py
from jinja2 import Template
template = Template('<h1>Hello, {{ name }}!</h1>')
print(template.render(name='John'))
```

- The Jinja2 template is defined within the Python code with {{ name }} as a placeholder.
- A Template object is created with the HTML string.
- The render method is called with the context (name='John'), generating and printing the final HTML.

Differences:



Aspect	Django Templates	Jinja2 Templates
Integration	Integrated with Django	Can be used independently or with other frameworks
Flexibility	Less control and flexibility	Provides more control and flexibility
Syntax and Features	Similar syntax to Jinja2, but with fewer advanced features	Similar syntax to Django, but offers more advanced features

4b. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event (10 Marks)

1. **Create a Django Project:**

```
django-admin startproject myproject
cd myproject
```

2. **Create a Django App:**

```
python manage.py startapp myapp
```

3. **Configure myapp in settings.py:**

```
INSTALLED_APPS = [
    # ...
    'myapp',
]
```

4. **Define the View in myapp/views.py:**

```
from django.shortcuts import render

def display_lists(request):
    fruits = ["Apple", "Banana", "Cherry"]
    students = ["Alice", "Bob", "Charlie"]
    return render(request, 'lists.html', {'fruits': fruits, 'students': students})
```

5. **Configure URL in myapp/urls.py:**

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

urlpatterns = [
    path('lists/', display_lists, name='display_lists'),
]
```

6. **Include App URLs in myproject/urls.py:**

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

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

7. **Create the Template myapp/templates/lists.html:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Lists</title>
</head>
<body>
    <h1>Fruits</h1>
    <ul>
        {% for fruit in fruits %}
            <li>{{ fruit }}</li>
        {% endfor %}
    </ul>

    <h1>Selected Students</h1>
    <ol>
        {% for student in students %}
            <li>{{ student }}</li>
        {% endfor %}
    </ol>
</body>
</html>
```

8. **Run the Server:**

   ```
   python manage.py runserver
   ```

9. **Access the URL:**

   - Navigate to http://localhost:8000/lists/ to see the unordered list of fruits and ordered list of selected students.

Module 3

Q.05

a. Discuss Migration of Database with an example. (10 Marks)

b. Create a simple Django project called urls_dispatcher_example with two applications (articles and blog). (10 Marks)

5a. Discuss Migration of Database with an example (10 Marks)

Migration:

- Migration is the process of moving the database schema and data from one version to another.
- In Django, migrations are used to propagate changes you make to your models (adding a field, deleting a model, etc.) into your database schema.
- This ensures that the database schema remains in sync with your model definitions.

Example:

1. **Create a New Model:**

```
# models.py
from django.db import models
class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()
```

2. **Generate Migrations**

```
python manage.py makemigrations
```

   - This command generates a migration file in the migrations directory.

3. **Apply Migrations:**

   python manage.py migrate

   - ❖ This command applies the migrations to the database, creating the necessary tables and columns.

4. **Example Output:**

```
Migrations for 'myapp':
  myapp/migrations/0001_initial.py
    - Create model Author
```

5. **Check Database**

   - ❖ The Author table should now exist in the database with the specified columns.

5b. Create a simple Django project called urls_dispatcher_example with two applications (articles and blog) (10 Marks)

Step-by-Step Guide:

1. **Create the Django Project:**

```
django-admin startproject urls_dispatcher_example
cd urls_dispatcher_example
```

2. **Create the articles App:**

```
django-admin startapp articles
```

3. **Create the blog App:**

```
django-admin startapp blog
```

4. **Project Structure:**

```
urls_dispatcher_example/
├── articles/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── blog/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── urls_dispatcher_example/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

5. **Define Views for Each App:**

   articles/views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Welcome to the Articles section")
```

   blog/views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Welcome to the Blog section")
```

6. **Configure URLs in Each App:**

   articles/urls.py:

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

urlpatterns = [
    path('', views.index, name='index'),
]
```

   blog/urls.py:

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

urlpatterns = [
    path('', views.index, name='index'),
]
```

7. **Include App URLs in the Project's urls.py:**

   urls_dispatcher_example/urls.py:

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

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

8. **Run the Server:**

```
python manage.py runserver
```

9. **Access the URLs:**
   - ❖ **Navigate to http://localhost:8000/articles/ to see the Articles section.**
   - ❖ **Navigate to http://localhost:8000/blog/ to see the Blog section.**

Q.06

a. Explain steps of Configuring URLs in Django. (10 Marks)

b. Discuss Django Form Submission. (10 Marks)

6a. Explain steps of Configuring URLs in Django (10 Marks)

Step 1: Create urls.py in Your App

- Define URL patterns specific to the app.
- Example:

```
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Step 2: Define URL Patterns in urls.py

- Map URLs to view functions.
- Example:

```
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
]
```

Step 3: Include the App's urls.py in the Project's urls.py

- Integrate the app’s URLs into the project’s URL configuration.
- Example:

```
# project/urls.py
from django.contrib import admin
from django.urls import path, include

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

Step 4: Test the URL Configuration

- Verify that the URLs route correctly to the specified views.
- **Steps:**
  - Run the server: python manage.py runserver
  - Visit the URLs (e.g., http://localhost:8000/myapp/ and http://localhost:8000/myapp/about/) to ensure they are functioning as expected.

6b. Discuss Django Form Submission (10 Marks)

Form Submission:

1. **Creating a Form Class:**
   - ❖ Django provides a form class to handle user input and validation.
   - ❖ Define a form class in forms.py.

```
# forms.py
from django import forms
class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)
```

2. **Rendering the Form in a Template:**
   - ❖ **Use the form class in a view to render the form in a template.**

```
# views.py
from django.shortcuts import render
from .forms import ContactForm

def contact(request):
    form = ContactForm()
    return render(request, 'contact.html', {'form': form})

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

3. **Handling Form Submission:**

   - Process the form data when the form is submitted.

```
# views.py
from django.shortcuts import render
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Process form data
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            message = form.cleaned_data['message']
            # Perform desired operations (e.g., save data, send email)
    else:
        form = ContactForm()
    return render(request, 'contact.html', {'form': form})
```

4. **Using GET or POST Methods:**

   - Forms can be submitted using GET or POST methods.
   - GET: Use for retrieving data.
   - POST: Use for submitting data that modifies the server state.

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

Module 4

Q.07

a. Define Generic Views and its types. (10 Marks)

b. What is MIME and discuss its types. (10 Marks)

Q.08

a. Discuss how to create templates for each view with an example. (10 Marks)

b. Explain Dynamic CSV using database. (10 Marks)

7a. Define Generic Views and its types (10 Marks)

Generic Views:

- Generic Views in Django are pre-built views that simplify common tasks like displaying a list of objects or handling forms.
- They provide a way to handle these tasks with less code by leveraging Django's built-in functionality.
- These views are designed to handle common patterns of web application functionality and can be easily customized for specific needs.

Types of Generic Views:

1. **ListView:**
   - Displays a list of objects from a model.
   - Example

```
from django.views.generic import ListView
from .models import Book
class BookListView(ListView):
    model = Book
    template_name = 'book_list.html'
```

2. **DetailView:**
   - ❖ Displays detailed information about a single object.
   - ❖ Example

```
from django.views.generic import DetailView
from .models import Book

class BookDetailView(DetailView):
    model = Book
    template_name = 'book_detail.html'
```

3. **CreateView:**
   - ❖ Provides a form to create a new object.
   - ❖ Example

```
from django.views.generic.edit import CreateView
from .models import Book

class BookCreateView(CreateView):
    model = Book
    fields = ['title', 'author', 'published_date']
    template_name = 'book_form.html'
```

4. **UpdateView:**
   - ❖ Provides a form to update an existing object.
   - ❖ Example

```
from django.views.generic.edit import UpdateView
from .models import Book

class BookUpdateView(UpdateView):
    model = Book
    fields = ['title', 'author', 'published_date']
    template_name = 'book_form.html'
```

5. **DeleteView:**
    - ❖ Provides a form to confirm the deletion of an object.
    - ❖ Example

```
from django.views.generic.edit import DeleteView
from .models import Book

class BookDeleteView(DeleteView):
    model = Book
    success_url = '/books/'
    template_name = 'book_confirm_delete.html'
```

7b. What is MIME and discuss its types (10 Marks)

MIME (Multipurpose Internet Mail Extensions):

- ❖ MIME is a standard for specifying the type of data being sent over the internet.
- ❖ It extends the format of email messages to support text in character sets other than ASCII, attachments, and multimedia content.
- ❖ MIME types (also known as media types) are used to specify the nature and format of a document, file, or set of bytes.

Common MIME Types:

1. **text/html:**
    - ❖ Represents HTML documents.
    - ❖ Example: A web page.
    - ❖ Usage: Content-Type: text/html

2. **text/plain:**
    - ❖ Represents plain text without formatting.
    - ❖ Example: A text file.
    - ❖ Usage: Content-Type: text/plain

3. **image/jpeg:**
    - ❖ Represents JPEG image files.
    - ❖ Example: A photograph.
    - ❖ Usage: Content-Type: image/jpeg

4. **image/png:**
    - ❖ Represents PNG image files.
    - ❖ Example: A logo or graphic.
    - ❖ Usage: Content-Type: image/png

5. **application/json:**
    - ❖ Represents JSON data.
    - ❖ Example: JSON data for API responses.
    - ❖ Usage: Content-Type: application/json

6. **application/xml:**
    - ❖ Represents XML data.
    - ❖ Example: XML data used in APIs or configurations.
    - ❖ Usage: Content-Type: application/xml

7. **application/pdf:**
    - ❖ Represents PDF documents.
    - ❖ Example: A document or report.
    - ❖ Usage: Content-Type: application/pdf

8. **multipart/form-data:**
    - ❖ Represents form data that includes files.
    - ❖ Example: File uploads in web forms.
    - ❖ Usage: Content-Type: multipart/form-data

8a. Discuss how to create templates for each view with an example (10 Marks)

Creating Templates for Each View:

1. **Define a View:**
   - ❖ Create a view function or class-based view to handle the request and render a template.

```
# views.py
from django.shortcuts import render

def home_view(request):
    return render(request, 'home.html')
```

2. **Create the Template File:**
   - ❖ Place the HTML template file in the appropriate template directory.
   - ❖ Template Directory Structure

```
myproject/
├── myapp/
│   ├── templates/
│   │   ├── home.html
│   │   └── other_template.html
```

home.html:

```
<!DOCTYPE html>

<html>

<head>

    <title>Home Page</title>

</head>

<body>
```

```
    <h1>Welcome to the Home Page</h1>

    <p>This is the home view of our application.</p>

</body>

</html>
```

3. **Map the View to a URL:**
   - ❖ Define URL patterns to map the view to a specific URL.

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

urlpatterns = [
    path('', home_view, name='home'),
]
```

4. **Render the Template:**
   - ❖ When the view is accessed via the URL, Django will render the home.html template, sending the resulting HTML to the browser.

8b. Explain Dynamic CSV using database (10 Marks)

Dynamic CSV:

- Dynamic CSV involves generating CSV files from database queries using Django.
- This is useful for exporting data in a format that can be easily used in spreadsheets or other applications.

Example:

1. **Define a Model:**

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

class MyModel(models.Model):
    field1 = models.CharField(max_length=100)
    field2 = models.IntegerField()
```

2. **Create a View to Export CSV:**
   - This view generates a CSV file containing data from the MyModel database table.

```
# views.py
import csv
from django.http import HttpResponse
from .models import MyModel

def export_csv(request):
    # Create the HTTP response with CSV content type
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="data.csv"'

    # Create a CSV writer
    writer = csv.writer(response)
```

```
# Write the header row
writer.writerow(['Field1', 'Field2'])

# Write data rows
for obj in MyModel.objects.all():
    writer.writerow([obj.field1, obj.field2])

return response
```

3. **Map the CSV Export View to a URL:**

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

urlpatterns = [
    path('export-csv/', export_csv, name='export_csv'),
]
```

4. **Test the CSV Export:**
   - ❖ Access the URL http://localhost:8000/export-csv/ to download the CSV file containing the data from MyModel.

Module 5

Q.09

a. Explain XHTML Http Request and Response. (10 Marks)

b. Develop a registration page for student enrollment without page refresh using AJAX. (10 Marks)

Q.10

a. Discuss how the setting of Javascript in Django. (10 Marks)

b. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched. (10 Marks)

9a. Explain XHTML Http Request and Response (10 Marks)

- ❖ XHTML (Extensible Hypertext Markup Language) is a stricter version of HTML, designed to be more compliant with XML standards.
- ❖ It enforces rules that HTML does not, such as requiring all tags to be properly closed and nested.

HTTP Request and Response in XHTML:

1. **HTTP Request:**
   - ❖ A request sent from the client (usually a web browser) to the server to retrieve or send data.

   Components:

   1. **Method:** Determines the action (GET, POST, etc.).
   2. **URL:** Specifies the resource.
   3. **Headers**: Includes metadata (e.g., Accept, Content-Type).
   4. **Body:** Data sent with the request (e.g., form data for POST requests).

Example:

```
GET /page.html HTTP/1.1
Host: www.example.com
Accept: application/xhtml+xml
```

2. **HTTP Response:**
    - ❖ The server's reply to the client's request, containing the requested resource or an error message.

Components:

1. **Status Code:** Indicates the result (e.g., 200 OK, 404 Not Found).
2. **Headers:** Metadata about the response (e.g., Content-Type).
3. **Body:** The content (e.g., XHTML document).

Example:

```
HTTP/1.1 200 OK
Content-Type: application/xhtml+xml

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Example Page</title></head>
<body><h1>Hello, world!</h1></body>
</html>
```

- ❖ XHTML must adhere to XML syntax rules.
- ❖ HTTP requests and responses are the same for XHTML as for HTML but ensure XHTML content is well-formed XML.

9b. Develop a registration page for student enrollment without page refresh using AJAX (10 Marks)

HTML & JavaScript (registration.html):

1. **Create the Form:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration</title>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            document.getElementById('registration-form').addEventListener('submit',
function(e) {
                e.preventDefault(); // Prevent default form submission

                fetch('/register/', {
                    method: 'POST',
                    headers: {
                        'X-CSRFToken': '{{ csrf_token }}', // CSRF protection
                        'Content-Type': 'application/x-www-form-urlencoded'
                    },
                    body: new URLSearchParams(new FormData(this)).toString()
                })
                .then(response => response.text())
                .then(data => {
                    document.getElementById('response').innerHTML = data; // Display
response
                })
                .catch(error => console.error('Error:', error));
            });
        });
    </script>
</head>
<body>
    <form id="registration-form">
        <input type="text" name="name" placeholder="Name" required>
        <input type="email" name="email" placeholder="Email" required>
        <button type="submit">Register</button>
    </form>
    <div id="response"></div>
</body>
</html>
```

2. **Django View (views.py):**

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
@csrf_exempt
def register(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        # Process registration (e.g., save to database)
        return HttpResponse(f"Registered {name} with email {email}")
    return HttpResponse("Invalid request", status=400)
```

3. **Django URL Configuration (urls.py):**

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

- The form submission uses JavaScript's fetch API to send data to the server without refreshing the page.
- CSRF protection is included by sending the CSRF token.

Q.10

a. Discuss how the setting of Javascript in Django. (10 Marks)

b. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched. (10 Marks)

10a. Discuss how to set JavaScript in Django (10 Marks)

Setting Up JavaScript in Django:

1. **Create Static Files:**
   - Place JavaScript files in the static directory of your Django app.
     Directory Structure:
     ```
     myproject/
     ├── myapp/
     │   ├── static/
     │   │   ├── js/
     │   │   │   └── script.js
     ├── manage.py
     └── myproject/
         └── settings.py
     ```
2. **Configure Static Files:**
   - Ensure STATIC_URL is set in settings.py. Optionally, configure STATICFILES_DIRS if your static files are outside the app directories.

settings.py:

```
STATIC_URL = '/static/'
```

3. **Include JavaScript in Templates:**
    - Use the {% static %} template tag to link JavaScript files.

Example HTML (base.html):

```
{% load static %}

<!DOCTYPE html>

<html>

<head>

   <title>My Django App</title>

   <script src="{% static 'js/script.js' %}" defer></script>

</head>

<body>

   <h1>Welcome to My Django App</h1>

</body>

</html>
```

4. **Run the Server:**
    - Start the Django development server to serve static files.

      ```
      python manage.py runserver
      ```

- ❖ Django's static files management allows including JavaScript files via the {% static %} tag.
- ❖ Use defer attribute for better performance, as it loads the script after the HTML content.

10b. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched (10 Marks)

HTML & JavaScript (search.html):

1. **Create the Search Input:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Search</title>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            document.getElementById('search').addEventListener('input', function() {
                fetch(`/search/?q=${this.value}`)
                    .then(response => response.json())
                    .then(data => {
                        const resultsDiv = document.getElementById('results');
                        resultsDiv.innerHTML = data.courses.map(course =>
`<p>${course}</p>`).join('');
                    })
                    .catch(error => console.error('Error:', error));
            });
        });
    </script>
</head>
<body>
    <input type="text" id="search" placeholder="Search for student">
    <div id="results"></div>
</body>
</html>
```

2. **Django View (views.py):**

```
from django.http import JsonResponse
from .models import Student

def search(request):
    query = request.GET.get('q', '')
    courses = []
    if query:
        student = Student.objects.filter(name__icontains=query).first()
        if student:
            courses = list(student.courses.values_list('name', flat=True))
    return JsonResponse({'courses': courses})
```

3. **Django URL Configuration (urls.py):**

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

urlpatterns = [
    path('search/', search, name='search'),
]
```

- The search feature uses AJAX to fetch and display data without a page refresh.
- The server responds with JSON data, which is processed by JavaScript to update the page content dynamically.