Model Question Paper-1/2 with effect from 2022-23 (CBCS Scheme)

USN

Fourth Semester B.E. Degree Examination Subject Title
Full Stack Development

TIME: 03 Hours **Max. Marks: 100**

Note: 01. Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**.

These Answers From TEXTBOOK

		Module 1 **Download**	***Bloom's Taxonomy Level**	**Marks**
Q.01	a	Define a) Web Frameworks with example. b) URLS Web Frameworks and URLs Web Frameworks with Example - Definition : A web framework is a software framework designed to aid in the development of web applications by providing a standard way to build and deploy them. It simplifies common tasks and promotes code reuse. - Example : Django is a prominent web development framework written in Python that aims to streamline web application development and maintenance. It allows developers to focus on the core aspects of their applications while providing abstractions for common web development patterns and shortcuts for programming tasks. URLs - Definition : URLs (Uniform Resource Locators) are web addresses used to locate resources on the internet. In the context of web development, URLs play a crucial role in mapping specific web pages or functions to their corresponding addresses. - URLconf in Django : In Django, URLconfs (URL configurations) define the mapping between URLs and the code that handles them. They help route incoming web requests to the appropriate views or functions within the web application. - URL Mapping Example : In Django, a URLconf typically consists of patterns that match specific URLs to corresponding views or functions. By defining these patterns, developers can easily direct incoming requests to the appropriate parts of their web applications.	L1	4M
	b	Explain features of Django Features of Django Introduction to Django Django is a web development framework that focuses on saving time and simplifying the web development process. It offers high-level abstractions of common web development patterns, shortcuts for programming tasks, and clear conventions for problem-solving. Django aims to allow developers to concentrate on the core aspects of their web applications while minimizing repetitive tasks.	L2	8M

		Django's Origins and Culture - Django was extracted from real-world code rather than being an academic exercise or a commercial product, focusing on solving web development problems faced by its developers. - The framework is actively improved almost daily to save developers time, ensure easy maintenance of applications, and performance under load. - Django's maintainers are motivated by their desire to enhance their own development experience. Structure and Curriculum - Chapters 1 through 12 form the foundation of learning how to use Django, enabling users to build and deploy Django-powered websites. - Chapters 1-7 cover the core curriculum, Chapters 8-11 delve into more advanced Django usage, and Chapter 12 focuses on deployment. - Chapters 13-20 concentrate on specific Django features, which can be read in any order, while the appendices serve as a reference. Required Knowledge - Readers are expected to have a grasp of procedural and object-oriented programming basics, including control structures, data structures, variables, classes, and objects. - Web development experience is helpful but not mandatory, as the book promotes best practices for readers without prior experience in the field. - Python knowledge is essential, as Django is essentially a collection of libraries written in Python, requiring users to write Python code to develop sites using Django. Django's History and Sweet Spot - Django originated from real-world applications built by a web development team in Lawrence, Kansas, USA. - The framework's creation process mirrored the typical evolution of a web developer inventing a framework after multiple application refactoring instances. - Django's history explains its focus on offering features tailored for content-driven sites while remaining versatile for various types of dynamic web applications. Django as a Web Framework - Django is part of a new generation of web frameworks that provide high-level abstractions of web development patterns and clear conventions for problem-solving. - The framework aims to simplify web development tasks, allowing developers to focus on the core functionality of their applications. Conclusion Django is a robust web development framework that streamlines the development process by offering shortcuts for common programming tasks, clear conventions, and high-level abstractions of web development patterns.		
	c	Explain MVC Architecture	L2	8M

Explaining MVC Architecture

Overview of MVC Architecture
- MVC stands for Model-View-Controller.
- MVC is a software development pattern that separates an application into three main components: Model, View, and Controller.
- The Model represents the data and business logic of the application.
- The View displays the data to the user and handles user interactions.
- The Controller acts as an intermediary that handles user input, processes requests, and updates the Model and View accordingly.
- MVC architecture promotes a separation of concerns, making the code easier to maintain and scale.

Components of MVC Architecture in Django
1. Model :
   - Represents the data structure and business logic.
   - In Django, models are Python classes that define the structure of the database tables.
   - Models interact with the database to store and retrieve data.

2. View :
   - Contains the application's business logic.
   - In Django, views are Python functions or classes that receive web requests and return web responses.
   - Views interact with models to retrieve data and pass it to templates for rendering.

3. Controller (URLconf):
   - Handles the incoming HTTP requests and directs them to the appropriate view.
   - In Django, URLconf maps URLs to view functions or classes.
   - URL patterns in Django define which view function should be called for a specific URL.

Implementation of MVC Architecture in Django
- Model :
  - Defined using Python classes in the `models.py` file.
  - Models interact with the database using Django's Object-Relational Mapping (ORM) system.
  - Example: `class Book(models.Model): name = models.CharField(max_length=50) pub_date = models.DateField()`

- View :
  - Contains the business logic for processing requests and generating responses.
  - Views interact with models to retrieve data and render templates.
  - Example:

```
 python
from django.shortcuts import render_to_response
from models import Book

def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html', {'book_list':
book_list})
```

		- Controller (URLconf): - Maps URLs to view functions in the `urls.py` file. - Specifies which view function should handle specific URL patterns. - Example: python from django.conf.urls.defaults import * import views urlpatterns = patterns('', (r'^latest/$', views.latest_books),) Model-View-Controller (MVC) Pattern in Django - In Django, the Model corresponds to the Model in MVC, the View corresponds to the View, and the Controller is represented by the URLconf. - Django follows the MVC pattern by separating the data (Model), logic (View), and presentation (Controller) aspects of web applications. Key Advantages of MVC Architecture - Separation of Concerns : Each component has a specific role, making the codebase modular and easier to maintain. - Reusability : Components can be reused in different parts of the application. - Testability : Components can be tested independently for easier debugging and quality assurance.		
		OR		
Q.02	a	Define a) Django URL Configuration. b) Loose Coupling. Django URL Configuration and Loose Coupling Django URL Configuration - Definition : Django URL Configuration, often referred to as URLconf, defines the mapping between URLs and the code that handles those URLs in a Django application. - Implementation : To add a URL and view to the URLconf, you create a Python tuple mapping a URL pattern to the view function - Example : from django.conf.urls.defaults import * from mysite.views import hello urlpatterns = patterns('', ('^hello/$', hello),) - Default Configuration : By default, the URLconf is empty in Django, and it acts as a blank slate until URLs and views are added. - Purpose : URLconf is crucial for specifying which view functions should be called for different URLs in a Django-powered website.	L1	4M

		Loose Coupling - Definition : Loose coupling is a software development approach that emphasizes making components interchangeable without affecting each other significantly. - Application in Django URLconfs : In Django, URL definitions and the view functions they call are loosely coupled. This means that changes to URLs or view functions can be made independently without impacting the other component. - Example : Changing the URL for an application or modifying the logic of a view function can be done without affecting the other part due to loose coupling. - Flexibility : Loose coupling in Django's URLconfs allows for easy modifications to URLs and views, enabling developers to switch out components without extensive repercussions. - Benefit : Facilitates maintenance and updates by providing a more modular and flexible structure for managing URLs and view functions.		
	b	List out Django Exceptions & Errors with neat diagram. Django Exceptions & Errors Overview Django Error Page - Django's error page displays key information about exceptions, Python tracebacks, and local variables. - It provides detailed traceback information, including file names, function names, line numbers, and source code snippets. - Users can interactively explore the stack frames, view local variables, and analyze the state of the program. - The error page allows users to trigger errors for debugging purposes without print statements. - It offers a "Switch to copy-and-paste view" option for easily sharing tracebacks and a "Share this traceback" button for posting on a public website. Example of Django Error Handling - Demonstrated an example where a small bug caused a TypeError due to a commented-out code snippet. - Utilized the error page to understand and debug the issue, emphasizing the importance of exploring the error page features. - Highlighted the sensitivity of error information to prevent exposing Python code and configuration details on the public Internet. Next Steps in Django Development - Transitioning from hard-coded HTML in Python view functions to utilizing Django's template engine for improved separation of design and code. - Introduction to Django's template engine in the upcoming chapter to enhance web development practices.	L2	8M
	c	Develop a Django app that displays current date & time. Sure! Let's develop a simple Django application that displays the current date and time. Here are the steps:	L2	8M

Step 1: Set up your Django project

1. **Install Django:**

   If you don't have Django installed, you can install it using pip:

   ```
   bash
   Copy code
   pip install django
   ```

2. **Create a new Django project:**

   ```
   bash
   Copy code
   django-admin startproject mydatetimeapp
   ```

3. **Navigate to your project directory:**

   ```
   bash
   Copy code
   cd mydatetimeapp
   ```

4. **Create a new Django app:**

   ```
   bash
   Copy code
   python manage.py startapp datetime_display
   ```

Step 2: Set up the Django app

1. **Add the app to your project settings:**

   Open `mydatetimeapp/settings.py` and add `'datetime_display'` to the `INSTALLED_APPS` list:

   ```
   python
   Copy code
   INSTALLED_APPS = [
       ...
       'datetime_display',
   ]
   ```

2. **Create a view to display the current date and time:**

   Open `datetime_display/views.py` and add the following code:

   ```
   python
   Copy code
   from django.shortcuts import render
   from django.utils import timezone

   def current_datetime(request):
       now = timezone.now()
   ```

```
    return render(request,
'datetime_display/current_datetime.html',
{'current_datetime': now})
```

3. **Create a URL pattern for the view:**

   Open `datetime_display/urls.py` and add the following code:

```
python
Copy code
from django.urls import path
from . import views

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

4. **Include the app URLs in the project URL configuration:**

   Open `mydatetimeapp/urls.py` and include the `datetime_display` URLs:

```
python
Copy code
from django.contrib import admin
from django.urls import include, path

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

5. **Create a template to display the current date and time:**

   Create a directory `datetime_display/templates/datetime_display/` and create a file `current_datetime.html` inside it with the following content:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Current Date and Time</title>
</head>
<body>
    <h1>Current Date and Time</h1>
    <p>{{ current_datetime }}</p>
</body>
</html>
```

Step 3: Run the Django app

1. **Run the development server:**

   ```
   bash
   Copy code
   python manage.py runserver
   ```

2. **Access the app in your web browser:**

   Open your web browser and go to `http://127.0.0.1:8000/`. You should see the current date and time displayed on the page.

		Module-2 Download		
Q. 03	a	Explain MVT Architecture. Explanation of MVT Architecture Model-View-Template (MVT) Architecture Overview - Model-View-Template (MVT) Architecture : - In Django, the Model-View-Template (MVT) architecture is used instead of the traditional Model-View-Controller (MVC) pattern. - Model (M) : Represents the data access layer, handling data-related aspects like access, validation, behaviors, and relationships . - Template (T) : Represents the presentation layer, focusing on how data should be displayed on a webpage or other documents. - View (V) : Represents the business logic layer, acting as a bridge between models and templates , accessing the model and deferring to appropriate templates. Django's MVT Implementation - Model Layer (M) : - Managed by Django's database layer, providing tools for performing database queries using Python. - Template Layer (T) : - Handled by views and templates in Django, responsible for selecting data to display and determining how to present it. - View Layer (V) : - Business logic layer in Django, delegates to views based on user input and accesses models. - In Django, views are often considered as controllers, and templates as views, leading to a distinction from traditional MVC frameworks. - MVT vs. MVC : - Django's MVT architecture closely resembles MVC but with a focus on models, templates, and views, leading to the term "MTV framework." - The framework itself handles the controller aspect, with the majority of the work happening in models, templates, and views. Clarification on MVT Components - MVC vs. MTV : - While MVC is a more traditional pattern where the controller manages user input, MTV in Django places more emphasis on models, templates, and views. - Django's structure aligns closely with the MTV pattern, deviating slightly from the MVC paradigm. Additional Insights - Communication Streamlining : - Defining patterns like MVC or MVT aims to enhance communication among developers by providing a shared vocabulary and framework for discussing software architecture. Comparison to Other Frameworks - Comparison to Other MVC Frameworks :	L2	10M

		- In Django, views are sometimes equated to controllers, and templates to views, leading to potential confusion with other MVC frameworks like Ruby on Rails. Key Features of MVT Architecture in Django 1. Model Layer : Handles data access logic. 2. Template Layer : Manages data presentation and display. 3. View Layer : Contains the business logic and serves as a mediator between models and templates.		
	b	Explain Template Inheritence with an example.	L2	10M

Explanation of Template Inheritance with an Example

Template Inheritance Overview

Template inheritance in Django allows you to create a base template with common elements and define "blocks" that child templates can override. This approach helps reduce duplication and manage common page areas efficiently.

Example Implementation

To illustrate template inheritance, consider creating a base template and a child template for a timestamp site:

# Base Template
```
 html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
   <title>{% block title %}{% endblock %}</title>
</head>
<body>
   <h1>My helpful timestamp site</h1>
   {% block content %}{% endblock %}
   {% block footer %}
   <hr>
   <p>Thanks for visiting my site.</p>
   {% endblock %}
</body>
</html>
```

# Child Template (Current_datetime.html)
```
 html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
   <title>The current time</title>
</head>
```

In this example, the child template `Current_datetime.html` extends the base template by filling in blocks for the title and content, inheriting the structure defined in the base template.

Advantages of Template Inheritance

		1. Reduces redundancy and duplication in template code. 2. Allows for a modular and organized approach to managing templates. 3. Enables easy updating of common elements across multiple pages. 4. Enhances code readability and maintainability in larger projects. Implementation Considerations When using template inheritance, ensure proper structure and naming conventions for base and child templates to maintain clarity and consistency throughout the project.		
OR				
Q.04	a	Explain Django and Python Templates with an example. Django and Python Templates Overview Explanation of Django Templates Django templates are used to separate the presentation of a document from its data. They define placeholders and basic logic to regulate how a document should be displayed. Templates can handle complex data structures such as lists, dictionaries, and custom objects. The system aims to accommodate both designers and Python programmers, offering ways to extend syntax by writing raw Python code if needed. Example of Using Django Templates A simple example demonstrates the usage of Django templates to create an HTML page thanking a person for placing an order. It involves placeholders for person's name, company name, shipping date, ordered items, warranty information, and a closing message. The template employs basic logic like loops and conditional statements to personalize the message based on the provided data. Details of Template Usage in Views Templates are integrated into views in Django to generate dynamic content. An example shows a view function called `current_datetime` that retrieves the current date and time, embeds it into an HTML template, and returns the rendered HTML response. Key Features of Django Template System - Templates are strings of text separating document presentation and data. - Templates handle complex data structures elegantly. - Django expects template authors to be comfortable editing HTML directly. - Templates can include basic logic like loops and conditional statements. - Template system allows for extending syntax with raw Python code. - Integration of templates with views enables dynamic content generation.	L2	10M

	b	Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event.	L2	10M

Let's develop a Django application that displays an unordered list of fruits and an ordered list of selected students for an event. Here are the steps:

Step 1: Set up your Django project

1. **Install Django:**

   If you don't have Django installed, you can install it using pip:

   ```
   bash
   Copy code
   pip install django
   ```

2. **Create a new Django project:**

   ```
   bash
   Copy code
   django-admin startproject mylistapp
   ```

3. **Navigate to your project directory:**

   ```
   bash
   Copy code
   cd mylistapp
   ```

4. **Create a new Django app:**

   ```
   bash
   Copy code
   python manage.py startapp list_display
   ```

Step 2: Set up the Django app

1. **Add the app to your project settings:**

   Open `mylistapp/settings.py` and add `'list_display'` to the `INSTALLED_APPS` list:

   ```
   python
   Copy code
   INSTALLED_APPS = [
       ...
       'list_display',
   ]
   ```

2. **Create a view to display the lists:**

   Open `list_display/views.py` and add the following code:

   ```
   python
   Copy code
   from django.shortcuts import render
   ```

```
def display_lists(request):
    fruits = ['Apple', 'Banana', 'Cherry',
'Date', 'Elderberry']
    students = ['Alice', 'Bob', 'Charlie',
'Diana', 'Edward']
    return render(request,
'list_display/display_lists.html', {'fruits':
fruits, 'students': students})
```

3. **Create a URL pattern for the view:**

   Open `list_display/urls.py` and add the following code:

```
python
Copy code
from django.urls import path
from . import views

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

4. **Include the app URLs in the project URL configuration:**

   Open `mylistapp/urls.py` and include the `list_display` URLs:

```
python
Copy code
from django.contrib import admin
from django.urls import include, path

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

5. **Create a template to display the lists:**

   Create a directory `list_display/templates/list_display/` and create a file `display_lists.html` inside it with the following content:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Lists of Fruits and
Students</title>
</head>
<body>
    <h1>Unordered List of Fruits</h1>
    <ul>
```

```
            {% for fruit in fruits %}
                <li>{{ fruit }}</li>
            {% endfor %}
        </ul>

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

Step 3: Run the Django app

1. **Run the development server:**

   ```
   bash
   Copy code
   python manage.py runserver
   ```

2. **Access the app in your web browser:**

   Open your web browser and go to `http://127.0.0.1:8000/`. You should see the unordered list of fruits and the ordered list of selected students displayed on the page.

Module-3 Download				
Q. 05	a	Discuss Migration of Database with an example. Discuss Migration of Database with an Example Migration of Database Database migration involves moving data from one database to another, usually during software upgrades or changes in database systems. It is essential to ensure data integrity and consistency throughout the migration process. Example of Database Migration in Django In Django, database migration can be achieved using tools like Django's built-in migration framework. This framework helps manage changes to the database schema over time. Steps for Database Migration 1. Plan the Migration : Evaluate the current database structure and the desired changes. 2. Create Migration Files : Use Django's migration commands to create migration files that capture the changes to be applied to the database schema. 3. Apply Migrations : Execute the migration files to implement the changes in the database. 4. Test Migration : Verify that the migration was successful and that data remains consistent. Considerations for Database Migration - Backup Data : Always back up the database before performing any migration to avoid data loss. - Version Control : Keep track of database schema changes using version control systems to manage migrations effectively. - Review Changes : Thoroughly review migration scripts before applying them to ensure they align with the intended modifications . Importance of Database Migration Database migration is crucial for maintaining data consistency, adapting to evolving software requirements, and ensuring the smooth operation of applications over time. Conclusion Database migration is a critical process in software development, especially during system upgrades or changes. By following best practices and utilizing tools like Django's migration framework, developers can efficiently manage database schema changes.	L2	10M
	b	Create a simple Django project called urls_dispatcher_example with two application(articles and Blog). Let's create a Django project called `urls_dispatcher_example` with two applications: `articles` and `blog`. Here are the steps: **Step 1: Set up the Django project**	L2	10M

1. **Install Django:**

   If you don't have Django installed, you can install it using pip:

   ```
   bash
   Copy code
   pip install django
   ```

2. **Create a new Django project:**

   ```
   bash
   Copy code
   django-admin startproject urls_dispatcher_example
   ```

3. **Navigate to your project directory:**

   ```
   bash
   Copy code
   cd urls_dispatcher_example
   ```

4. **Create the `articles` and `blog` applications:**

   ```
   bash
   Copy code
   python manage.py startapp articles
   python manage.py startapp blog
   ```

Step 2: Set up the applications

1. **Add the applications to your project settings:**

   Open `urls_dispatcher_example/settings.py` and add `'articles'` and `'blog'` to the `INSTALLED_APPS` list:

   ```
   python
   Copy code
   INSTALLED_APPS = [
       ...
       'articles',
       'blog',
   ]
   ```

2. **Create views for each application:**
   - **Articles app:** Open `articles/views.py` and add the following code:

     ```
     python
     Copy code
     from django.http import HttpResponse

     def index(request):
         return HttpResponse("Welcome to the Articles
     Home Page!")

     def detail(request, article_id):
         return HttpResponse(f"You're looking at
     article {article_id}.")
     ```

   - **Blog app:** Open `blog/views.py` and add the following code:

     ```
     python
     Copy code
     ```

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Welcome to the Blog Home
Page!")

def detail(request, post_id):
    return HttpResponse(f"You're reading blog
post {post_id}.")
```

3. **Create URL patterns for each application:**
   - **Articles app:** Create a file `articles/urls.py` and add the following code:

     ```
     python
     Copy code
     from django.urls import path
     from . import views

     urlpatterns = [
         path('', views.index, name='articles_index'),
         path('<int:article_id>/', views.detail,
     name='article_detail'),
     ]
     ```

   - **Blog app:** Create a file `blog/urls.py` and add the following code:

     ```
     python
     Copy code
     from django.urls import path
     from . import views

     urlpatterns = [
         path('', views.index, name='blog_index'),
         path('<int:post_id>/', views.detail,
     name='blog_detail'),
     ]
     ```

4. **Include the application URLs in the project URL configuration:**

   Open `urls_dispatcher_example/urls.py` and include the `articles` and `blog` URLs:

   ```
   python
   Copy code
   from django.contrib import admin
   from django.urls import include, path

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

Step 3: Run the Django project

1. **Run the development server:**

   ```
   bash
   Copy code
   python manage.py runserver
   ```

2. **Access the app in your web browser:**
   - For articles home page: Open your web browser and go to `http://127.0.0.1:8000/articles/`.
   - For specific article detail: Go to `http://127.0.0.1:8000/articles/1/`.
   - For blog home page: Open your web browser and go to `http://127.0.0.1:8000/blog/`.
   - For specific blog post detail: Go to `http://127.0.0.1:8000/blog/1/`.

OR

Q. 06	a	Explain steps of Configuring URLs in Django.	L2	10M

Steps of Configuring URLs in Django

Overview

Configuring URLs in Django involves defining URL patterns in the URLconf, which maps URLs to views in the application. This process is crucial for directing incoming web requests to the appropriate view functions for processing.

Steps to Configure URLs in Django

1. Defining URL Patterns : Create URL patterns in the URLconf file to match incoming URLs.
2. Importing Views : Import the necessary view functions at the top of the URLconf file to associate them with specific URL patterns.
3. Including Other URLconfs : Use the `include()` function to include other URLconfs within the main URLconf for modular and organized URL routing.
4. URL Pattern Matching : Understand how Django matches incoming URLs to the defined patterns in the URLconf.
5. Passing Parameters : Handle URL parameters and pass them to view functions for dynamic content generation.
6. Using String Approach : Choose between passing view functions as function objects or strings in the URLconf based on personal coding style and needs.
7. Special-Casing URLs in Debug Mode : Dynamically alter URLconf behavior based on the Django debug mode setting to control URL routing.
8. Named Groups : Utilize named regular expression groups in URL patterns to capture URL segments and pass them as keyword arguments to views for more advanced routing.

Best Practices and Considerations

- Code Organization : Maintain a structured approach to URLconf management as the application grows in complexity.
- View Function Handling : Ensure consistency in importing view functions and passing them in URL patterns for clarity and maintainability.
- Debugging and Testing : Validate URL configurations and routing behavior during development to prevent issues in production.

Configuring URLs in Django

		Steps to Configure URLs in Django: 1. Utilize the linear way a URLconf is processed and special case specific URLs at the beginning of the patterns. 2. Implement an optional URLconf parameter with a dictionary of keyword arguments to pass to the view function. 3. Factor out the type of object being displayed in views for reusability. 4. Streamline function imports in URLconfs by importing the views module itself. 5. Use the include() function to include URL patterns from a separate file, passing extra options to every line in the included URLconf. Key Points: - Django URLconfs are treated as Python code and can be customized in various ways. - Optional URLconf parameters can be used to pass additional arguments to view functions. - By factorizing views based on object types, you can create reusable and model-agnostic views. - Importing the views module directly can simplify managing the URLconf as the application grows in complexity. - The include() function allows for including URL patterns from separate files, passing extra options to each view in the included URLconf. Additional Notes: - The configuration of URLs in Django involves structuring URL patterns to map specific URLs to view functions. - Properly configuring URLs is essential for directing incoming requests to the appropriate views in a Django web application. SOURCES: - Page 52: Django documentation details advanced functionality in views and URLconfs, including streamlining function imports and using the include() function. - Page 58: Explains how to use optional URLconf parameters in Django for passing keyword arguments to view functions. - Page 61: Demonstrates how to factor out object types in views for reusability and model-agnostic views.		
	b	Discuss Django Form Submission. Overview of Django Form Submission Getting Data from the Request Object - HttpRequest objects in Django are used to access information about the current request, such as the user or web browser loading the page. - These objects contain various attributes and methods to retrieve details about the requested URL. Handling Django Form Submission - When processing form submissions in Django, it is essential to check if the request method is 'POST' to isolate form processing from form display. - Accessing form data using `request.POST` is necessary when the form utilizes `method="post"`. - Validating form fields is crucial, including required fields like subject and message, using `request.POST.get()` with a default value to	L2	10M

		handle missing data. - While optional, fields like email can also be validated using custom algorithms, like checking for the presence of an '@' character. - Form submission success should trigger a redirect, following web development best practices. Django Forms Framework - Django provides a forms library (`django.forms`) to handle form-related tasks, including validation, data cleanup, and form redisplay with errors. - Defining a Form class for each HTML form allows customization of field rendering using widgets to control the presentation logic. Challenges and Best Practices - Manual handling of form validation can become complex and error-prone, especially with multiple form fields. - Redisplaying forms with submitted data filled in helps users identify errors effectively. - Utilizing higher-level libraries can streamline form handling and validation processes.		

Module-4 Download		

Q. 07	a	Define Generic Views and its types Generic Views and Their Types Overview of Generic Views - Purpose: Django's generic views simplify the process of presenting views on database content by providing built-in functionalities. - Benefits: Generic views make generating list and detail views of objects easy and efficient for developers. - Implementation: Configuration dictionaries are created in URLconf files to utilize generic views for different tasks. Types of Generic Views 1. common “simple” tasks: - Redirect to a different page or render a specific template. 2. List and Detail Views for Single Objects: - Display list pages for multiple objects and detail pages for individual objects. 3. Date-Based Objects Views: - Present date-based objects in year/month/day archive pages, associated detail views, and "latest" pages. 4. Customization and Extension of Generic Views: - Generic views can be extended to handle a wider array of situations beyond the standard functionalities. 5. Friendly Template Contexts: - Customize template contexts for better readability and understanding, such as renaming variables for clarity. Extending Generic Views - Purpose: While generic views are efficient for common tasks, extensions may be needed for more specific or complex requirements. - Process: Developers can extend generic views to handle a broader range of use cases using predefined patterns. - Benefits: Extending generic views offers flexibility and customization options tailored to specific project needs. - Common Scenarios : Situations that require extending generic views typically fall into specific patterns dealt with in development.	L2	10M
	b	What is MIME and discuss its types MIME Types and Their Discussion What is MIME? MIME stands for Multipurpose Internet Mail Extensions and is a	L2	10M

		standard that extends the format of email messages to support text in character sets other than ASCII, as well as attachments of audio, video, images, and application programs. It is a way to identify files on the internet according to their nature and format.It allows for the transmission of multimedia content through email and the internet. Types of MIME 1. Text MIME Types: Used for plain text content in emails or web pages. 2. Image MIME Types: Include formats like JPEG, PNG, and GIF for embedding images in emails or web pages. 3. Audio MIME Types: Support audio files such as MP3 or WAV for sending audio content. 4. Video MIME Types: Enable the transmission of video files such as MP4 or AVI. 5. Application MIME Types: Used for sending applications or program files, ensuring proper handling by the recipient's system 6. Message MIME Types: Facilitate the encapsulation of message content, allowing for structured data exchange. 7. Multipart MIME Types: Allow combining different types of content within a single message, like mixing text and images. 8. Model MIME Types: Support the transmission of 3D model data for applications like virtual reality or CAD software. Importance of MIME MIME types are crucial for proper handling and interpretation of content across various internet protocols, ensuring that different file types are correctly identified and processed by email clients, web browsers, and other applications. By specifying the MIME type, senders can communicate the nature of the content being shared, enabling recipients to display or interact with the data appropriately. Application of MIME Types - Email Attachments: MIME types are used to define the format of email attachments, allowing recipients to download and open files correctly. - Web Content: In web development, MIME types are essential for browsers to interpret and display various types of content, such as images, videos, and scripts. - Data Transfer: MIME types are employed in data transfer protocols to ensure that data is transmitted in the correct format and can be properly decoded at the receiving end.		
		OR		
Q. 08	a	Discuss how create templates for each view with an example. Creating Templates for Each View in Django Overview of Creating Templates for Views - In Django, templates are used to render the visual presentation of data. - Templates can be associated with views to display content to users. - Templates can be explicitly specified or inferred based on naming conventions. - Context processors in Django help automate variable setting in	L2	10M

templates without explicit declaration.

Example: Creating Templates for Generic Views

1. To build a list page of all publishers, a URLconf and template can be set up as follows:

```
 python
from django.views.generic import list_detail
from mysite.books.models import Publisher

publisher_info = {
    'queryset': Publisher.objects.all(),
    'template_name': 'publisher_list_page.html',
}

urlpatterns = patterns('',
    (r'^publishers/$', list_detail.object_list, publisher_info)
)
```

2. In the absence of explicitly specifying a template_name, Django infers the template based on naming conventions.
3. A simple template for this scenario might look like the following:

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

Utilizing Generic Views for Template Rendering

- Purpose: Django’s generic views simplify the process of presenting views on database content by providing built-in functionalities.
- Benefits: Generic views make generating list and detail views of objects easy and efficient for developers.
- Implementation: Configuration dictionaries are created in URLconf files to utilize generic views for different tasks.

Context Processors for Template Automation

- Context processors in Django help reduce redundancy by automatically setting variables in templates.
- By using RequestContext instead of Context when rendering a template, context processors can be applied to streamline variable setting.

	b	Explain Dynamic CSV using database	L2	10M

Dynamic CSV Using Database

Overview of Dynamic CSV and Database Interaction

Dynamic CSV generation involves creating CSV files on-the-fly based on data from a database, allowing for real-time updates and customization. This process is commonly used in web applications to export data for analysis or sharing.

Steps to Implement Dynamic CSV Using a Database

1. Data Retrieval : Fetch data from the database based on the user's criteria.
2. CSV Generation : Convert the retrieved data into CSV format

dynamically.
3. File Download : Provide a mechanism for users to download the generated CSV file.

Python Frameworks for Dynamic CSV Generation
- Django: Django provides functionalities to interact with databases and generate dynamic content, including CSV files.
- Pandas: Pandas library in Python offers robust tools for data manipulation and CSV generation.
- Flask: Flask framework can be used to create dynamic web applications with CSV export capabilities.

Database Considerations for Dynamic CSV
- Database Queries : Efficient database queries are crucial for retrieving data to populate the CSV.
- Data Formatting : Ensure data retrieved is formatted correctly for CSV generation.
- Security : Implement proper security measures to prevent unauthorized access to sensitive data.

Benefits of Dynamic CSV Using a Database
- Real-Time Data : Allows for up-to-date information exports.
- Customization : Data can be filtered and formatted as needed.
- Automation : Streamlines the process of generating and exporting data.

Module-5 Download				
Q. 09	a	Explain XHTML Http Request and Response. Discussing XHTML, HTTP Request, and Response XHTML - HTML and XHTML are fundamental markup languages for the web. - XHTML is seen as HTML brought back into the fold, making web development more structured. - XHTML is an HTML variant that is cleaner and more predictable for parsers compared to early HTML. - XHTML is good for web development and allows for machine interpretation in a sensible way. HTTP Request and Response XMLHttpRequest Functionality - XMLHttpRequest is commonly used to fetch XML, XHTML, HTML, plain text, and JSON. - The XMLHttpRequest object can handle various text formats, not limited to XML. - The responseText field contains the text served up, whether in XML or any other format. - The responseXML field holds a parsed XML document if the format turns out to be XML. XMLHttpRequest Properties - XMLHttpRequest properties include onreadystatechange, readyState, responseText, responseXML, status, and statusText. - The readyState progresses through states like Uninitialized, Open, Sent, and Loaded based on the XMLHttpRequest process. - The status field contains the HTTP code, such as 200 for OK, while statusText provides a short text description like OK. - Callback event handlers should check readyState and status before acting on server-provided data. XMLHttpRequest Methods - XMLHttpRequest methods include open(), send(), and onreadystatechange. - The open() method initializes a request, send() sends the request, and onreadystatechange handles state changes. - The XMLHttpRequest object's lifecycle involves transitioning through various states to complete the request and response process. Summary XHTML plays a crucial role in providing structured and predictable markup for web development, while XMLHttpRequest functionality allows for fetching various text formats, including XML, XHTML, and plain text. Understanding the properties and methods of XMLHttpRequest is essential for handling HTTP requests and responses effectively.	L2	10M
	b	Develop a registration page for student enrolment without page refresh	L2	10M

using AJAX.

To develop a registration page for student enrollment without page refresh using AJAX, you'll need to set up both the frontend and the backend. The frontend will use JavaScript and AJAX to submit the form, and the backend will handle the request and respond appropriately.

Let's go through the steps:

Step 1: Set up the Django Project and Application

1. **Create a new Django project and application:**

```
bash
Copy code
django-admin startproject student_enrollment
cd student_enrollment
python manage.py startapp enrollment
```

2. **Add the application to your project settings:**

   Open `student_enrollment/settings.py` and add `'enrollment'` to the `INSTALLED_APPS` list:

```
python
Copy code
INSTALLED_APPS = [
    ...
    'enrollment',
]
```

Step 2: Create Models and Forms

1. **Create the Student model:**

   Open `enrollment/models.py` and define the `Student` model:

```
python
Copy code
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=15)
    address = models.TextField()

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

2. **Create a Student form:**

   Open `enrollment/forms.py` and create the form for the `Student` model:

```
python
Copy code
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['first_name', 'last_name', 'email',
'phone', 'address']
```

3. **Apply migrations:**

```
bash
Copy code
python manage.py makemigrations
python manage.py migrate
```

Step 3: Create Views and URLs

1. **Create views to handle the form submission:**

   Open `enrollment/views.py` and add the following views:

```
python
Copy code
from django.shortcuts import render, redirect
from django.http import JsonResponse
from .forms import StudentForm

def student_registration(request):
    form = StudentForm()
    return render(request, 'enrollment/register.html',
{'form': form})

def submit_registration(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False,
'errors': form.errors})
    return JsonResponse({'success': False, 'errors':
'Invalid request'})
```

2. **Create URLs for the views:**

   Create `enrollment/urls.py` and add the following URLs:

```
python
Copy code
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.student_registration,
name='student_registration'),
    path('submit/', views.submit_registration,
name='submit_registration'),
]
```

3. **Include the application URLs in the project URL configuration:**

Open `student_enrollment/urls.py` and include the `enrollment` URLs:

```
python
Copy code
from django.contrib import admin
from django.urls import include, path

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

Step 4: Create Templates and JavaScript for AJAX

1. **Create a template for the registration form:**

   Create a directory `enrollment/templates/enrollment/` and create a file `register.html` inside it with the following content:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration</title>
    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
</head>
<body>
    <h1>Student Registration</h1>
    <form id="registrationForm" method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Register</button>
    </form>
    <div id="response"></div>

    <script>
        $(document).ready(function(){
            $(' registrationForm').on('submit',
function(event){
                event.preventDefault();
                $.ajax({
                    url: "{% url 'submit_registration'
%}",
                    type: "POST",
                    data: $(this).serialize(),
                    success: function(response){
                        if(response.success){

$(' response').html("<p>Registration successful!</p>");
                        } else {
                            let errors = "<ul>";
                            $.each(response.errors,
function(key, value){
                                errors += "<li>" + key
+ ": " + value[0] + "</li>";
                            });
                            errors += "</ul>";

$(' response').html(errors);
                        }
                    }
                });
```

```
                    });
                });
            </script>
        </body>
        </html>
```

Step 5: Run the Django Project

1. **Run the development server:**

   ```
   bash
   Copy code
   python manage.py runserver
   ```

2. **Access the registration page in your web browser:**

   Open your web browser and go to `http://127.0.0.1:8000/enrollment/register/`. You should see the registration form. When you submit the form, the AJAX request will handle the form submission without refreshing the page.

OR

Q. 10	a	Discuss how the setting of Javascript in Django.	L2	10M

Setting JavaScript in Django

Django Templating Language
- Django's templating language is designed to separate presentation and logic.
- It is optimized for designers to use for designing rather than programmers for programming.
- Unlike ASP, JSP, and PHP, Django's templating engine is not a programming language interspersed with HTML.
- Django templating provides enough power for presentation without allowing serious programming work where it doesn't belong.
- Learning the templating basics is comparable to learning simple HTML or SQL.
- The language is simple and easier to grasp than a regular programming language.

Integrating JavaScript and Static Content
- For the development server, static content like images, CSS, and other static content can be placed easily.
- Production use recommends using a separate server optimized for serving static media.
- Steps for serving static content in Django:
  1. Create a directory named static within the project.
  2. Edit the settings.py file to specify the media root and URL.
  3. Add specific code in settings.py to serve static media.

Development Server Considerations
- Django comes with a minimal, single-threaded web server for development purposes.
- This server is not suitable for production and has not undergone a security audit.

- Django creators advise using a good, serious web server for deployment.

Django Installation and Project Setup
- Instructions for installing Django and creating a sample project are provided.
- The process involves creating a project, setting up a basic template, and running the server for local development.
- Specific steps include creating a project, template, editing files like urls.py and settings.py, and running the development server.

Strengths and Challenges of JavaScript in Django
- JavaScript is considered an excellent but complex language with deep strengths.
- The language decisions in JavaScript can pose challenges, including unexpected behaviors.
- JavaScript is described as a language with mixed characteristics, being both excellent and challenging to work with.

	b	Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.	L2	10M

To develop a search application in Django using AJAX that displays courses enrolled by a student being searched, follow these steps:

Step 1: Set up the Django Project and Application

1. **Create a new Django project and application:**

```
bash
Copy code
django-admin startproject course_search
cd course_search
python manage.py startapp students
```

2. **Add the application to your project settings:**

   Open `course_search/settings.py` and add `'students'` to the `INSTALLED_APPS` list:

```
python
Copy code
INSTALLED_APPS = [
    ...
    'students',
]
```

Step 2: Create Models

1. **Create the Student and Course models:**

   Open `students/models.py` and define the `Student` and `Course` models:

```
python
Copy code
from django.db import models
```

```
class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)

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

class Course(models.Model):
    name = models.CharField(max_length=100)
    students = models.ManyToManyField(Student,
related_name='courses')

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

2. **Apply migrations:**

```
bash
Copy code
python manage.py makemigrations
python manage.py migrate
```

Step 3: Create Views and URLs

1. **Create views to handle the search and display of courses:**

   Open `students/views.py` and add the following views:

```
python
Copy code
from django.shortcuts import render
from django.http import JsonResponse
from .models import Student

def search_student(request):
    return render(request, 'students/search.html')

def get_courses(request):
    if request.is_ajax() and 'email' in request.GET:
        email = request.GET.get('email')
        try:
            student = Student.objects.get(email=email)
            courses = student.courses.all()
            courses_list = [{'name': course.name} for
course in courses]
            return JsonResponse({'success': True,
'courses': courses_list})
        except Student.DoesNotExist:
            return JsonResponse({'success': False,
'message': 'Student not found'})
    return JsonResponse({'success': False, 'message':
'Invalid request'})
```

2. **Create URLs for the views:**

   Create `students/urls.py` and add the following URLs:

```
python
Copy code
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('search/', views.search_student,
name='search_student'),
    path('get_courses/', views.get_courses,
name='get_courses'),
]
```

3. **Include the application URLs in the project URL configuration:**

   Open `course_search/urls.py` and include the `students` URLs:

```
python
Copy code
from django.contrib import admin
from django.urls import include, path

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

Step 4: Create Templates and JavaScript for AJAX

1. **Create a template for the search page:**

   Create a directory `students/templates/students/` and create a file `search.html` inside it with the following content:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Student Course Search</title>
    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
</head>
<body>
    <h1>Search Courses by Student Email</h1>
    <form id="searchForm" method="get">
        <label for="email">Student Email:</label>
        <input type="email" id="email" name="email"
required>
        <button type="submit">Search</button>
    </form>
    <div id="response"></div>

    <script>
        $(document).ready(function(){
            $('#searchForm').on('submit',
function(event){
                event.preventDefault();
                $.ajax({
                    url: "{% url 'get_courses' %}",
                    type: "GET",
                    data: $(this).serialize(),
                    success: function(response){
                        if(response.success){
                            let courses = "<ul>";
                            $.each(response.courses,
function(index, course){
                                courses += "<li>" +
course.name + "</li>";
                            });
                            courses += "</ul>";
$(' response').html(courses);
                        } else {
                            $(' response').html("<p>" +
response.message + "</p>");
                        }
                    }
                });
            });
        });
    </script>
</body>
</html>
```

Step 5: Run the Django Project

1. **Run the development server:**

```
bash
Copy code
python manage.py runserver
```

2. **Access the search page in your web browser:**

   Open your web browser and go to `http://127.0.0.1:8000/students/search/`. You should see the search form. When you enter a student's email and submit the form, the AJAX request will handle the form submission and display the courses without refreshing the page.

*****Bloom's Taxonomy Level: Indicate as L1, L2, L3, L4, etc. It is also desirable to indicate the COs and POs to be attained by every bit of questions.