Django Learning

Shad Sheikh

Introduction to Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Create and run project

```
django-admin startproject project-name
cd project-name
python3 manage.py runserver
python3 manage.py startapp app-name
```

File Structure

These files are:

- The outer **`mysite/`** root directory is a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.
- **`manage.py`**: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about **`manage.py`** in django-admin and manage.py.
- The inner **`mysite/`** directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. **`mysite.urls`**).
- **`mysite/__init__.py`**: An empty file that tells Python that this directory should be considered a Python package. If you're a Python beginner, read more about packages in the official Python docs.
- **`mysite/settings.py`**: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
- **`mysite/urls.py`**: The URL declarations for this Django project; a "table of contents" of your Django-powered site. You can read more about URLs in URL dispatcher.
- **`mysite/asgi.py`**: An entry-point for ASGI-compatible web servers to serve your project. See How to deploy with ASGI for more details.
- **`mysite/wsgi.py`**: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Urls

- In Django, the URL configuration plays a fundamental role in mapping URLs to views. It determines which view function or class should handle a specific URL pattern and helps in building the overall structure of a Django application.
- URLs Configuration File: In a Django project, you typically have a file named urls.py (or sometimes multiple files) where you define the URL patterns for your application. This file acts as a central routing mechanism for incoming requests.
- URL Patterns: URL patterns are regular expressions or path patterns that define the structure of a URL and map it to a view. The URL patterns are defined using the urlpatterns list in the urls.py file. Each pattern is associated with a specific view function or class.
- Regular Expressions vs. Path Patterns: Django provides two ways to define URL patterns: regular expressions (regex) and path patterns. Regular expressions are powerful but can be complex for beginners. Path patterns offer a more readable and simplified syntax. In recent versions of Django, path patterns are the recommended approach.
- Mapping URLs to Views: To map a URL pattern to a view, you specify the pattern as the first argument in the path() function, followed by the corresponding view function or class.

Views

- In Django, a view is a Python function or a class-based method that processes an HTTP request and returns an HTTP response. It plays a crucial role in handling user requests and generating dynamic web pages.
- Request and Response: When a user makes a request to a Django application, the web server receives the request and passes it to Django. Django's URL dispatcher then maps the URL to a particular view. The view function or method processes the request and generates a response, which is sent back to the user.
- Function-based Views: The simplest type of view in Django is a function-based view. It is a Python function that takes an HttpRequest object as its first argument and returns an HttpResponse object as its response. You can define these views in your Django application's views.py file.
- Class-based Views: Django also provides class-based views, which offer a more powerful and reusable way to define views. Class-based views are defined as Python classes that inherit from Django's base view classes. These classes provide methods that handle different HTTP methods (such as GET, POST, etc.) and are responsible for generating responses.

```
28
29  def index(request):
30      list_items = ""
31      months = list(monthly_challenges.keys())
32      return render(request,"challenges/index.html",{
33          "months": months
34      })
35
36  def monthly_challenge_by_number(request, month):
37      months = list(monthly_challenges.keys())
38
39      if month > len(months):
40          return HttpResponseNotFound("Invalid month")
41
42      redirect_month = months[month - 1]
43      redirect_path = reverse("month-challenge",args=[redirect_month])
44      return HttpResponseRedirect(redirect_path)
45
46  def monthly_challenge(request, month):
47      try:
48          challenge_text = monthly_challenges[month]
49          return render(request,"challenges/challenge.html",
50                        {"text": challenge_text,"month_name":month})
51      except:
52          raise Http404()
```

Template and HTML/CSS

- In Django, templates are used to generate dynamic HTML content that is sent to the user's browser. Templates provide a way to separate the design (HTML) from the logic (Python code) in your Django application.
- Templates: Templates in Django are files containing HTML markup with embedded Django template tags. These template tags allow you to insert dynamic content, perform logic, and iterate over data in your templates. Templates can be reusable and help maintain a consistent look and feel across different pages of your application.
- HTML: HTML (Hypertext Markup Language) is the standard markup language used to structure and present content on the web. In Django templates, you write HTML code to define the structure and elements of your web pages. You can add CSS classes, inline styles, and attributes to HTML elements to control their appearance.
- CSS: CSS (Cascading Style Sheets) is a stylesheet language used to describe the visual presentation of an HTML document. CSS is responsible for styling the elements defined in the HTML markup. In Django, you can use CSS to control the appearance of your HTML templates.

index.html

Django > Learning > urls_template_views > challenges > templates > challenges > index.html

```
{% extends "base.html" %}
{% load static %}
{% block css_files %}
<link rel="stylesheet" href="{% static "challenges/challenges.css"%}">
<link rel="stylesheet" href="{% static "challenges/includes/header.css"%}">
{% endblock %}
<link>
{% block page_title%}
All Challenges
{% endblock %}

{% block content %}
{% include "challenges/includes/header.html" %}
<ul>
    {% for month in months%}
    <li><a href="{% url "month-challenge" month %}">{{month|title}}</a></li>
    {% endfor %}
</ul>
{% endblock %}
```

challenge.css

Django > Learning > urls_template_views > challenges > static > challenges > challenge.css > .fallb

```
h1,
h2 {
  text-align: center;
  color: white;
}

h1 {
  font-size: 1.5rem;
  margin: 2rem 0 1rem 0;
  font-weight: normal;
  color: #cf54a6;
}

h2 {
  font-size: 3rem;
  font-weight: bold;
}

.fallback {
  text-align: center;
  color: white;
}
```

Database Model

- Django's database models provide a foundation for storing and retrieving data. Learn hoIn Django, a database model is a Python class that represents a database table. It defines the structure and behavior of the data stored in the database. Django provides an Object-Relational Mapping (ORM) layer, which allows you to interact with the database using Python code instead of writing raw SQL queries.
- Defining a Model: To define a database model in Django, you create a subclass of the `django.db.models.Model` class. Each attribute of the model represents a database field, defining the type of data it can store. Django provides various field types such as `CharField`, `IntegerField`, `DateField`, `ForeignKey`, etc.
- Migrations: Once you've defined your models, you need to create database tables that correspond to those models. Django provides a migration system that automates the process of creating, modifying, and managing database schema changes. Migrations are Python files that contain instructions for transforming the database schema.

  To generate and apply migrations, you use the `makemigrations` and `migrate` management commands:

```
python manage.py makemigrations
python manage.py migrate
```

```python
class Country(models.Model):
    name = models.CharField(max_length=80)
    code = models.CharField(max_length=2)

    def __str__(self):
        return self.name

    class Meta:
        verbose_name_plural = "Countries"

class Address(models.Model):
    street = models.CharField(max_length=80)
    postel_code = models.CharField(max_length=5)
    city = models.CharField(max_length=50)

    def __str__(self):
        return f"{self.street}, {self.postel_code}, {self.city}"

    class Meta:
        verbose_name_plural = "Address Entries"

class Author(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    address = models.OneToOneField(Address,on_delete=models.CASCADE,null=True)

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

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

Admin

- In Django, the admin is a built-in feature that provides a web-based interface for managing and manipulating data in your Django application. It offers a convenient way to perform CRUD (Create, Read, Update, Delete) operations on your models without writing additional code.
- Registering Models: Once the admin interface is enabled, you can register your models to make them accessible through the admin interface. By registering a model, you tell Django to generate default admin views and forms for managing the model's data.To register a model, create an admin file (e.g., `admin.py`) in your app directory, import the model, and use the `admin.site.register()` method to register it:

• Admin Interface Features: The admin interface provides various features to manage your data:

a. List View: The list view displays all instances of a registered model, allowing you to browse through the existing records.

b. Detail View: The detail view shows the detailed information of a single record. It displays all the fields defined in the model.

c. Create, Update, and Delete: The admin interface provides forms for creating, updating, and deleting records. It automatically generates appropriate form fields based on the model's field types.

d. Search and Filtering: You can search for specific records using a search box and apply filters to narrow down the displayed records.

e. Sorting and Pagination: The admin interface supports sorting records by different fields and paginates the results to improve performance.

f. Customization: You can customize the admin interface by overriding templates, defining custom admin actions, adding extra fields, or modifying the display of fields.

admin.py

Django > Learning > data_model > book_outlet > admin.py > ...

```python
from django.contrib import admin

from .models import Book,Author,Address,Country

# Register your models here.

class BookAdmin(admin.ModelAdmin):
    prepopulated_fields = {"slug": ("title",)}
    list_filter = ("author", "rating",)
    list_display = ("title", "author",)

admin.site.register(Book, BookAdmin),
admin.site.register(Author),
admin.site.register(Address),
admin.site.register(Country)
```

Forms

- Forms in Django provide a convenient way to handle user input, validate data, and process form submissions in web applications. They allow you to define form fields, display them in templates, validate user input, and perform various operations on the submitted data.
- Form Basics: In Django, forms are created using Python classes that inherit from the `django.forms.Form` or `django.forms.ModelForm` class. You define form fields as attributes of the form class, specifying the field types, labels, validation rules, and other attributes.
- Field Types: Django provides a wide range of field types that you can use in your forms, such as `CharField`, `EmailField`, `IntegerField`, `DateField`, `ChoiceField`, etc. These field types determine the type of data that can be entered and perform basic validation.
- Rendering Forms: To display a form in a template, you pass an instance of the form class to the template context. Django provides template tags and filters to render individual form fields, labels, error messages, and the form as a whole.

```python
class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = "__all__"
        labels = {
            "user_name": "Your Name",
            "review_text": "Your Feedback",
            "rating": "Your Rating"
        }
        error_messages = {
            "user_name": {
              "required": "Your name must not be empty!",
              "max_length": "Please enter a shorter name!"
            }
        }
```


Thank You
Shad Sheikh