Handling Forms and Creating Records

Objective: Learn how to create and handle forms in Django, including implementing a create view with form validation for the TODO App.

Creating Forms and Handling Form Submissions

Forms are an essential part of web applications, allowing users to submit data. Django provides a powerful form handling library to simplify the creation and processing of forms.

Creating Forms

You can create forms using Django's forms library. A form class defines the fields and their validation rules.

Form Example:

```python
# forms.py
from django import forms
from .models import Task

class TaskForm(forms.ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'due_date', 'status']
```

Handling Form Submissions

To handle form submissions, you need to create a view that processes the form data. If the form is valid, you save the data to the database and redirect the user.

Function-Based Create View Example:

```
# views.py
from django.shortcuts import render, redirect
from .forms import TaskForm

def create_task(request):
    if request.method == 'POST':
        form = TaskForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('task_list')
    else:
        form = TaskForm()
    return render(request, 'create_task.html', {'form': form})
```

Template Example:

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

Implementing Create View with Form Validation

Creating a view to handle form submissions includes validating the form data before saving it to the database. Django's form class automatically handles validation based on the field types and any additional validation rules you define.

Custom Validation Example:

```python
# forms.py
from django import forms
from .models import Task

class TaskForm(forms.ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'due_date', 'status']

    def clean_title(self):
        title = self.cleaned_data.get('title')
        if 'badword' in title:
            raise forms.ValidationError("Title contains inappropriate language.")
        return title
```

Validate and Process Form Data

Overview: Django forms automatically handle validation. You can access validation errors and process the data if it is valid.

Form Validation

- When a form is submitted, Django automatically checks if the data is valid based on the field types and constraints defined in the form class.
- Use `form.is_valid()` to check if the form data is valid.

Handle Valid and Invalid Data

- If the form is valid, you can access the cleaned data via `form.cleaned_data`.
- If the form is not valid, you can display validation errors in the template.

Validation Example

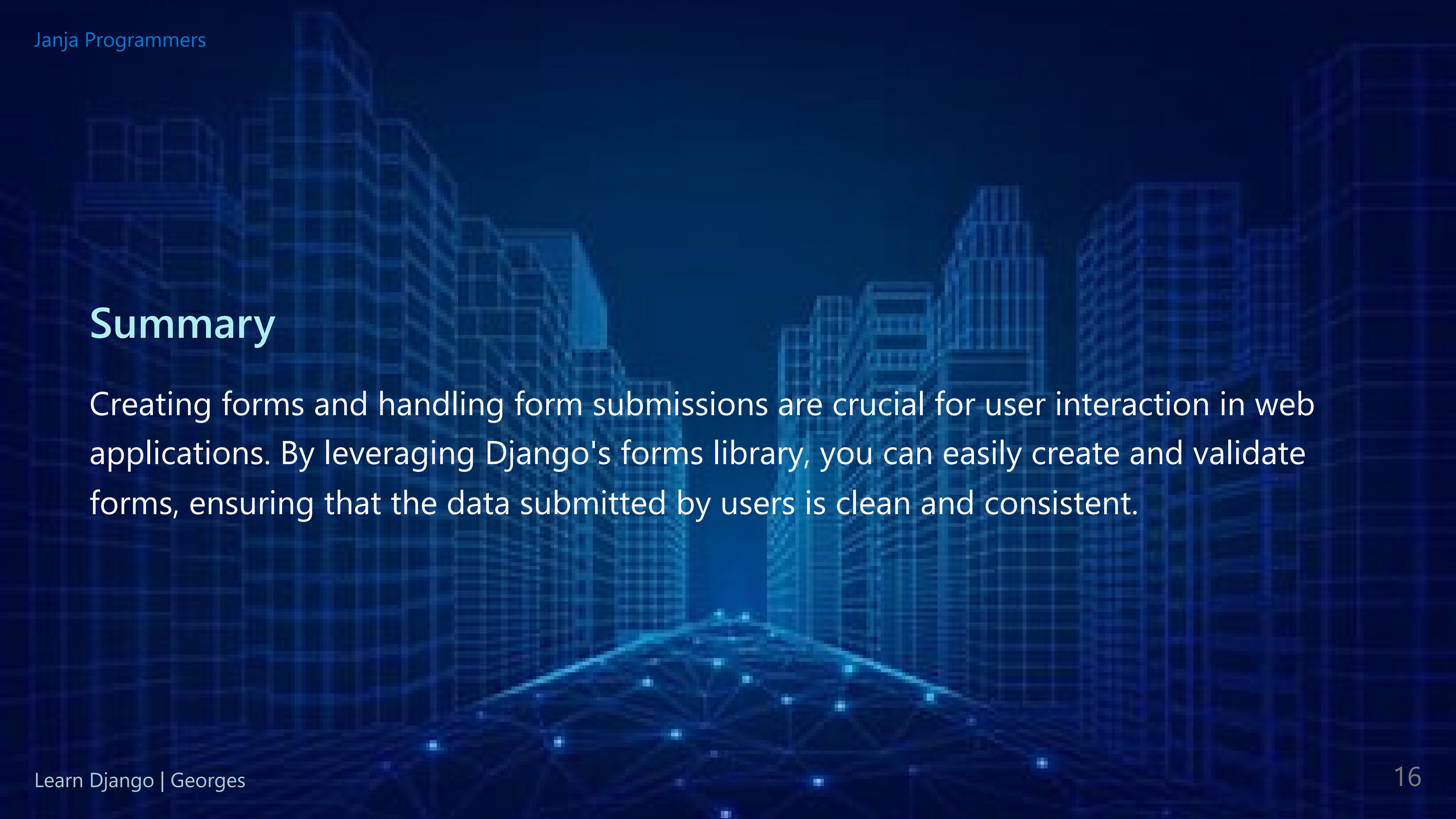
```python
# blog/views.py
from django.shortcuts import render
from .forms import PostForm

def create_post(request):
    if request.method == 'POST':
        form = PostForm(request.POST)
        if form.is_valid():
            title = form.cleaned_data['title']
            content = form.cleaned_data['content']
            author = form.cleaned_data['author']
            # Process the valid data (e.g., save it to the database)
        else:
            # Handle form errors
            pass
    else:
        form = PostForm()

    return render(request, 'blog/create_post.html', {'form': form})
```

Display Validation Errors

- Django automatically includes validation errors in the form. You can render these errors in the template using `{{ form.errors }}` or `{{ field.errors }}`.

Example Template Displaying Errors

```
<!DOCTYPE html>
<html>
  <head>
    <title>Create Post</title>
  </head>
  <body>
    <h1>Create a New Post</h1>
    <form method="post">
      {% csrf_token %} {{ form.as_p }}
      <button type="submit">Submit</button>
    </form>
    {% if form.errors %}
    <ul>
      {% for field in form %} {% for error in field.errors %}
      <li>{{ error }}</li>
      {% endfor %} {% endfor %}
    </ul>
    {% endif %}
  </body>
</html>
```



Summary

Creating forms and handling form submissions are crucial for user interaction in web applications. By leveraging Django's forms library, you can easily create and validate forms, ensuring that the data submitted by users is clean and consistent.