# Competency Based Learning Material (CBLM)

## Web Application Development with Python

**Level-4**

## Module: Customizing UI

**Code: CBLM- OU-ICT-WADP-05-L4-V1**

**National Skills Development Authority**
**Chief Advisor's Office**
**Government of the People's Republic of Bangladesh**

## Copyright

National Skills Development Authority
Chief Advisor's Office
Level10-11, Biniyog Bhaban,
E-6 / B, Agargaon, Sher-E-Bangla Nagar Dhaka-1207, Bangladesh.
Emailec@nsda.gov.bd
Websitewww.nsda.gov.bd.
National Skills Portalhttp:\\skillsportal.gov.bd

Approved by the Authority…… meeting held on **.....**

## How to use this Competency Based Learning Material (CBLM)

The module contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information sheet s.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information sheet s** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information sheet** . **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information sheet** . Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information sheet s. This is your opportunity to practise the job. You may need to practise the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Module

Table of Contents

# Module Content

| | |
|---|---|
| **Unit of Competency** | **Customize UI** |
| **Unit Code** | OU-ICT-WADP-05-L4-V1 |
| **Module Title** | **Customizing UI** |
| Module Descriptor | This module covers the knowledge, skills and attitudes required to customize UI<br>It includes the task of using API, implementing custom API, firebase API and Firebase Cloud Messaging (FCM) |
| Nominal Hours | **2**0 Hours |
| Lerning Outcome | After completing the practice of the module, the trainees will be able to perform the following jobs<br>1. Manage static resources<br>2. Apply Bootstrap on templates<br>3. Implement Accessibility-friendly methods and techniques |

**Assessment Criteria**

1. Static directory is set up
2. Static resources are collected and reposited.
3. Static-based url is set up
4. Statics files are used on templates
5. Bootstrap form class is applied
6. Card layout is used
7. Table is designed
8. Navigation is applied using Navbar
9. Validation alert is used
10. The concept of User Accessibility is introduced from UX perspective.
11. Best practices are introduced.
12. Corresponding methods and techniques are set up.

## Learning Outcome 1: Manage static resources

| | |
|---|---|
| Assessment Criteria | 1. Static directory is set up<br>2. Static resources are collected and reposited.<br>3. Static-based url is set up<br>4. Statics files are used on template |
| Conditions and Resources | 1. Real or simulated workplace<br>2. CBLM<br>3. Handouts<br>4. Laptop<br>5. Multimedia Projector<br>6. Paper, Pen, Pencil, Eraser<br>7. Internet facilities<br>8. White board and marker<br>9. Audio Video Device |
| Contents | 1. Set up of static directory<br>2. Static resources<br>3. Set up of static-based url<br>4. Use of static files on templates |
| Activities/job/Task | 1. Setting Up Static Files in Django |
| Training Methods | 1. Discussion<br>2. Presentation<br>3. Demonstration<br>4. Guided Practice<br>5. Individual Practice<br>6. Project Work<br>7. Problem Solving<br>8. Brainstorming |
| Assessment Methods | Assessment methods may include but not limited to<br>1. Written Test<br>2. Demonstration<br>3. Oral Questioning<br>4. Portfolio |

## Learning Experience 1: Manage static resources

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials 'Manage static resources' |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Manage static resources" | 2. Read Information sheet 1: Manage static resources<br>3. Answer Self-check 1: Manage static resources<br>4. Check your answer with Answer key 1: Manage static resources |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job Sheet-1: Setting Up Static Files in Django |

# Information sheet 1: Manage static resources

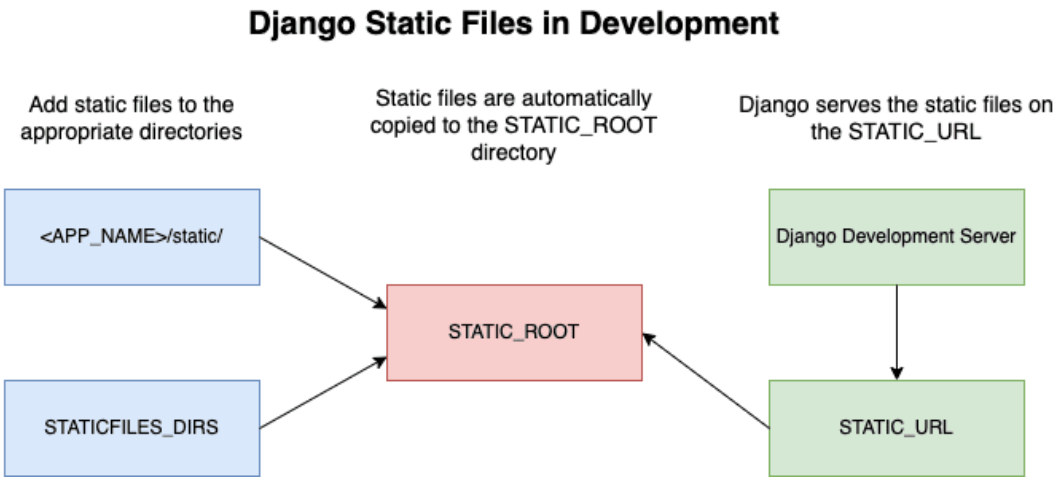
**Learning Objective:**

After completion of this Information sheet , the learners will be able to explain, define and interpret the following contents:

1.1. Setup Static directory
1.2. Collect and reposit Static resources.
1.3. Setup Static-based url
1.4. Statics files on templates

## 1.1. Setting Up the Static Directory

The static directory serves as the central location for storing all your project's non-Python files – primarily CSS, JavaScript, and image files – that enhance the presentation and interactivity of your web pages. Here's how to set it up:




**1. Create the Directory:**

- Navigate to your Django project's root directory using the terminal or command prompt.
- Use the mkdir (make directory) command to create a directory named static:

Bash

mkdir static

**2. Organize Your Static Files (Optional):**

While you can simply dump all static files into the static directory, creating subdirectories is a good practice for better organization. Here's a common structure:

```
project_root/
    static/
        css/  # For all your CSS stylesheets
            style.css
            main.css
        js/   # For all your JavaScript files
            script.js
            main.js
        images/ # For all your image files
            logo.png
            banner.jpg
    ... (other project files)
```

By using subdirectories, you improve project maintainability and can easily reference specific files within templates.

**3. Next Steps:**

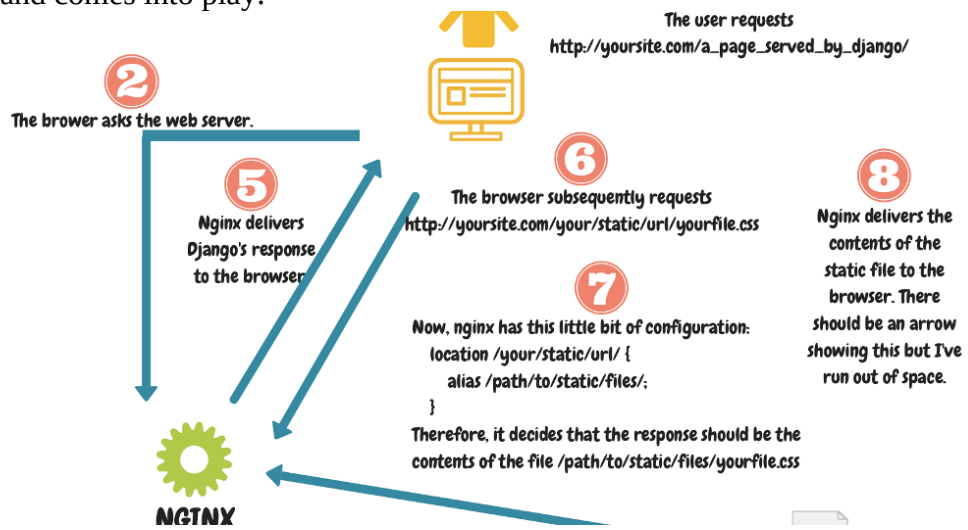
Once you've created the static directory with the desired structure, you can proceed with:

- **Collecting and Repositing Static Resources:** Learn how to gather these files and place them in a designated location for serving.
- **Setting Up Static-Based URL:** Configure the URL prefix that will be used to access static files in templates.
- **Using Static Files in Templates:** Discover how to utilize the static template tag to reference these files in your HTML templates.

These steps work together to make your static resources accessible within your Django application.

## 1.2. Collecting and Repositing Static Resources

After setting up your static directory, you need to gather those files and prepare them for serving by your web server. This is where Django's collectstatic management command comes into play.




**The collectstatic Command:**

1. **Running the Command:**
    - Open your terminal or command prompt and navigate to your Django project's root directory.
    - Execute the following command:

Bash
python manage.py collectstatic

    - This command scans for all static files within your static directory (and its subdirectories if present) based on the settings you've configured.
2. **Repositing Static Files:**
    - The collectstatic command gathers the discovered static files and stores them in a location specified by the STATIC_ROOT setting in your settings.py file.
    - This location is typically not within your project's root directory for security reasons. It's recommended to keep them separate from your application code for easier management and deployment.
3. **Understanding STATIC_ROOT:**
    - In your project's settings.py file, locate the STATIC_ROOT setting and ensure it's configured with the desired path where you want to store the collected static files.
    - You can set it like this:

Python
STATIC_ROOT = os.path.join(BASE_DIR, "staticfiles")

        - This example uses the os.path.join function to construct a path relative to your project's root directory (BASE_DIR). Here, the collected files will be placed in a directory named staticfiles within the project's root directory.
    - Remember to adjust the path according to your project's structure and desired location (outside your project directory is recommended).

**Additional Considerations:**

- **Development Server vs. Production Environment:**
  - During development, you might prefer to use a development server (like Django's built-in development server) that automatically serves static files directly from your project's static directory. This avoids the need to run collectstatic every time you make a change. However, ensure your server is configured to serve from this location.
  - In production environments, where security is key, your web server should be configured to serve static files from the STATIC_ROOT directory. This allows users to access your web pages with the necessary static resources (CSS, JavaScript, images).
- **STATICFILES_DIRS (Optional):**
  - You might have static files scattered across various apps within your Django project. Use the STATICFILES_DIRS setting (a list of tuples) in settings.py to specify additional locations where the collectstatic command should search for static files.

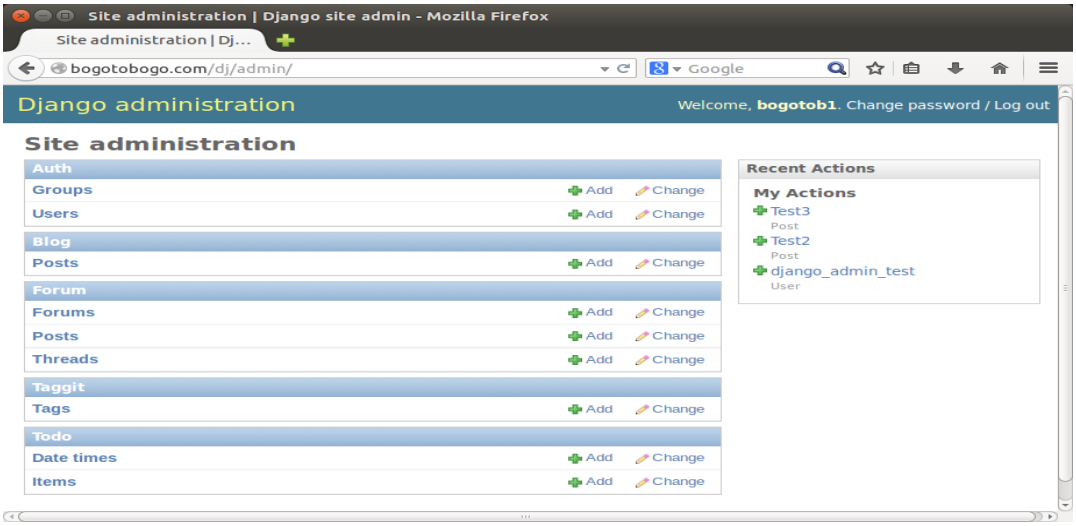
Python
```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "app_name/static"),
]
```

  - This example tells collectstatic to also search for static files in the static directory within the app_name app.

By understanding the collectstatic command and properly setting up STATIC_ROOT (and optionally STATICFILES_DIRS), you can effectively manage static resources in your Django project, ensuring a clean and organized approach for serving them to your users.

## 1.3. Setting Up Static-Based URL

The static-based URL acts as a prefix that Django uses to construct the complete URL for accessing your static files (CSS, JavaScript, images) in your templates. Here's how to configure it:



**1. Define STATIC_URL in settings.py:**

- Locate the settings.py file within your Django project's root directory.
- Add or update the STATIC_URL setting with the desired prefix for your static files. A common convention is to use /static/, but you can customize it to your preference.

Python
```
STATIC_URL = "/static/"
```

**2. How STATIC_URL Works:**

- When you use the static template tag within your HTML templates to reference a static file, Django constructs the final URL by combining the configured STATIC_URL with the relative path you provide:

  HTML

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

  - In this example:
    - STATIC_URL is set to /static/
    - The relative path is css/style.css
  - Django will construct the complete URL as: http://yourdomain.com/static/css/style.css

**3. Serving Static Files:**

- Remember that the collectstatic command gathers static files and stores them in the STATIC_ROOT directory you configured in settings.py.
- During development, you might use a development server that automatically serves static files directly from your project's static directory.
- In production environments, ensure your web server is configured to serve static files from the STATIC_ROOT directory. This allows your web server to locate and deliver the static files to users based on the constructed URL using STATIC_URL and the relative path.

**Additional Considerations:**

- **Customizing STATIC_URL:** While /static/ is a common convention, you can choose a different prefix if it better aligns with your project structure or preferences.
- **STATICFILES_DIRS (Optional):** If your static files are scattered across various apps within your project, ensure STATICFILES_DIRS (a list of tuples) in settings.py points to those additional locations for the collectstatic command to search.

By setting STATIC_URL appropriately and considering these additional points, you establish a mechanism for Django to construct correct URLs for accessing your static files, ensuring they are delivered as expected within your web pages.

## 1.4. Using Static Files in Templates

Once you've set up your static directory, collected your static resources, and configured the static-based URL, it's time to leverage them within your Django templates. Here's how:

**1. The static Template Tag:**

- Django provides the static template tag to reference static files (CSS, JavaScript, images) in your HTML templates.
- This tag helps construct the complete URL for the file based on the configured STATIC_URL and the relative path you provide.

**2. Basic Usage:**

HTML

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

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

<img src="{% static 'images/logo.png' %}" alt="Logo">
```

- In these examples:
  - The static tag is used with the relative paths of the static files (e.g., css/style.css).
  - Django will combine the configured STATIC_URL (e.g., /static/) with these paths to generate the final URLs.

**3. Advantages of the static Tag:**

- **Flexibility:** You can reference static files located anywhere within your static directory structure.
- **Consistency:** Ensures URLs are constructed correctly regardless of your project's deployment location.
- **Avoids Hardcoding URLs:** Eliminates the need to manually write out the complete URL for each static file, preventing potential errors if you change your deployment setup.

**4. Additional Considerations:**

- **STATIC_URL Configuration:** Ensure you've defined STATIC_URL in your settings.py file with the appropriate prefix.
- **Development vs. Production:**
  - During development, your development server might serve static files directly from your project's static directory.
  - In production, your web server should be configured to serve static files from the STATIC_ROOT directory based on the constructed URLs using STATIC_URL.

## Self-Check Sheet 1: Manage static resources

**1. Which directory stores your Django project's non-Python files like CSS, JavaScript, and images?**

- A. templates
- B. static
- C. urls
- D. models

**2. What Django command gathers static files from your project and stores them in a designated location?**

- A. python manage.py migrate
- B. python manage.py runserver
- C. python manage.py collectstatic
- D. python manage.py createsuperuser

**3. Where does the `STATIC_ROOT` setting in `settings.py` specify the collected static files to be stored?**

- A. Within the `static` directory itself
- B. In a subdirectory within the app where the static files reside
- C. In a designated location outside the project directory
- D. It depends on the web server configuration.

**4. What is the purpose of the `static` template tag in Django templates?**

- A. To define variables used within the template
- B. To create a link to another template
- C. To reference static files (CSS, JavaScript, images) with correct URLs
- D. To include a reusable component within the template

**5. During development, how might Django serve static files?**

- A. It always requires running the `collectstatic` command.
- B. Your development server might serve them directly from the project's `static` directory.
- C. It requires a separate web server configuration for static files.
- D. There's no way to serve static files during development.

## Answer Key  1: Manage static resources

**1. Which directory stores your Django project's non-Python files like CSS, JavaScript, and images?**

**Answer:** B. static

**2. What Django command gathers static files from your project and stores them in a designated location?**

**Answer:** C. python manage.py collectstatic

**3. Where does the `STATIC_ROOT` setting in `settings.py` specify the collected static files to be stored?**

**Answer:** C. In a designated location outside the project directory

**4. What is the purpose of the `static` template tag in Django templates?**

**Answer:** . To reference static files (CSS, JavaScript, images) with correct URLs

**5. During development, how might Django serve static files?**

**Answer:** B. Your development server might serve them directly from the project's `static` directory.

## **Job Sheet**-1: **Setting Up Static Files in Django**

### **UoC Cover**

OU-ICT-WADP-05- L4-V1: Customized UI

### **Working Procedure / Steps**

1. Setting Up the Static Directory
    - Create the directory: Use the mkdir static command in your terminal to create a directory named static in your project's root directory.
    - Organize your files (Optional): It's recommended to create subdirectories within static for better organization.
2. Collecting and Repositing Static Resources
    - The collectstatic command: Use the python manage.py collectstatic command to gather static files from your static directory (and subdirectories) and store them in a designated location.
    - Understanding STATIC_ROOT: Configure the STATIC_ROOT setting in your settings.py file to specify the desired path where these collected files will be placed.
    - Development vs. Production:
        - During development, your development server might serve static files directly from the project's static directory.
        - In production environments, configure your web server to serve static files from the STATIC_ROOT directory.
    - STATICFILES_DIRS (Optional): If static files reside in various apps within your project, use the STATICFILES_DIRS setting (a list of tuples) to specify additional locations for the collectstatic command to search.
3. Setting Up Static-Based URL
    - Define STATIC_URL in settings.py: Set the STATIC_URL setting in your settings.py file with the desired prefix for your static files. A common convention is /static/.
    - How STATIC_URL works: When you use the static template tag in your templates, Django combines STATIC_URL with the relative path you provide to construct the complete URL for the static file.
    - Serving Static Files:
        - Development: Your development server might serve static files directly from the project's static directory.
        - Production: Ensure your web server is configured to serve static files from the STATIC_ROOT directory.
4. Using Static Files in Templates
    - The static template tag: Django provides the static template tag to reference static files within your HTML templates.

## Specification Sheet 1: Setting Up Static Files in Django

**Technical Requirements**

- **Programming Language:** Python 3.7 or later
- **Framework:** Django 3.2 or later
- **Text Editor/IDE:** Visual Studio Code, PyCharm, Sublime Text (or any preferred code editor)

**Tools and Equipment**

- **Computer:** A computer with sufficient processing power, RAM, and storage.
- **Terminal:** A command-line interface for executing commands.

## Learning Outcome 2: Apply Bootstrap on templates

| | |
|---|---|
| **Assessment Criteria** | 1. Bootstrap form class is applied<br>2. Card layout is used<br>3. Table is designed<br>4. Navigation is applied using Navbar<br>5. Validation alert is used |
| **Conditions and Resources** | 1. Real or simulated workplace<br>2. CBLM<br>3. Handouts<br>4. Laptop<br>5. Multimedia Projector<br>6. Paper, Pen, Pencil, Eraser<br>7. Internet facilities<br>8. White board and marker<br>9. Audio Video Device |
| **Contents** | 1. Bootstrap form class<br>2. Card layout<br>3. Design table<br>4. Apply navigation<br>5. Use validation alert |
| **Activities/job/Task** | 1. Create a Navigation Bar with Bootstrap in Django Templates |
| **Training Methods** | 1. Discussion<br>2. Presentation<br>3. Demonstration<br>4. Guided Practice<br>5. Individual Practice<br>6. Project Work<br>7. Problem Solving<br>8. Brainstorming |
| **Assessment Methods** | Assessment methods may include but not limited to<br>1. Written Test<br>2. Demonstration<br>3. Oral Questioning<br>4. Portfolio |

## Learning Experience 2: Apply Bootstrap on templates

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials 'Apply Bootstrap on templates' |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Apply Bootstrap on templates" | 2. Read Information sheet 2: Apply Bootstrap on templates<br>3. Answer Self-check 2: Apply Bootstrap on templates<br>4. Check your answer with Answer key 2: Apply Bootstrap on templates |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job Sheet-2: Create a Navigation Bar with Bootstrap in Django Templates |

# Information sheet 2: Apply Bootstrap on templates

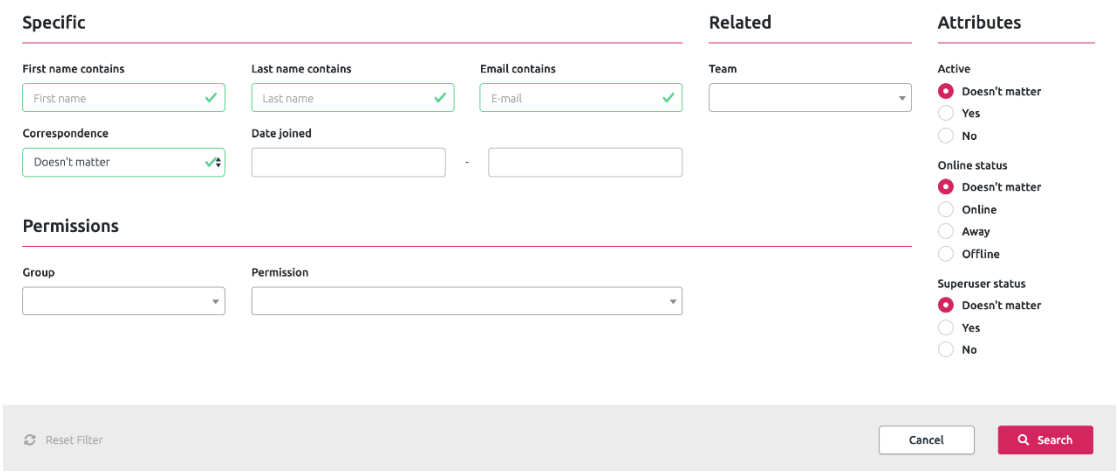
**Learning Objective:**

After completion of this Information sheet , the learners will be able to explain, define and interpret the following contents:

2.1. Bootstrap form class
2.2. Card layout
2.3. Design Table
2.4. Navigation using Navbar
2.5. Use Validation alert

## 2.1. Bootstrap Form Classes

In Django templates, you can leverage Bootstrap's pre-built form classes to style your Django forms effectively.




### 1. Include Bootstrap CSS

- Download Bootstrap (or use a CDN link) and include its CSS file in your base template (e.g., base.html) to make the classes available throughout your application.

```
HTML
<!DOCTYPE html>
<html>
<head>
   <title>My Django App</title>
   <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
integrity="sha385-
```

```
qKJQzCGN5vhwqhwGxyELzDKsRiRiBsBlutlnpQ6VB6qvyxYoC2VOGQcQyHizsR
" crossorigin="anonymous">
   </head>
<body>
   </body>
</html>
```

**2. Apply Form Classes:** In your form templates (e.g., form.html), inherit from your base template and then apply Bootstrap classes to specific form elements.

HTML

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

{% block content %}
 <form class="form-horizontal">  <div class="form-group">
    <label for="name" class="col-sm-2 control-label">Name:</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" id="name" name="name"
placeholder="Enter your name">
    </div>
  </div>
  <div class="form-group">
    <label for="email" class="col-sm-2 control-label">Email:</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="email" name="email"
placeholder="Enter your email">
    </div>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
 </form>
{% endblock %}
```

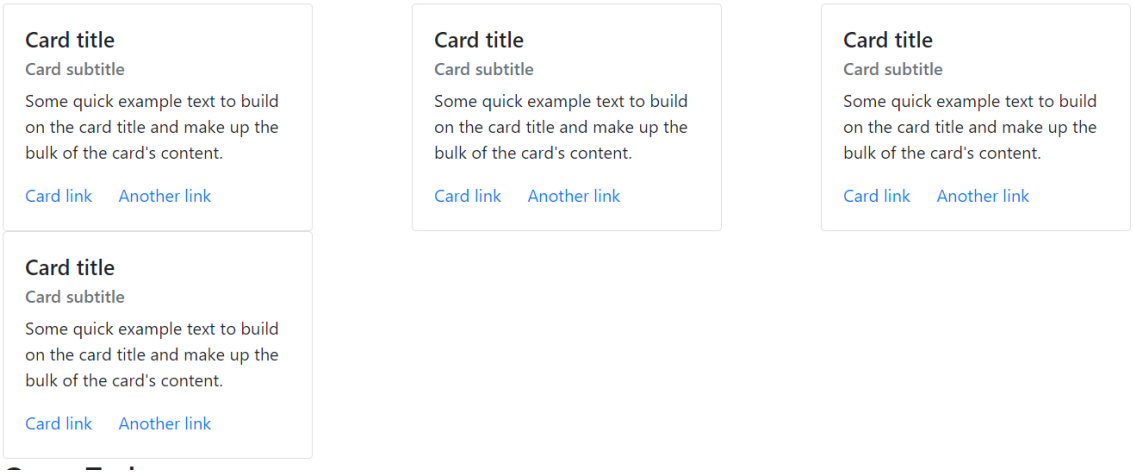
**Explanation**

- form-horizontal: This class lays out form elements horizontally, with labels on the left and inputs on the right.
- form-control: This class applies basic styling (padding, rounded corners, etc.) to form elements like inputs and textareas.

By effectively applying Bootstrap form classes, you can create visually appealing and user-friendly forms in your Django application.

## 2.2. Card Layout

Bootstrap's card component provides a versatile way to structure and style content sections in your Django templates. Here's how to integrate them




## 1. Prerequisites

- Ensure Bootstrap is downloaded and included in your project (refer to Bootstrap documentation for installation instructions: https://getbootstrap.com/docs/5.2/getting-started/introduction/).

## 2. Basic Card Structure

Wrap your content within a parent element with the .card class.

```
HTML
<div class="card">
 </div>
```

## 3. Adding Content

- You can include various elements within the card, such as:
  - Images: Use the .card-img-top class on an <img> tag for an image at the top of the card.
  - Headers: Use the .card-title class on an <h1> or <h2> tag for the card title.
  - Body Content: Include your main content within the .card-body class.
  - Buttons, links, and other elements can be placed inside .card-body or a dedicated .card-footer class.

## Example

```
HTML
<div class="card">
 <img src="..." class="card-img-top" alt="Card image cap">
 <div class="card-body">
  <h5 class="card-title">Card title</h5>
  <p class="card-text">Some quick example text to build on the card's content.</p>
  <a href="#" class="btn btn-primary">Go somewhere</a>
 </div>
</div>
```

**4. Additional Options**

- Bootstrap provides various card styles and options. Here are some examples:
  - Borders and Colors: Use Bootstrap utility classes like .border and .bg-* to add borders and backgrounds.
  - Card Size: Use classes like .card-lg or .card-sm to adjust card size.
  - Text Alignment: Use Bootstrap's text alignment classes (.text-center, .text-right) within the card body.

**5. Using Cards in Django Templates**

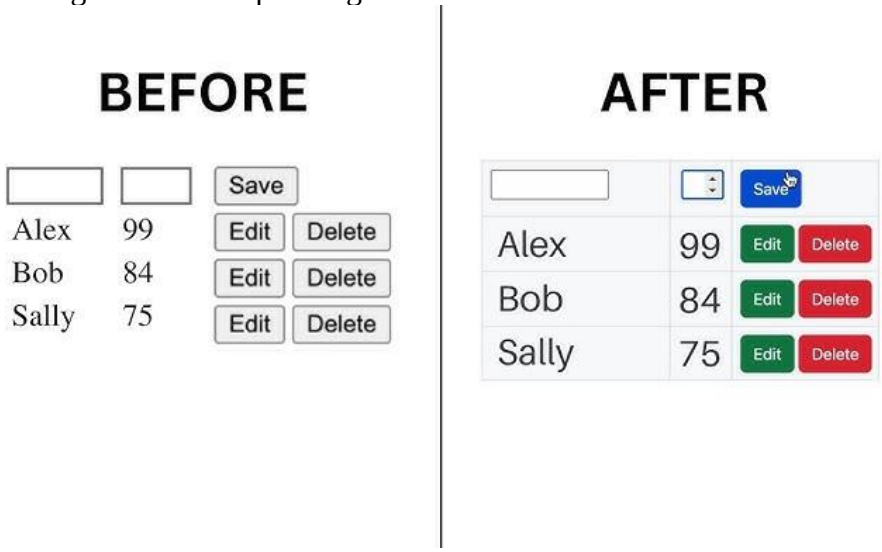
In your Django templates, you can create cards dynamically using context variables or loops:

```
HTML
{% for item in items %}
 <div class="card">
  <img src="{{ item.image_url }}" class="card-img-top" alt="{{ item.title }}">
  <div class="card-body">
   <h5 class="card-title">{{ item.title }}</h5>
   <p class="card-text">{{ item.description }}</p>
   <a href="{{ item.detail_url }}" class="btn btn-primary">View Details</a>
  </div>
 </div>
{% endfor %}
```

This example iterates through a list of items (each with properties like image_url, title, and description) and creates a card for each item.

## 2.3. Table Design

Bootstrap offers a variety of classes to enhance the look and feel of your Django tables. Here's a guide on incorporating them



**1. Prerequisites:** Ensure Bootstrap is downloaded and included in your base template (e.g., base.html).

**2. Basic Table Styling:** Wrap your HTML table element with the .table class for basic styling.

```
HTML
<table>
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Username</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>John</td>
      <td>Doe</td>
      <td>jdoe</td>
    </tr>
  </tbody>
</table>
```

**3. Additional Styling Options:**

- Bootstrap provides various table styles beyond basic formatting. Here are some popular options:
    - Stripped Rows: Add the .table-striped class for alternating row colors.
    - Bordered Tables: Include the .table-bordered class for borders around the table and cells.
    - Hoverable Rows: Use the .table-hover class to add a hover effect on rows.
    - Contextual Classes: Apply classes like .table-success or .table-danger to highlight specific rows.

```
HTML
<table class="table table-striped table-bordered">
  ... (your table content)
</table>
```

**4. Responsive Tables:**

- Bootstrap's responsive tables automatically adjust their layout for different screen sizes. Use the .table-responsive class on a parent element to enable responsiveness.

```
HTML
<div class="table-responsive">
  <table class="table">
    ... (your table content)
  </table>
</div>
```

**5. Using Tables in Django Templates:** You can populate your tables dynamically using Django's template tags and filter

```
HTML
<table>
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Username</th>
    </tr>
  </thead>
  <tbody>
    {% for user in users %}
    <tr>
      <th scope="row">{{ user.id }}</th>
      <td>{{ user.first_name }}</td>
      <td>{{ user.last_name }}</td>
      <td>{{ user.username }}</td>
    </tr>
    {% endfor %}
  </tbody>
</table>
```

This example iterates through a list of users (users) and displays their details in a table.

**Resources**

- Bootstrap Table Documentation: https://getbootstrap.com/docs/4.0/content/tables/
- Django Template Tags and Filters: https://docs.djangoproject.com/en/5.0/ref/templates/builtins/ [invalid URL removed]

By effectively utilizing Bootstrap's table classes and Django's templating capabilities, you can create well-formatted and responsive tables in your Django applications.

## 2.4. Navigation with Navbar


Bootstrap's Navbar component offers a clean and efficient way to implement navigation menus in your Django templates. Here's a step-by-step guide:

**1. Prerequisites:** Ensure Bootstrap is downloaded and included in your base template (e.g., base.html).

**2. Basic Navbar Structure:** Wrap your navigation elements within a <nav> element with the .navbar class.

```
HTML
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 </nav>
```

**3. Navbar Branding:** Use the .navbar-brand class on an anchor <a> tag to define your website's brand or logo.

```
HTML
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 <a class="navbar-brand" href="#">My Website</a>
 </nav>
```

**4. Toggling Navbar for Smaller Screens:**

- Bootstrap's responsive navigation requires a toggle button for collapsing the navbar on smaller screens.
    - Include the .navbar-toggler class on a <button> element.
    - Set data-toggle="collapse" and data-target="#navbarNavAltMarkup" attributes to link the button with the collapsible content (explained later).

```
HTML
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 <a class="navbar-brand" href="#">My Website</a>
 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
   <span class="navbar-toggler-icon"></span>
 </button>
 </nav>
```

**5. Collapsible Navigation Links:**
Create the collapsible content for your navigation links using a  element with the .collapse class and a unique ID (e.g., navbarNavAltMarkup).

Wrap the navigation links within an unordered list <ul> element with the .navbar-nav class.

```
HTML
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 <a class="navbar-brand" href="#">My Website</a>
 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
   <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
  <ul class="navbar-nav">
    <li class="nav-item">
      <a class="nav-link active" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">About</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Contact</a>
    </li>
  </ul>
 </div>
 </nav>
```

**6. Additional Considerations**

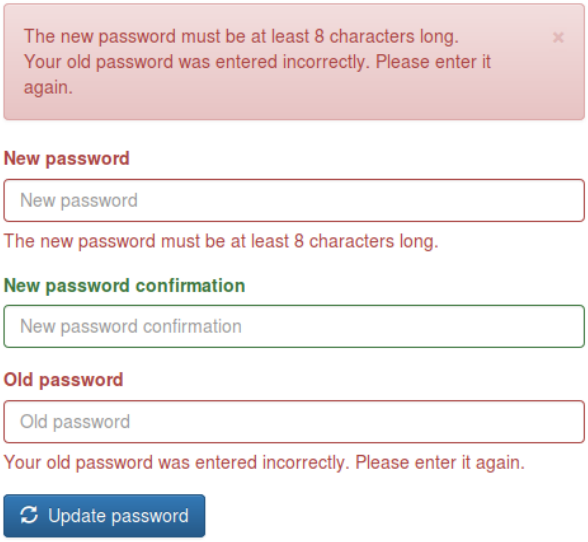
- You can style your navbar further using Bootstrap's utility classes (e.g., .bg-primary, .text-white) for backgrounds and text colors.
- Consider using Django template tags like {% url %} to dynamically generate URLs for your navigation links.

**7. Including Navbar in Django Templates:** Inherit your other templates (e.g., index.html) from your base template (base.html) to include the navbar throughout your application.

By following these steps and exploring Bootstrap's navbar options, you can create a user-friendly and responsive navigation bar for your Django web application.

## 2.5. Use Validation Alert

Bootstrap provides **validation classes** to indicate error, success, or warning messages in forms. These are commonly displayed using alert components.




**How to Apply:**

- Use .alert, .alert-success, .alert-danger, .alert-warning, etc., to display validation messages.
- **Example**:

```
html
Copy
<div class="alert alert-danger" role="alert">
  There was an error with your submission. Please try again.
</div>
```

**Key Classes:**

- .alert: Base class for alerts.
- .alert-success: For success messages.
- .alert-danger: For error messages.
- .alert-warning: For warning messages.

**Summary of Key Concepts:**

1. **Bootstrap Form Class**: Use Bootstrap's built-in classes like .form-group, .form-control to style forms.
2. **Card Layout**: Apply the .card class for flexible content display.
3. **Table Design**: Use .table, .table-striped, and other table-related classes to design structured tables.
4. **Navbar**: Implement responsive navigation with .navbar and its responsive classes.
5. **Validation Alert**: Provide feedback messages using .alert classes for validation purposes.

## Self-Check Sheet 2: Apply Bootstrap on templates

1. **Which Bootstrap class is used to add a hover effect to table rows?**
    - A. .table-striped
    - B. .table-bordered
    - C. .table-hover
    - D. .table-responsive

2. **What element within the Bootstrap navbar structure is used to define the website's brand or logo?**
    - A. <nav>
    - B. <ul>
    - C. .navbar-toggler
    - D. .navbar-brand

3. **In Django templates, how can you dynamically generate URLs for navigation links?**
    - A. Using Bootstrap utility classes
    - B. By directly including the URL path in the <a> tag's href attribute.
    - C. Using Django's template tag {% url %}
    - D. There's no way to dynamically generate URLs in Django templates.

4. **Which of the following classes is used to layout form elements horizontally in Bootstrap?**
    - A. .form-horizontal
    - B. .form-inline
    - C. .form-group
    - D. .form-check

5. **What Bootstrap component provides a versatile way to structure and style content sections in Django templates?**
    - A. Navbar
    - B. Cards
    - C. Tables
    - D. Forms

## Answer Key  Apply Bootstrap on templates

1. Which Bootstrap class is used to add a hover effect to table rows?

**Answer:** C. .table-hover

2. What element within the Bootstrap navbar structure is used to define the website's brand or logo?

**Answer:** D. .navbar-brand

3. In Django templates, how can you dynamically generate URLs for navigation links?

**Answer:** C. Using Django's template tag {% url %}

4. Which of the following classes is used to layout form elements horizontally in Bootstrap?

**Answer:** A. .form-horizontal

5. What Bootstrap component provides a versatile way to structure and style content sections in Django templates?

**Answer:** B. Cards

## **Job Sheet**-2: **Create a Navigation Bar with Bootstrap in Django Templates**

### UoC Cover

OU-ICT-WADP-05- L4-V1: Customized UI

### Working Procedure / Steps

1. **Basic Navbar Structure:**
    - In your base template (base.html), add a <nav> element with the classes navbar navbar-expand-lg navbar-light bg-light. This creates a basic navbar layout.
2. **Navbar Branding:**
    - Within the <nav>, add an anchor tag <a> with the class navbar-brand and a link to your website's homepage. This will display your website's name or logo.
3. **Toggling Navbar for Mobile Devices:**
    - Include a <button> element with the classes navbar-toggler and type="button".
    - Set the attributes data-toggle="collapse" and data-target="#your-unique-id". Replace #your-unique-id with a unique ID you'll use later. This button will toggle the navbar on smaller screens.
4. **Collapsible Navigation Links:**
    - Create a  element with the classes collapse navbar-collapse and the same unique ID you used in step 3 (e.g., id="navbarNavAltMarkup").
    - Inside the , add an unordered list <ul> with the class navbar-nav.
    - Within the <ul>, create list items <li> with the class nav-item for each navigation link.
    - Inside each <li>, add an anchor tag <a> with the class nav-link and a link to the corresponding page. You can also add the class active to the first link to highlight it as the current page.

**Specification Sheet 2: Exploring Django Forms**

**Technical Requirements**

- **Programming Language:** Python 3.7 or later
- **Framework:** Django 3.2 or later
- **HTML, CSS:** Basic understanding of HTML and CSS for structuring and styling the navbar.
- **Bootstrap (Optional):** If using Bootstrap for styling, ensure it's installed and configured.

**Tools and Equipment**

- **Computer:** A computer with sufficient processing power, RAM, and storage.
- **Text Editor/IDE:** Visual Studio Code, PyCharm, Sublime Text (or any preferred code editor).
- **Web Browser:** A modern web browser for viewing and testing your navbar.

## Learning Outcome 3: Implement Accessibility-friendly methods and techniques

| | |
|---|---|
| **Assessment Criteria** | 1. The concept of User Accessibility is introduced from UX perspective.<br>2. Best practices are introduced.<br>3. Corresponding methods and techniques are set up. |
| **Conditions and Resources** | 1. Real or simulated workplace<br>2. CBLM<br>3. Handouts<br>4. Laptop<br>5. Multimedia Projector<br>6. Paper, Pen, Pencil, Eraser<br>7. Internet facilities<br>8. White board and marker<br>9. Audio Video Device |
| **Contents** | 1. Set up of static directory<br>2. Static resources<br>3. Set up of static-based url<br>4. Use of static files on templates<br>5. Set up of accessibility options in platform<br>6. Use of visual-audio-text options in platform |
| **Activities/job/Task** | 1. Create a Responsive Image Gallery with Bootstrap in Django Templates |
| **Training Methods** | 1. Discussion<br>2. Presentation<br>3. Demonstration<br>4. Guided Practice<br>5. Individual Practice<br>6. Project Work<br>7. Problem Solving<br>8. Brainstorming |
| **Assessment Methods** | Assessment methods may include but not limited to<br>1. Written Test<br>2. Demonstration<br>3. Oral Questioning<br>4. Portfolio |

## Learning Experience 3: Implement Accessibility-friendly methods and techniques

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials 'Implement Accessibility-friendly methods and techniques' |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Implement Accessibility-friendly methods and techniques' | 2. Read Information sheet 3: Implement Accessibility-friendly methods and techniques<br>3. Answer Self-check 3: Implement Accessibility-friendly methods and techniques<br>4. Check your answer with Answer key 3: Implement Accessibility-friendly methods and techniques |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job Sheet-3: Create a Responsive Image Gallery with Bootstrap in Django Templates |

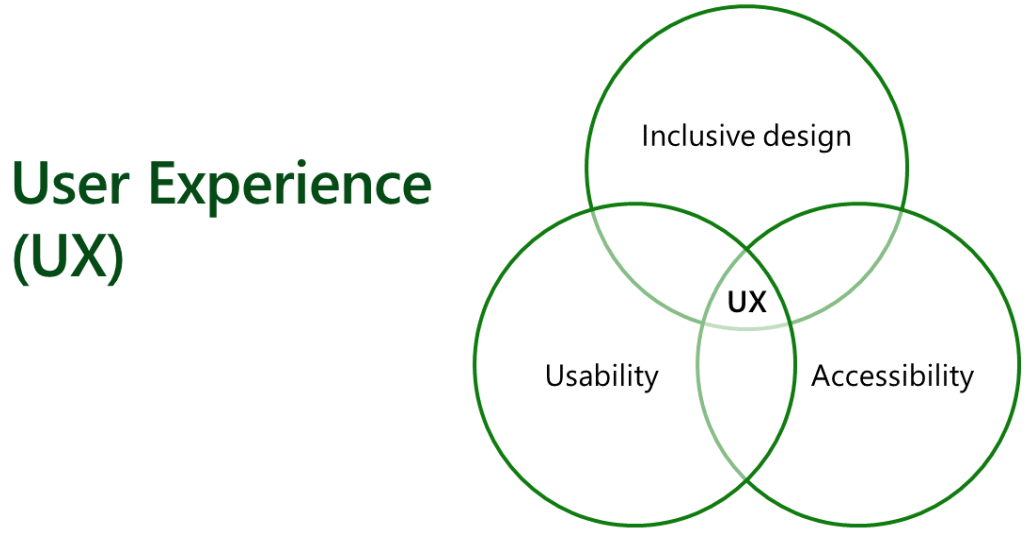
# Information sheet 3: Implement Accessibility-friendly methods and techniques

**Learning Objective:**

After completion of this Information sheet , the learners will be able to explain, define and interpret the following contents:

3.1. The concept of User Accessibility from UX perspective.
3.2. Best practices
3.3. Corresponding methods and techniques

## 3.1. The concept of User Accessibility from UX perspective.




**Principles of UX Accessibility:**

- **Perceivable:** Information and user interface (UI) components must be presented in a way that users can perceive. This includes:
    - **Visual:** Providing alternative text descriptions for images, sufficient color contrast, and proper use of color combinations for users with color blindness.
    - **Auditory:** Offering text alternatives for audio content, transcripts for videos, and captions for non-essential sounds.

    - **Touch:** Ensuring touch targets are large enough and spaced appropriately for users with motor impairments.
- **Operable:** UI components and navigation must be operable and usable. This includes:
    - **Keyboard Accessibility:** All functionalities should be accessible using a keyboard, allowing users who cannot use a mouse to interact effectively.
    - **Error Prevention:** Implementing clear instructions, error messages, and confirmation steps to guide users and prevent frustration.
    - **Focus Management:** Controlling the order in which elements receive focus helps users navigate the interface predictably.
- **Understandable:** Information and the operation of the UI must be understandable. This includes:
    - **Clear and Simple Language:** Using plain language that is easy to read and comprehend for users with varying cognitive abilities.
    - **Predictable:** The UI should behave consistently, with clear relationships between elements and actions.
    - **Instruction and Documentation:** Providing clear instructions and well-written documentation to guide users through the application's functionalities.
- **Robust:** Content must be robust and compatible with a wide range of assistive technologies. This includes:
    - **Compatibility with Screen Readers:** Ensuring screen readers can accurately interpret and convey information on the page.
    - **Assistive Technology Support:** Developing the UI with various assistive technologies in mind, such as voice control or screen magnification tools.

**Benefits of Accessible UX Design:**

- **Increased User Base:** Reach a wider audience by catering to users with disabilities.
- **Improved User Experience:** Create a more positive and inclusive experience for all users.
- **Enhanced Brand Image:** Demonstrate a commitment to social responsibility and inclusivity.
- **Reduced Development Costs:** Early consideration of accessibility can prevent costly fixes later in the development process.
- **Legal Compliance:** Accessibility regulations might be mandatory in certain regions.

**Examples of Accessibility Issues:**

- **Missing alternative text descriptions for images:** Screen reader users wouldn't know what the image represents.
- **Insufficient color contrast between text and background:** Users with low vision might struggle to read the content.
- **Navigation solely reliant on a mouse:** Users with motor impairments wouldn't be able to use keyboard navigation.
- **Complex and technical language:** Users with cognitive disabilities might find the content difficult to understand.

By understanding these principles and best practices, UX designers and developers can create user interfaces that are inclusive and usable for everyone. This leads to a more satisfying experience for all users, regardless of their abilities.

## 3.2. Best practices

Here are some key best practices you can follow when developing accessible Django applications:

**1. Semantic HTML:**

- Use HTML elements that convey meaning and structure to the content they represent.
- Examples:
    - Headings: Use <h1> to <h6> for headings in a hierarchical order.
    - Buttons: Use the <button> element for interactive buttons.
    - Lists: Use <ul> for unordered lists and <ol> for ordered lists.

**2. Alternative Text (Alt Text):**

- Provide alternative text descriptions for images using the alt attribute.
- This helps screen readers convey the image's content to users.
- Be descriptive, but concise, in your alt text descriptions.

**3. Keyboard Accessibility:**

- Ensure all interactive elements, like buttons, links, and forms, can be accessed and used with a keyboard.
- This allows users who rely on keyboards for navigation to interact with your application effectively.

**4. Color Contrast:**

- Maintain sufficient color contrast between text and background colors for readability.

- Tools like WebAIM's Contrast Checker (https://webaim.org/resources/contrastchecker/?fcolor=FFFFFF&bcolor=E9F1FC) can help you evaluate color contrast ratios.

**5. Focus Management:**

- Control the order in which interactive elements receive focus when a user tabs through the page.
- This helps users navigate the interface predictably and efficiently.

**6. ARIA Attributes:**

- Leverage WAI-ARIA (Accessible Rich Internet Applications) attributes to provide additional information about interactive elements for assistive technologies.
- ARIA attributes can clarify the functionality of complex components or dynamic content.

**7. Responsive Design:**

- Ensure your application adapts well to different screen sizes and devices.
- This allows users with varying screen resolutions and devices to access your application comfortably.

**8. Testing:**

- Regularly test your application with accessibility tools and with users with disabilities.
- Accessibility testing tools can identify potential issues, while user testing provides valuable feedback on the actual user experience.

**Additional Best Practices:**

- Use clear and simple language in your content.
- Provide clear instructions and error messages.
- Use descriptive labels for form fields.
- Ensure tables have proper headers and structure.
- Make content usable without relying solely on sound or color.

By following these best practices, you can create Django applications that are more inclusive and accessible to a wider range of users. This not only improves the user experience for everyone but also demonstrates your commitment to building a web that is usable by all.

## 3.3. Corresponding methods and techniques

Building upon the best practices discussed earlier, here are specific methods and techniques you can implement to enhance the accessibility of your Django applications:

**1. Django Packages:**
Several Django packages can simplify incorporating accessibility features into your application:

- **django-crispy-forms:** This package helps create accessible forms using Bootstrap classes. It automatically adds necessary ARIA attributes and ensures forms are keyboard-friendly.
- **django-allauth:** This package provides accessible user authentication functionalities like login, registration, and password resets. It adheres to accessibility guidelines and works well with screen readers.
- **django-accessibility:** This package offers various accessibility-related utilities. It includes tools for checking color contrast, adding keyboard navigation shortcuts, and managing focus indicators.

**2. Template Tags:**
Django provides built-in template tags that can be used to enhance accessibility:

- {% ifequal %} and {% ifnotequal %}: These tags can be used to display alternative content for screen readers. For example, you could display a longer description for complex elements only to screen readers while keeping the visual UI concise.
- {% autoescape %}: This tag helps prevent cross-site scripting (XSS) vulnerabilities, improving security for assistive technologies like screen readers. It ensures that user-generated content is properly escaped, preventing malicious code injection.

**3. Third-Party Libraries:**
Consider integrating third-party libraries to add specific accessible functionalities:

- **jQuery UI:** This library provides accessible widgets like datepickers, sliders, and accordion menus. These widgets are keyboard-friendly and work well with screen readers.
- **DAISY (Digital Accessible Information SYstem):** While not directly a library, DAISY is a format for creating accessible audio content. If your application involves audio elements, consider using DAISY to create accessible audio versions alongside your primary content.

**4. WCAG Compliance:**
Strive to comply with the Web Content Accessibility Guidelines (WCAG) set forth by the World Wide Web Consortium (W3C). WCAG provides a set of internationally

recognized recommendations for making web content accessible to people with disabilities. There are three conformance levels (A, AA, and AAA) that define the increasing level of accessibility a web page or application should achieve.
Here are some resources to get you started with WCAG compliance:

- W3C WCAG: https://www.w3.org/WAI/standards-guidelines/
- WCAG Evaluation Tools: https://www.w3.org/WAI/test-evaluate/tools/list/

**Additional Methods:**

- **Use ARIA attributes directly in your templates:** While Django packages can help, you can also manually add ARIA attributes to specific elements to provide additional information for assistive technologies.
- **Document your accessibility features:** Clearly document the accessibility features you have implemented in your application. This can be helpful for users with disabilities and can demonstrate your commitment to accessibility.
- **Stay updated:** Accessibility standards and best practices are constantly evolving. Stay up-to-date on the latest guidelines and techniques to ensure your applications remain accessible.

By implementing these methods and techniques in conjunction with the best practices outlined previously, you can create Django applications that cater to a wider audience and provide a positive user experience for everyone, regardless of their abilities.

## Self-Check Sheet 3: Implement Accessibility-friendly methods and techniques

**1. Which of the following best practices focuses on providing alternative ways for users to perceive information?**

- A. Using clear and simple language (Correct Answer)
- B. Implementing keyboard accessibility
- C. Maintaining sufficient color contrast
- D. Employing responsive design

**2. What is the primary purpose of the** django-crispy-forms **package?**

- A. To improve the visual appearance of forms.
- B. To create accessible forms using Bootstrap classes. (Correct Answer)
- C. To add user authentication functionalities.
- D. To check color contrast ratios in Django applications.

**3. According to WCAG (Web Content Accessibility Guidelines), what is the level that defines the minimum accessibility requirements?**

- A. Level 1 (A) (Correct Answer)
- B. Level 2 (AA)
- C. Level 3 (AAA)
- D. WCAG doesn't have different levels.

**4. What functionality does the** {% autoescape %} **template tag provide in Django?**

- A. To display alternative content for screen readers.
- B. To prevent cross-site scripting vulnerabilities. (Correct Answer)
- C. To improve SEO (Search Engine Optimization).
- D. To simplify the creation of semantic HTML elements.

**5. ARIA (Accessible Rich Internet Applications) attributes are used to:**

- A. Replace the need for alt text descriptions for images.
- B. Enhance the visual design of interactive elements.
- C. Provide additional information about interactive elements for assistive technologies. (Correct Answer)
- D. Reduce the overall file size of the Django application.

## Answer Key 3: Implement Accessibility-friendly methods and techniques

**1. Which of the following best practices focuses on providing alternative ways for users to perceive information?**

**Answer:** A. Using clear and simple language

**2. What is the primary purpose of the** django-crispy-forms **package?**

**Answer:** B. To create accessible forms using Bootstrap classes.

**3. According to WCAG (Web Content Accessibility Guidelines), what is the level that defines the minimum accessibility requirements?**

**Answer:** A. Level 1 (A)

**4. What functionality does the** {% autoescape %} **template tag provide in Django?**

**Answer:** B. To prevent cross-site scripting vulnerabilities.

**5. ARIA (Accessible Rich Internet Applications) attributes are used to:**

**Answer:** C. Provide additional information about interactive elements for assistive technologies.

## **Job Sheet**-3: **Create a Responsive Image Gallery with Bootstrap in Django Templates**

### **UoC Cover**

OU-ICT-WADP-05- L4-V1: Customized UI

### **Working Procedure / Steps**

1. Basic Image Gallery Structure:
    - In your template (e.g., gallery.html), create a container element with the class row.
    - Inside the .row, add multiple .col-* elements (e.g., .col-md-4) to define the number of images per row (adjust the number of columns based on your desired layout).
    - Within each column, wrap your image with a link element <a> and set the href attribute to the larger version of the image (optional for lightbox functionality).
    - Inside the <a>, place an <img> element with the desired image (src) and appropriate Bootstrap image classes (e.g., .img-fluid for responsive sizing).
2. Adding Lightbox Functionality (Optional):
    - Include a lightbox library like Lightbox2 or Magnific Popup in your project.
    - Follow the library's instructions to initialize the lightbox functionality on your image links.
3. Responsive Design:
    - Bootstrap's grid system and image classes help with responsiveness.
    - Use different column sizes (e.g., .col-sm-6 for smaller screens) within your .row to adjust the image layout on various devices.
4. Customization:
    - Explore Bootstrap's utility classes (e.g., .text-center, .border) to style your gallery further.
    - Consider using Bootstrap's carousel component for a slideshow-like image presentation.

## Specification Sheet 3: Create a Responsive Image Gallery with Bootstrap in Django Templates

**Technical Requirements**

- **Programming Language:** Python 3.7 or later
- **Framework:** Django 3.2 or later
- **HTML, CSS:** Basic understanding of HTML and CSS for structuring and styling the gallery.
- **Bootstrap:** Bootstrap 4 or later (or a compatible CSS framework) for responsive layout and styling.

**Tools and Equipment**

- **Computer:** A computer with sufficient processing power, RAM, and storage.
- **Text Editor/IDE:** Visual Studio Code, PyCharm, Sublime Text (or any preferred code editor).
- **Web Browser:** A modern web browser for viewing and testing your gallery.

**Working Procedure / Steps**

1. **Basic Image Gallery Structure:**
    - In your template (e.g., gallery.html), create a container element with the class row.
    - Inside the .row, add multiple .col-* elements (e.g., .col-md-4) to define the number of images per row (adjust the number of columns based on your desired layout).
    - Within each column, wrap your image with a link element <a> and set the href attribute to the larger version of the image (optional for lightbox functionality).
    - Inside the <a>, place an <img> element with the desired image source (src) and appropriate Bootstrap image classes (e.g., .img-fluid for responsive sizing).
2. **Adding Lightbox Functionality (Optional):**
    - Include a lightbox library like Lightbox2 or Magnific Popup in your project.
    - Follow the library's instructions to initialize the lightbox functionality on your image links.
3. **Responsive Design:**
    - Bootstrap's grid system and image classes help with responsiveness.
    - Use different column sizes (e.g., .col-sm-6 for smaller screens) within your .row to adjust the image layout on various devices.

# Reference


1. https://forum.djangoproject.com/
2. https://www.webforefront.com/django/

## Review of Competency

Below is yourself assessment rating for module “Customize UI”

| Assessment of performance Criteria | Yes | No |
|---|---|---|
| Static directory is set up | ☐ | ☐ |
| Static resources are collected and reposited. | ☐ | ☐ |
| Static-based url is set up | ☐ | ☐ |
| Statics files are used on templates | ☐ | ☐ |
| Bootstrap form class is applied | ☐ | ☐ |
| Card layout is used | ☐ | ☐ |
| Table is designed | ☐ | ☐ |
| Navigation is applied using Navbar | ☐ | ☐ |
| Validation alert is used | ☐ | ☐ |
| The concept of User Accessibility is introduced from UX perspective. | ☐ | ☐ |
| Best practices are introduced. | ☐ | ☐ |
| Corresponding methods and techniques are set up. | ☐ | ☐ |

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

# Development of CBLM

The Competency based Learning Material (CBLM) of 'Customize UI' (Occupation: Web Application Development with Python , Level-4) for National Skills Certificate is developed by NSDA with the assistance of SAMAHAR Consultants Ltd.in the month of June, 2024 under the contract number of package SD-9C dated 15th January 2024.

| SL No. | Name and Address | Designation | Contact Number |
|---|---|---|---|
| 1 | Khan Mohammad Mahmud Hasan | Writer | Cell: 01714087897<br>Email: kmmhasan@gmail.com |
| 2 | A K M Mashuqur Rahman Mazumder | Editor | Cell: 01676323576<br>Email :<br>mashuq.odelltech@odell.com.bd |
| 3 | Khan Mohammad Mahmud Hasan | Co-Ordinator | Cell: 01714087897<br>Email: kmmhasan@gmail.com |
| 4 | Md. Saif Uddin | Reviewer | Cell:01723004419<br>Email:<br>engrbd.saif@gmail.com |