Modul 1: Einführung in Django und Webentwicklung mit Backend

Ziele

Verständnis der MVC/MVT-Architektur, Einrichtung der Arbeitsumgebung und grundlegende Django-Konzepte.

---

1. HTTP-Protokoll und Request/Response Mechanismus

Allgemein:

Das **HTTP-Protokoll** (Hypertext Transfer Protocol) ist das Fundament der Kommunikation im Web. Es basiert auf einer Anfrage-Antwort-Architektur, bei der der Client (in der Regel ein Webbrowser) eine Anfrage an den Server sendet und der Server eine Antwort zurückgibt.

- **HTTP-Request**: Der Client sendet eine Anfrage mit einer Methode wie GET, POST, PUT, oder DELETE. Diese Methode bestimmt, welche Art von Aktion der Server durchführen soll (z.B. Daten abfragen, senden, aktualisieren oder löschen).
- **HTTP-Response**: Der Server verarbeitet die Anfrage und sendet eine Antwort, die aus einem **Statuscode** (z.B. 200 OK, 404 Not Found) und oft einem **Antwortkörper** (z.B. HTML, JSON) besteht.

Request/Response in Django:

Django übernimmt die Steuerung von Anfragen und Antworten automatisch:

- Die Anfrage wird von Django verarbeitet und an eine passende View-Funktion oder -Methode weitergeleitet.
- Die View erstellt eine Antwort, typischerweise durch das Rendern eines HTML-Templates oder das Senden einer JSON-Antwort.

Django bietet somit eine abstrakte Ebene, um HTTP-Interaktionen im Hintergrund zu steuern, während Entwickler sich auf die Geschäftslogik konzentrieren können.

---

2. Installation und Einrichtung

Virtuelle Umgebung erstellen und Django installieren

1. **Virtuelle Umgebung erstellen**:

- Warum? Virtuelle Umgebungen sind wichtig, um Projekte zu isolieren. Jede Umgebung kann eigene Abhängigkeiten und Versionen von Bibliotheken haben, was Konflikte vermeidet.

```
python3 -m venv myenv
```

2. **Virtuelle Umgebung aktivieren**:
    - **Windows**: `myenv\Scripts\activate`
    - **Mac/Linux**: `source myenv/bin/activate`
3. **Django installieren**:

```
pip install django

pip install django==3.2.9

python -m django --version

pip install --upgrade django
```

Erstellen eines neuen Django-Projekts

1. **Standard-Befehl für das Erstellen eines Projekts**:

```
django-admin startproject myproject
```

2. **Mit oder ohne Punkt am Ende des Befehls**

A: Mit Punkt am Ende

Vorteile:

1. Ordnerstruktur bleibt kompakt:
- Die Projektdateien (wie manage.py, settings.py) werden direkt in das aktuelle Verzeichnis geschrieben. Dies ist nützlich, wenn du in einem leeren Ordner arbeitest und alles übersichtlich in einem Verzeichnis halten möchtest.
2. Ideal für isolierte Projekte:
- Wenn du das Projekt in einem Container, in einem virtuellen Server-Setup oder für kleinere Einzelprojekte erstellst, bleibt das gesamte Projekt in einem Verzeichnis und ist leicht verschiebbar und isoliert.
1. Weniger Pfade:
- Da alle wichtigen Projektdateien direkt verfügbar sind, gibt es weniger tiefe Verzeichnisstrukturen, was für einfache Projekte effizient ist.

Nachteile:

1. Unübersichtlich bei mehreren Apps:
- Wenn du mehrere Apps hinzufügst oder das Projekt wächst, kann das Verzeichnis schnell unübersichtlich werden, da alle Dateien im selben Verzeichnis liegen.
2. Weniger Flexibilität für zusätzliche Dateien:
- Ein Hauptordner für das Projekt bietet mehr Flexibilität für zusätzliche Konfigurationsdateien (wie Dockerfiles, CI/CD-Konfigurationen) und Ordnerstrukturen.

B: Ohne Punkt am Ende

Vorteile

1. Bessere Struktur:
- Django erstellt einen eigenen Ordner (meinprojekt), in dem alle Projektdaten organisiert liegen. Dies macht das Projekt übersichtlicher und leichter zu navigieren, besonders bei größeren Projekten mit mehreren Apps.
2. Trennung von Konfiguration und Quellcode:
- Der äußere Ordner enthält Konfigurationsdateien (manage.py, .env, Docker-Konfigurationen), während der innere Projektordner alle Django-spezifischen Einstellungen und Dateien enthält. Das trennt die Struktur sauber und hilft bei der Versionskontrolle.
3. Flexibler für größere Projekte:
- Diese Struktur eignet sich besser für skalierbare, größere Projekte, bei denen mehrere Teammitglieder arbeiten und zusätzliche Konfigurationsdateien und Verzeichnisse erforderlich sind.

Nachteile

1. Zusätzliche Ordnerstruktur:
- Es wird ein zusätzlicher Ordner erstellt, was die Navigation etwas tiefer macht. Für sehr einfache Projekte kann dies als unnötig erscheinen.
2. Verwaltung von Pfaden:
- Bei Referenzen auf Projektdateien oder bei Änderungen an der Projektstruktur können zusätzliche Pfadreferenzen notwendig sein, was die Verwaltung etwas komplexer macht.

3. **Projektstruktur verstehen**:

- myproject/: Enthält die Projektkonfigurationsdateien und stellt den Hauptordner des Projekts dar.
- manage.py: Ein Kommandozeilentool, um Django-Befehle wie das Starten des Servers und das Erstellen von Migrationsdateien auszuführen.
- myproject/settings.py: Konfigurationsdatei des Projekts (z. B. Datenbankeinstellungen, installierte Apps).
- myproject/urls.py: Enthält die URL-Routings für das gesamte Projekt.

---

3. Überblick über MVC vs. MVT-Architektur

Django verwendet das **MVT-Modell** als Abwandlung des klassischen **MVC-Modells**.

- **MVC (Model-View-Controller)**:
  - **Model**: Die Datenstruktur und -logik, die die Datenbank-Interaktionen steuert.
  - **View**: Präsentationsschicht, die die Benutzeroberfläche anzeigt und oft Logik zur Präsentation enthält.
  - **Controller**: Steuert die Verbindungen zwischen Model und View, interpretiert Benutzeraktionen und aktualisiert das Modell oder die Ansicht.
- **MVT (Model-View-Template)**:
  - **Model**: Bleibt unverändert und stellt die Datenstruktur und Logik dar.
  - **View**: In Django ist dies die Controller-Logik, die den Datenfluss und die Anwendung steuert.
  - **Template**: HTML-Dateien mit Platzhaltern für dynamische Inhalte, steuern die Darstellung für den Benutzer.

Vorteile von MVT in Django:

- Trennung der Logik: Das MVT-Pattern in Django strukturiert den Code sauberer, da Views und Templates klar getrennt sind.
- Django übernimmt automatisch die Aufgaben des Controllers, was Entwicklungszeit spart.

Anders Ausgedrückt

1. MVC vs. MVT

- MVC (Model-View-Controller): Eine weit verbreitete Architektur für die Softwareentwicklung, die die Komponenten einer Anwendung trennt.
- MVT (Model-View-Template): Django implementiert eine Variante des MVC namens MVT. Die Controller-Logik wird von Django automatisch übernommen, weshalb sie nicht explizit im Code vorhanden ist.

2. Model (Modell)

- Stellt die Datenstruktur der Anwendung dar und definiert, wie Daten gespeichert und abgerufen werden. In Django sind Model-Klassen Python-Klassen, die Tabellen in der Datenbank repräsentieren.

3. View (Sicht)

- Die Logik, die für den Benutzer angezeigt wird. Views in Django verarbeiten Anfragen und liefern eine Antwort. Die Antwort kann HTML, JSON oder eine andere Art von Antwort sein.

4. Template (Vorlage)

- HTML-Dateien, die zusammen mit Django-Template-Tags zur Darstellung von Daten genutzt werden. Templates enthalten HTML-Strukturen und spezielle Tags, um dynamische Daten aus den Views einzufügen.

---

4. Erstellen einer Django-App

1. **Was ist eine Django-App?**: Eine App in Django ist ein Teilprojekt, das eine spezifische Funktionalität erfüllt. Zum Beispiel könnte man für eine Webseite eine Blog-App, eine Benutzer-App und eine E-Commerce-App haben. Ein Django-Projekt besteht meist aus mehreren Apps.
2. **App erstellen**:

```
python manage.py startapp myapp
```

3. **App registrieren**:

- Warum? Django muss wissen, dass die App existiert und verwendet wird.
- Anleitung: Öffne myproject/settings.py und füge myapp zur Liste INSTALLED_APPS hinzu.

```
```

```
# In myproject/settings.py
INSTALLED_APPS = [
    ...,
    'myapp',
]
```

---

5. Übungen: Einrichten eines simplen Projekts und Erstellen einer Basis-App

1. **Ein einfaches Model erstellen**:

```
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

2. **Migrationen erstellen und anwenden**:

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

3. **Admin-Bereich konfigurieren**:

```
from django.contrib import admin
from .models import BlogPost

admin.site.register(BlogPost)
```

4. **View erstellen**:

```
from django.shortcuts import render
from .models import BlogPost

def blog_list(request):
    posts = BlogPost.objects.all()
    return render(request, 'blog_list.html', {'posts': posts})
```

5. **Template erstellen**:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blog List</title>
</head>
<body>
    <h1>Blog Posts</h1>
    {% for post in posts %}
        <h2>{{ post.title }}</h2>
        <p>{{ post.content }}</p>
        <small>{{ post.created_at }}</small>
    {% endfor %}
```

```
</body>
</html>
```

6. **URL-Routing hinzufügen**:

```
# In myapp/urls.py
from django.urls import path
from . import views

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

```
# In myproject/urls.py
from django.urls import path, include

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

Dieser Leitfaden bietet eine umfassende Einführung in die grundlegenden Django-Konzepte, HTTP-Grundlagen und die MVC/MVT-Architektur, die eine solide Basis für den Aufbau komplexer Anwendungen schaffen.