**PART A**

**Experiment No. 07**

**A.1 Aim:** Create Django application using view and template for online portfolio showcase

**Objective:** To learn and understand how use view and template in Django

HTML, CSS, Javascript

**A.3 Outcome:**

After successful completion of these applications students will be able to understand Django Framework, Creating Views and Templates:, Dynamic Content Rendering and using css and js in Django

**A.4 Theory**

Django Overview:

* Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the MVC (Model-View-Controller) architectural pattern, where:
  + Model: Represents the data structure.
  + View: Handles the business logic and returns the appropriate response.
  + Template: Defines the presentation layer (HTML).

Django Views:

* A view function in Django is a Python function that takes a web request and returns a web response. It can return HTML content, redirect to another URL, or return a JSON response.

Django Templates:

* Templates are HTML files that define how the content is presented. They allow for dynamic content generation using Django's templating language, which includes template tags and filters.

URL Routing:

* Django uses a URL dispatcher to route incoming requests to the appropriate view based on the requested URL pattern defined in the urls.py file.

Static and Media Files:

* Understanding how to manage static files (like CSS and JavaScript) and media files (like images) in Django is essential for enhancing the visual appeal and functionality of the application.

Context Data:

* The context is a dictionary that contains the data passed from the view to the template. This data can be rendered in the HTML using template syntax (e.g., {{ variable\_name }}).

**Scenario: Online Portfolio Showcase**

Overview:  
Tom, a budding web developer, is building his online portfolio to showcase his projects, skills, and personal interests. He wants to create a simple web application using Django that allows visitors to learn about him, view his work, and interact with the content through a user-friendly interface. He aims to make it visually appealing by incorporating images, CSS styling, and JavaScript functionality.

**Project Features:**

* A homepage that welcomes visitors and provides an overview of Tom’s work.
* A dedicated page showcasing a list of items, such as projects he’s completed and skills he possesses.
* Integration of basic CSS for styling and JavaScript for interactivity, enhancing the user experience.

**Problem Statement 1: Content Display Issues**

Challenge:  
When users visit the my\_view page, they expect to see a well-organized list of Tom's skills and projects along with his personal details. However, upon testing, Tom discovers that the dictionary data for his personal information is not displaying correctly.

Questions to Address:

* How can Tom ensure that the dictionary data is accessed and displayed correctly in the template?
* What changes are needed in the my\_view.html file to fix the data access?

**Problem Statement 2: Styling and Aesthetic Enhancements**

**Challenge:**  
The current CSS styles are too minimal, and the h1 headings do not stand out as intended. Tom wants to improve the visual appeal of his portfolio by enhancing the styling of the headings and other text elements. He also wants the images to have rounded corners to give them a softer look.

**Questions to Address:**

* What specific CSS rules can Tom implement to improve the styling of the headings and text in my\_view.html and indexstatic.html?
* How can he ensure that the images maintain a consistent and visually appealing style across the site?

**Problem Statement 3: Interactivity and User Engagement**

Challenge:  
Tom wants to make his website more engaging by adding interactivity. He wishes to alert users when they visit the site and show a friendly greeting. However, the JavaScript code he has written is not functioning as expected.

**Questions to Address:**

* What steps can Tom take to troubleshoot the JavaScript code to ensure the alert function works correctly when the page loads?
* How can he implement additional JavaScript features to further enhance user engagement on the site, such as dynamically updating content or responding to user actions?

Hints:

my\_project/

│

├── my\_app/

│ ├── views.py

│ ├── templates/

│ │ ├── my\_view.html

│ │ ├── indexstatic.html

│ ├── static/

│ │ ├── css/

│ │ │ ├── styles.css

│ │ ├── js/

│ │ │ ├── scripts.js

│ │ ├── images/

│ │ │ ├── pic.png

├── my\_project/

│ ├── settings.py

│ ├── urls.py

**Sampe code for 1. views.py**

def my\_view(request):

items = ["Item 1", "Item 2", "Item 3", "Item 4"]

my\_dict = {"name": "Tom", "age": 30, "city": "New York"}

context = {

"author": "Rajiv Gupta",

"data": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

"itemList": items,

"my\_dict": my\_dict,

}

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. : S020 | Name: Husain Chhil |
| Class : MBA.Tech DS | Batch : J1 |
| Date of Experiment : 10.10.24 | Date/Time of Submission : 10.10.24 |
| Grade : |  |

**B.1 Code:**

my\_view.html

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>My View</title>

    <!-- Load the static template tag library -->

    {% load static %}

    <!-- Link to the external stylesheet -->

    <link *rel*="stylesheet" *href*="{% static 'css/styles.css' %}">

</head>

<body>

    <h1>Welcome to Tom's Portfolio</h1>

    <!-- Displaying personal information from the dictionary -->

    <h2>About Tom:</h2>

    <p>Name: {{ my\_dict.name }}</p>

    <p>Age: {{ my\_dict.age }}</p>

    <p>City: {{ my\_dict.city }}</p>

    <!-- Displaying the list of items (projects, skills, etc.) -->

    <h2>Projects and Skills</h2>

    <ul>

        {% for item in itemList %}

        <li>{{ item }}</li>

        {% endfor %}

    </ul>

</body>

</html>

indexstatic.html

<!DOCTYPE *html*>

<html *lang*="en">

  <head>

    <meta *charset*="UTF-8" />

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

    <title>Tom's Portfolio</title>

    <link *rel*="stylesheet" *href*="{% static 'css/styles.css' %}" />

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

  </head>

  <body>

    <div *class*="container">

      <h1>Welcome to Tom's Portfolio</h1>

      <p *id*="greeting-message">Feel free to explore!</p>

      <!-- Button to trigger the greeting update -->

      <button *onclick*="showGreeting()">Click me for a greeting</button>

    </div>

  </body>

</html>

script.js

// Alert users when they visit the site

window.onload = function () {

  alert("Welcome to Tom's Portfolio!");

};

// Additional feature: dynamically update content when clicking a button

function showGreeting() {

  const message = document.getElementById("greeting-message");

  message.innerText = "Thanks for visiting!";

}

styles.css

/\* General styles for the body \*/

body {

  font-family: Arial, sans-serif;

  margin: 0;

  padding: 20px;

  background-color: #f4f4f4;

  color: #333;

}

/\* Styling the h1 headings \*/

h1 {

  color: #2c3e50;

  font-size: 2.5rem;

  text-align: center;

  margin-bottom: 20px;

}

/\* Styling the h2 subheadings \*/

h2 {

  color: #2980b9;

  font-size: 1.8rem;

  margin-bottom: 15px;

}

/\* Styling the unordered list items \*/

ul {

  list-style-type: square;

  padding-left: 20px;

}

li {

  font-size: 1.2rem;

  margin-bottom: 10px;

}

/\* Styling images to have rounded corners \*/

img {

  border-radius: 15px;

  max-width: 100%;

  height: auto;

  display: block;

  margin: 0 auto 20px;

}

/\* Styling for a responsive container \*/

*.container* {

  max-width: 800px;

  margin: 0 auto;

  padding: 20px;

  background-color: #fff;

  border-radius: 10px;

  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

views.py

from django.shortcuts import render

def my\_view(*request*):

    items = ["Project A", "Skill B", "Project C", "Skill D"]

    my\_dict = {"name": "Tom", "age": 30, "city": "New York"}

    context = {

        "author": "Rajiv Gupta",

        "data": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

        "itemList": items,

        "my\_dict": my\_dict,

    }

    return render(*request*, 'my\_view.html', context)

myapp/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('portfolio', views.my\_view, *name*='my\_view'),

]

myproject.urls.py

from django.contrib import admin

from django.urls import path, include

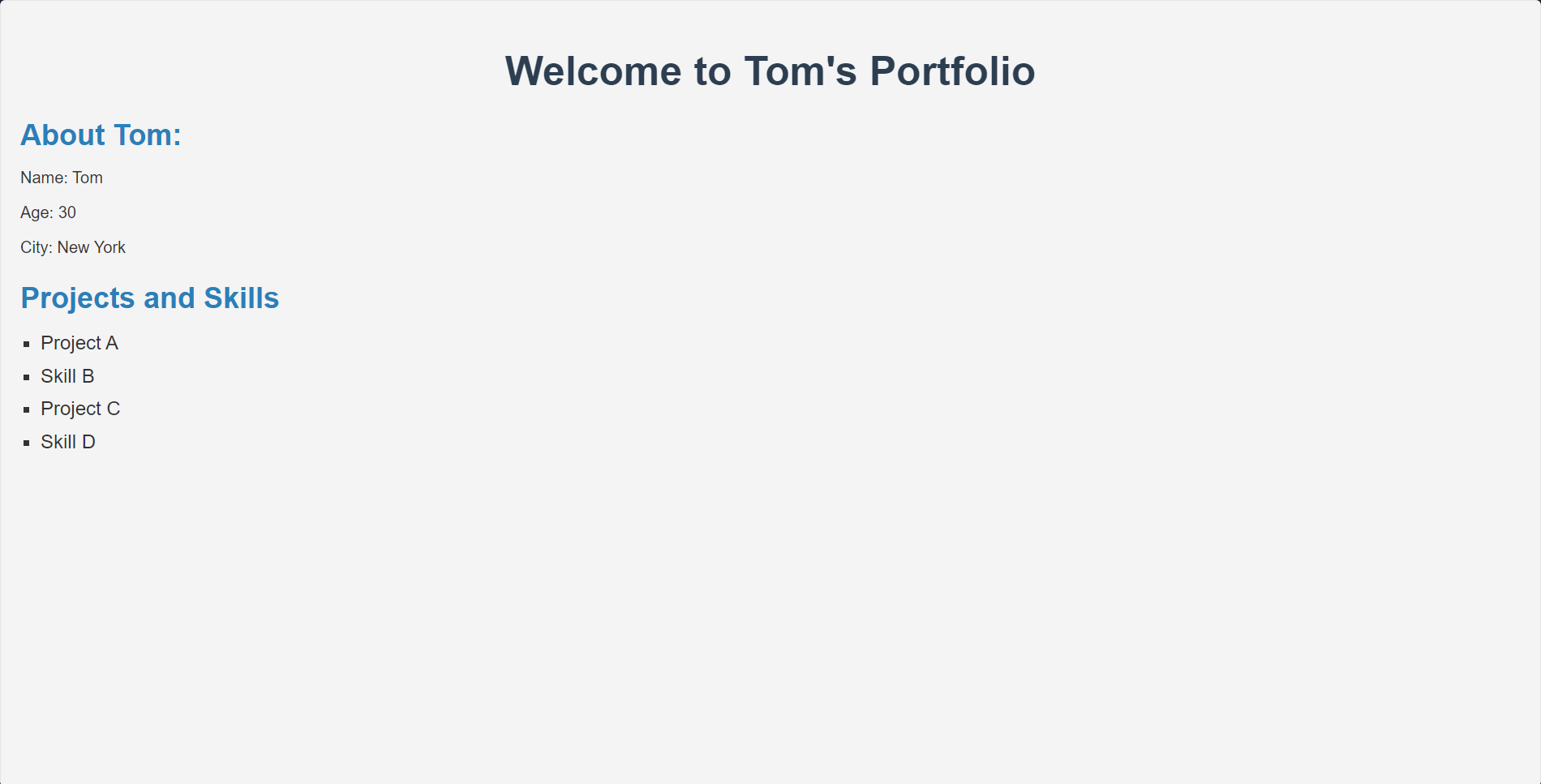
urlpatterns = [

    path('admin/', admin.site.urls),

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

]

**B.2 Output**



**B.3 Conclusion:**

In this practical, we explored the essential components of Django, including URL routing, templates, context passing, and static file management. We learned how Django efficiently handles URL requests and routes them to the appropriate view functions. Through the use of templates, we discovered how dynamic content can be rendered on web pages, offering flexibility in the design and presentation of content.

Moreover, we studied the use of Django template tags, which facilitate the inclusion of logic and dynamic data in HTML templates. The importance of the context dictionary for passing data between views and templates was also emphasized. We further understood the role of static files in a Django project and how they are managed using Django's built-in tools.

This practical has provided us with a foundational understanding of Django's web development capabilities, empowering us to build responsive and interactive web applications that efficiently handle both static and dynamic content.

**B.3 Observations and Learning:**

Django's URL Routing:

Django uses a clean URL routing system, where URL patterns are defined in the urls.py file to map specific URLs to corresponding views.

This decouples the URL structure from the underlying code, making it more user-friendly and easy to manage.

By using regular expressions or path converters, we can handle dynamic URLs effectively.

Dynamic Content Rendering with Templates:

Templates in Django allow for dynamic content rendering by separating the backend logic from the presentation.

We can pass data from views to templates, making it easy to display dynamic content like user information, database results, and form data without hardcoding the values.

Template Tags:

Django’s template tags provide tools for inserting dynamic data into HTML. Common tags include {% for %} for looping, {% if %} for conditional statements, and {% static %} for including static resources like CSS and JavaScript files.

This allows templates to adapt based on the data passed to them, enhancing flexibility and reusability.

Context Data Passing:

In Django, views pass data to templates using a dictionary called context. This dictionary contains key-value pairs where the keys are the variable names used in the template, and the values are the data being passed.

For example, a dictionary {"name": "Tom", "age": 30} can be passed to a template, where the variables {{ name }} and {{ age }} will display the respective values.

Static File Management:

Static files, like CSS, JavaScript, and images, are stored separately from the application logic and are served to enhance the visual design and interactivity of the website.

Django manages static files using the {% static %} tag to refer to these files and the STATIC\_URL setting to define their location. Static files are collected into a single location during production using the collectstatic command.

**B.4 Questions of Curiosity**

How does Django handle URL routing, and what role does the urls.py file play in this process?

Django’s URL routing system matches URLs from user requests to the correct view functions. The urls.py file plays a key role by defining these patterns and mapping them to views, which then handle the request and return a response.

What is a template in Django, and how does it allow for dynamic content rendering?

A template in Django is an HTML file with placeholders for dynamic content. It allows for dynamic rendering by filling in the placeholders with data passed from the view using the context, making the web page content adaptable based on the data provided.

What are Django template tags, and how are they used to include dynamic data in templates?

Django template tags are special syntax that allows for logic and dynamic content within templates. Tags like {% for %}, {% if %}, and {% static %} enable looping, conditional rendering, and including static files in the HTML, respectively.

Describe how context data is passed from a view to a template in Django. What format does this data take?

Context data is passed from a view to a template as a dictionary, where each key corresponds to a variable used in the template. The values in the dictionary are the actual data (like strings, lists, or objects) that get rendered in the template.

What are static files in Django, and how do you manage them within a Django project?

Static files are assets such as images, CSS, and JavaScript that enhance the design and functionality of a web page. Django uses the {% static %} template tag to reference these files and the STATIC\_URL setting to define their location. During production, the collectstatic command gathers all static files into a single directory for serving.