02 - Django: Projekt-Setup und erste URLs

Einleitung

- Dieses Kapitel konzentriert sich auf die essenziellen ersten Schritte mit Django.
- Es behandelt die Einrichtung einer isolierten Entwicklungsumgebung und die Installation von Django.
- Zentral sind die Initialisierung eines neuen Django-Projekts und das Verständnis seiner grundlegenden Dateistruktur.
- Ein weiterer Schwerpunkt liegt auf den ersten Schritten im URL-Routing, um zu verstehen, wie Django Anfragen verarbeitet.

---

1. Vorbereitung der Entwicklungsumgebung

Eine saubere und isolierte Entwicklungsumgebung ist der Schlüssel für eine effiziente und konfliktfreie Arbeit an Webprojekten.

1.1 Virtuelle Umgebungen (`venv`)

Virtuelle Umgebungen sind von entscheidender Bedeutung, um Python-Projekte voneinander zu isolieren. Jede Umgebung kann eigene Abhängigkeiten und spezifische Versionen von Bibliotheken enthalten, was Konflikte zwischen verschiedenen Projekten vermeidet. Ohne virtuelle Umgebungen könnte es passieren, dass eine Bibliothek, die für Projekt A in Version 1.0 benötigt wird, mit Projekt B in Version 2.0 kollidiert. Virtuelle Umgebungen schaffen hier klare Grenzen.

Erstellung einer virtuellen Umgebung: Der Befehl `python3 -m venv myenv` erstellt eine neue virtuelle Umgebung namens `myenv` im aktuellen Verzeichnis. Es ist bewährte Praxis, den Ordner der virtuellen Umgebung, wie z.B. `myenv`, in der `.gitignore`-Datei zu ignorieren, damit er nicht im Versionskontrollsystem landet.

Aktivierung der virtuellen Umgebung: Bevor mit der Arbeit an einem Projekt begonnen wird, muss die zugehörige virtuelle Umgebung aktiviert werden.

- **Windows:** Auf Windows findest du nach der Erstellung der virtuellen Umgebung mehrere `activate`-Dateien im `myenv\Scripts\`-Verzeichnis:
    - `activate.bat`: Dies ist die Standard-Batch-Datei zur Aktivierung in der Windows-Eingabeaufforderung (CMD).
    - `Activate.ps1`: Dies ist ein PowerShell-Skript zur Aktivierung in PowerShell.
    - `activate.fish`: Für Fish-Shell-Benutzer.
    - `activate.csh`: Für C-Shell-Benutzer.

    Best Practice auf Windows:

    - Wenn du die **Windows-Eingabeaufforderung (CMD)** verwendest, nutze: `myenv\Scripts\activate.bat`
    - Wenn du **PowerShell** verwendest, nutze: `myenv\Scripts\Activate.ps1` (Möglicherweise musst du die Ausführungsrichtlinie für Skripte einmalig anpassen, falls Fehlermeldungen auftreten, z.B. mit `Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`).

    Die gängigste und einfachste Methode ist die Verwendung von `myenv\Scripts\activate` ohne Dateiendung, da die Shell (CMD oder PowerShell) oft automatisch die richtige Datei findet und ausführt, abhängig von deiner Umgebung. **Beispiel (für CMD oder PowerShell):** `.\myenv\Scripts\activate` (Der `./` ist wichtig, wenn du dich im übergeordneten Verzeichnis befindest).

- **Mac/Linux:** `source myenv/bin/activate`

Nach der Aktivierung wird der Name der virtuellen Umgebung (z.B. `(myenv)`) oft in Klammern vor der Kommandozeile angezeigt. Alle nun installierten Python-Pakete landen ausschließlich in dieser Umgebung.

Überprüfung der Python-Installation: Es ist hilfreich zu überprüfen, ob die korrekte Python-Installation (aus der virtuellen Umgebung) genutzt wird.

- **Mac/Linux:** Du kannst den Pfad zum verwendeten Python-Interpreter mit dem Befehl `which python` oder `which python3` überprüfen. Wenn die virtuelle Umgebung aktiv ist, sollte der Pfad auf den `python`-Interpreter innerhalb deines `myenv/bin`-Verzeichnisses zeigen, z.B. `/path/to/your/project/myenv/bin/python`.

- **Windows:** Auf Windows kannst du den Befehl `where python` in CMD oder `Get-Command python` in PowerShell verwenden, um den Pfad zum verwendeten Python-Interpreter zu sehen. Wenn die virtuelle Umgebung aktiv ist, sollte der Pfad auf die

`python.exe` innerhalb deines `myenv\Scripts\`-Verzeichnisses zeigen, z.B. `C:\path\to\your\project\myenv\Scripts\python.exe`.

Diese Überprüfung hilft dir sicherzustellen, dass du wirklich im Kontext deiner virtuellen Umgebung arbeitest und nicht eine systemweite Python-Installation verwendest.

1.2 Django installieren

Sobald die virtuelle Umgebung aktiv ist, kann Django installiert werden.

- **Installation des neuesten Django-Releases:** `pip install django`
- **Installation einer spezifischen Django-Version (empfohlen für Kurse):** Um sicherzustellen, dass alle Teilnehmenden mit der gleichen Version arbeiten, ist es oft sinnvoll, eine spezifische Version zu installieren. Die Version `3.2.9` wurde beispielsweise in den alten Skripten erwähnt.

  ```
  pip install django==3.2.9
  ```

- **Überprüfung der installierten Django-Version:** `python -m django --version` Dieser Befehl zeigt die aktuell in der aktiven Umgebung installierte Django-Version an.
- **Upgrade von Django:** Wenn zu einem späteren Zeitpunkt ein Update auf eine neuere Version erforderlich ist, kann dies mit folgendem Befehl durchgeführt werden: `pip install --upgrade django`

---

2. Erstellen eines neuen Django-Projekts

Ein Django-Projekt ist die oberste Organisationsstruktur für eine Webanwendung. Es kann eine oder mehrere "Apps" enthalten, die spezifische Funktionalitäten bereitstellen.

2.1 Der `django-admin startproject` Befehl

Der Befehl `django-admin startproject` wird verwendet, um ein neues Django-Projekt zu initialisieren.

Grundlegende Verwendung: `django-admin startproject myproject` Dieser Befehl erstellt ein neues Verzeichnis namens `myproject`, das die grundlegende Projektstruktur enthält.

2.2 Projektstruktur: Mit oder ohne Punkt am Ende?

Beim Erstellen eines Projekts ist die Wahl, ob ein Punkt (`.`) am Ende des `startproject`-Befehls verwendet wird, eine wichtige Entscheidung, die die Verzeichnisstruktur beeinflusst.

2.2.1 Ohne Punkt am Ende (`django-admin startproject myproject`)

Struktur: Django erstellt einen äußeren Ordner (`myproject`), der den inneren Projektordner (`myproject/myproject`) enthält.

```
myproject/
├── manage.py
└── myproject/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Vorteile:

- **Bessere Struktur:** Diese Methode erzeugt einen separaten Ordner (meinprojekt), in dem alle Projektdaten organisiert sind. Dies macht das Projekt übersichtlicher und leichter zu navigieren, besonders bei größeren Projekten mit mehreren Apps.
- **Trennung von Konfiguration und Quellcode:** Der äußere Ordner kann Konfigurationsdateien wie `manage.py`, virtuelle Umgebungen (`env`) oder Docker-Konfigurationen enthalten. Der innere Projektordner enthält alle Django-spezifischen Einstellungen und Dateien, was eine saubere Trennung und Versionskontrolle erleichtert.

- **Flexibler für größere Projekte:** Diese Struktur ist besser für skalierbare, größere Projekte geeignet, in denen mehrere Teammitglieder arbeiten und zusätzliche Konfigurationsdateien und Verzeichnisse erforderlich sind.

Nachteile:

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

2.2.2 Mit Punkt am Ende (`django-admin startproject myproject .`)

Struktur: Die Projektdateien (`manage.py`, `settings.py` etc.) werden direkt in das aktuelle Verzeichnis geschrieben. Es wird kein zusätzlicher äußerer Ordner erstellt.

```
. (aktuelles Verzeichnis)
├── manage.py
└── myproject/ (innerer Projektordner mit Einstellungen)
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Vorteile:

- **Kompakte Ordnerstruktur:** Die Projektdateien werden direkt in das aktuelle Verzeichnis geschrieben. Dies ist nützlich, wenn in einem leeren Ordner gearbeitet wird und alles übersichtlich in einem Verzeichnis gehalten werden soll.
- **Ideal für isolierte Projekte:** Wenn das Projekt in einem Container, in einem virtuellen Server-Setup oder für kleinere Einzelprojekte erstellt wird, bleibt das gesamte Projekt in einem Verzeichnis und ist leicht verschiebbar und isoliert.
- **Weniger Pfade:** Da alle wichtigen Projektdateien direkt verfügbar sind, gibt es weniger tiefe Verzeichnisstrukturen, was für einfache Projekte effizient sein kann.

Nachteile:

- **Unübersichtlich bei mehreren Apps:** Bei der Hinzufügung mehrerer Apps oder wenn das Projekt wächst, kann das Verzeichnis schnell unübersichtlich werden, da alle Dateien im selben Verzeichnis liegen.
- **Weniger Flexibilität für zusätzliche Dateien:** Ein Hauptordner für das Projekt bietet weniger Flexibilität für zusätzliche Konfigurationsdateien (wie Dockerfiles, CI/CD-Konfigurationen) und Ordnerstrukturen.

Empfehlung für diesen Kurs: Für größere, besser strukturierte Projekte, wie sie im professionellen Umfeld üblich sind und die in diesem Kurs erstellt werden, ist die Variante **ohne Punkt am Ende** (`django-admin startproject myproject`) die bevorzugte Wahl. Sie schafft eine klarere Trennung und Skalierbarkeit.

2.3 Die grundlegende Projektstruktur verstehen

Nach der Erstellung eines Django-Projekts wird folgende Basisstruktur generiert:

- `myproject/` **(Äußerer Ordner):** Dies ist der Hauptordner des Projekts.
  - `manage.py`**:** Ein Kommandozeilentool, das für die Ausführung verschiedener Django-Befehle verwendet wird, wie das Starten des Entwicklungsservers oder das Erstellen von Migrationsdateien.
- `myproject/` **(Innerer Projektordner):** Dieser Ordner enthält die Python-Pakete des Projekts.
  - `myproject/__init__.py`**:** Eine leere Datei, die Python anzeigt, dass dieses Verzeichnis ein Python-Paket ist.
  - `myproject/settings.py`**:** Die zentrale Konfigurationsdatei des Projekts, in der Datenbankeinstellungen, installierte Apps und viele weitere globale Einstellungen definiert werden.
  - `myproject/urls.py`**:** Enthält die URL-Routings für das gesamte Projekt. Dies ist die erste Stelle, an der Django nach passenden URLs für eingehende Anfragen sucht.
  - `myproject/wsgi.py`**:** Eine Datei, die für die Bereitstellung des Projekts auf einem WSGI-kompatiblen Webserver (wie Gunicorn) dient.
  - `myproject/asgi.py`**:** Eine Datei, die für die Bereitstellung des Projekts auf einem ASGI-kompatiblen Webserver (für asynchrone Anwendungen) dient.

---

3. Erstellen einer Django-App

Ein großes Django-Projekt wird typischerweise in kleinere, modulare Einheiten unterteilt, die als **Apps** bezeichnet werden. Jede App ist für eine spezifische Funktionalität innerhalb des Projekts zuständig.

3.1 Was ist eine Django-App?

Eine App in Django ist ein Teilprojekt, das eine spezifische Funktionalität erfüllt. Zum Beispiel könnte eine Webseite eine Blog-App, eine Benutzer-App (`accounts`) oder eine E-Commerce-App haben. Ein Django-Projekt besteht meist aus mehreren solcher Apps. Diese Modularisierung fördert die Wiederverwendbarkeit und die klare Trennung von Verantwortlichkeiten.

3.2 App erstellen

Der Befehl `python manage.py startapp` wird verwendet, um eine neue App zu erstellen:

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

Dieser Befehl erstellt ein neues Verzeichnis namens `myapp` im Hauptprojektordner, das eine grundlegende Struktur für die App enthält (z.B. `models.py`, `views.py`, `admin.py`, `apps.py`, `tests.py`, `urls.py`).

3.3 App registrieren

Nach der Erstellung einer App muss diese im Django-Projekt registriert werden.

Warum registrieren? Django muss wissen, dass die neu erstellte App existiert und im Projekt verwendet werden soll. Ohne Registrierung wird Django die App nicht erkennen und ihre Funktionalität nicht laden.

Anleitung: Öffnen Sie die Datei `myproject/settings.py` (im inneren Projektordner) und fügen Sie den Namen Ihrer App zur Liste `INSTALLED_APPS` hinzu.

```
# In myproject/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',  # Die neu erstellte App hier hinzufügen
]
```

Es ist bewährte Praxis, eigene Apps nach den Django-Standard-Apps am Ende der Liste zu platzieren.

---

4. Erster View und URL-Routing

Nachdem das Projekt und die erste App erstellt und registriert wurden, ist der nächste Schritt, eine einfache Webanfrage zu verarbeiten. Hier kommen Views und URL-Routing ins Spiel.

4.1 Views in Django

Eine **View-Funktion** ist eine Python-Funktion, die eine Webanfrage entgegennimmt und eine Webantwort zurückgibt. Die Antwort kann HTML-Inhalt, eine HTTP-Weiterleitung, ein 404-Fehler oder ein XML-Dokument sein.

Erstellung einer einfachen View: Öffnen Sie die Datei `myapp/views.py` und fügen Sie eine einfache Funktion hinzu:

```
# In myapp/views.py
```

```python
from django.http import HttpResponse

def hello_world(request):
    """
    Eine einfache View-Funktion, die 'Hello, World!' zurückgibt.
    """
    return HttpResponse("Hello, World!")
```

- `from django.http import HttpResponse`: Importiert die `HttpResponse`-Klasse, die verwendet wird, um den Inhalt der Webantwort zu erstellen.
- `def hello_world(request):`: Definiert eine Funktion namens `hello_world`. Jede View-Funktion muss mindestens ein Argument entgegennehmen, üblicherweise `request`, das ein `HttpRequest`-Objekt darstellt.
- `return HttpResponse("Hello, World!")`: Diese Zeile erstellt eine HTTP-Antwort mit dem Text "Hello, World!" und sendet diese an den Client zurück.

4.2 URL-Routing in Django

URL-Routing ist der Prozess, bei dem eine eingehende URL einer bestimmten View-Funktion zugeordnet wird. In Django wird dies über die `urls.py`-Dateien konfiguriert.

4.2.1 App-spezifische URLs (`myapp/urls.py`)

Es ist eine Best Practice, für jede App eine eigene `urls.py`-Datei zu erstellen, um die URL-Konfiguration modular zu halten.

Erstellung von `myapp/urls.py`: Erstellen Sie eine neue Datei namens `urls.py` im Verzeichnis Ihrer App (`myapp/urls.py`) und fügen Sie folgenden Inhalt hinzu:

```python
# In myapp/urls.py

from django.urls import path
from . import views # Importiert die views.py der aktuellen App

urlpatterns = [
    path('hello/', views.hello_world, name='hello'), # Definiert eine URL-Route
]
```

- `from django.urls import path`: Importiert die `path`-Funktion, die für die Definition von URL-Patterns verwendet wird.
- `from . import views`: Importiert das `views`-Modul der aktuellen App. Der Punkt `.` bedeutet "aus dem aktuellen Paket".
- `urlpatterns = [...]`: Eine Liste von URL-Patterns. Django durchsucht diese Liste von oben nach unten, um eine passende URL zu finden.
- `path('hello/', views.hello_world, name='hello')`: Dies definiert ein einzelnes URL-Pattern:
  - `'hello/'`: Der URL-Pfad, der mit dieser Route übereinstimmen soll.
  - `views.hello_world`: Die View-Funktion, die aufgerufen wird, wenn dieser Pfad übereinstimmt.
  - `name='hello'`: Ein optionaler, aber sehr empfehlenswerter Name für diese URL-Route. Dieser Name ermöglicht es, auf die URL in anderen Teilen des Projekts (z.B. in Templates oder Python-Code) zu verweisen, ohne den harten Pfad kodieren zu müssen. Dies macht das Projekt flexibler, da URL-Pfade geändert werden können, ohne alle Verweise aktualisieren zu müssen.

4.2.2 Projektweite URLs (`myproject/urls.py`)

Um die URLs aus Ihrer App (`myapp/urls.py`) im gesamten Projekt zugänglich zu machen, müssen sie in der Haupt-URL-Konfiguration des Projekts (`myproject/urls.py`) "eingebunden" werden.

Anpassung von `myproject/urls.py`: Öffnen Sie die Datei `myproject/urls.py` (im inneren Projektordner) und passen Sie sie wie folgt an:

```python
# In myproject/urls.py

from django.contrib import admin
```

```python
from django.urls import path, include # 'include' muss importiert werden

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app/', include('myapp.urls')), # Bindet die URLs aus myapp ein
]
```

- `from django.urls import path, include`: Die `include`-Funktion, die das Einbinden von URL-Patterns aus anderen `urls.py`-Dateien ermöglicht, wird importiert.
- `path('app/', include('myapp.urls'))`: Dieses Pattern besagt, dass, wenn eine URL mit `'app/'` beginnt, Django den Rest der URL an die `urls.py`-Datei der `myapp` App delegieren soll.
    - Wenn ein Nutzer also `http://127.0.0.1:8000/app/hello/` aufruft, wird Django zuerst das `path('app/', ...)` finden, dann den Rest der URL (`hello/`) an `myapp.urls` weiterleiten. In `myapp.urls` wird dann `path('hello/', views.hello_world, ...)` übereinstimmen und die `hello_world`-View aufgerufen.

---

5. Starten des Entwicklungsservers

Nach der Konfiguration von Projekt, App, View und URLs kann der Django-Entwicklungsserver gestartet werden, um die Anwendung im Browser zu testen.

Befehl zum Starten des Servers: Navigieren Sie im Terminal in das äußere Projektverzeichnis (dort, wo sich die `manage.py`-Datei befindet) und führen Sie den Befehl aus:

```
python manage.py runserver
```

Der Server startet üblicherweise auf `http://127.0.0.1:8000/`.

Testen im Browser: Öffnen Sie nun Ihren Webbrowser und navigieren Sie zu `http://127.0.0.1:8000/app/hello/`. Der Text "Hello, World!" sollte sichtbar sein.

Fazit

- Die Grundlage für die Arbeit mit Django wurde gelegt.
- Die Einrichtung einer isolierten Entwicklungsumgebung und die Installation von Django wurden behandelt.
- Das Verständnis der Projekt- und App-Struktur ist entscheidend für modularen Code.
- Erste Schritte im URL-Routing und die Erstellung einer einfachen View-Funktion demonstrieren den grundlegenden Request-Response-Zyklus in Django.

Cheat Sheet: Skript 02 - Django Projekt-Setup und erste URLs

Virtuelle Umgebungen

- **Erstellen:** `python3 -m venv myenv`
- **Aktivieren (Windows):** `myenv\Scripts\activate`
- **Aktivieren (Mac/Linux):** `source myenv/bin/activate`

Django Installation

- **Installieren:** `pip install django`
- **Spezifische Version:** `pip install django==X.Y.Z`
- **Version prüfen:** `python -m django --version`

Django Projekt & App

- **Projekt erstellen (empfohlen):** `django-admin startproject myproject` (ohne Punkt)
- **App erstellen:** `python manage.py startapp myapp`
- **App registrieren:** In `myproject/settings.py` die App zu `INSTALLED_APPS` hinzufügen:

```
INSTALLED_APPS = [
    # ...
    'myapp',
]
```

Django Projektstruktur (Kernkomponenten)

- `manage.py`: Kommandozeilentool für Django-Befehle.
- `myproject/settings.py`: Globale Projektkonfiguration.
- `myproject/urls.py`: Projektweites URL-Routing.
- `myapp/views.py`: Enthält View-Funktionen.
- `myapp/urls.py`: App-spezifisches URL-Routing.

Django Views

- **Einfache View:**

```
# myapp/views.py
from django.http import HttpResponse

def my_view(request):
    return HttpResponse("Inhalt der Antwort")
```

Django URL-Routing

- **App-URLs (`myapp/urls.py`):**

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

urlpatterns = [
    path('meine-route/', views.my_view, name='meine_route'),
]
```

- **Projekt-URLs (`myproject/urls.py`):**

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('praefix/', include('myapp.urls')), # App-URLs einbinden
]
```

Django Entwicklungsserver

- **Starten:** `python manage.py runserver`
- **Standard-Adresse:** `http://127.0.0.1:8000/`

Übungsaufgaben zu Skript 02

Ziel der Übungen: Praktische Anwendung der Schritte zum Django-Projekt-Setup und zum ersten URL-Routing.

1. **Einrichtung einer isolierten Entwicklungsumgebung und Django-Installation:**
    - **Aufgabe:**
        1. In einem neuen, leeren Verzeichnis eine virtuelle Umgebung namens `webdev_env` erstellen.
        2. Diese virtuelle Umgebung aktivieren.
        3. Django in dieser Umgebung installieren.
        4. Überprüfen, ob Django korrekt installiert wurde und welche Version aktiv ist.
        5. Die virtuelle Umgebung deaktivieren und erneut aktivieren, um den Prozess zu festigen.
2. **Initialisierung eines Django-Projekts und einer App:**
    - **Aufgabe:**
        1. In der aktiven virtuellen Umgebung ein neues Django-Projekt namens `kurs_projekt` erstellen (Variante ohne Punkt am Ende des Befehls verwenden).
        2. In das neu erstellte äußere Projektverzeichnis (`kurs_projekt`) navigieren.
        3. Innerhalb dieses Projekts eine neue Django-App namens `homepage` erstellen.
        4. Die `homepage`-App in der `settings.py` des `kurs_projekt` registrieren.
3. **Erster "Hello, World!" View und URL-Routing:**
    - **Aufgabe:**
        1. Die `views.py` der `homepage`-App öffnen.
        2. Eine View-Funktion namens `startseite` hinzufügen, die den Text "Willkommen auf der Startseite!" als `HttpResponse` zurückgibt.
        3. In der `homepage`-App eine `urls.py`-Datei erstellen (falls noch nicht vorhanden).
        4. Ein URL-Pattern in `homepage/urls.py` hinzufügen, das den Pfad `'start/'` der `startseite`-View-Funktion zuordnet und ihr den Namen `'startseite'` gibt.
        5. Die `homepage.urls` in die Haupt-`urls.py` des `kurs_projekt` einbinden. Das Präfix `'web/'` verwenden, sodass die finale URL `http://127.0.0.1:8000/web/start/` lautet.
        6. Den Django-Entwicklungsserver starten und die erstellte URL im Browser aufrufen, um die Funktionalität zu testen.
4. **Fehleranalyse: Nicht registrierte App:**
    - **Aufgabe:** In der `settings.py` die Zeile, die die `homepage`-App in `INSTALLED_APPS` registriert, auskommentieren.
    - Versuchen, den Entwicklungsserver erneut zu starten und die URL `http://127.0.0.1:8000/web/start/` aufzurufen.
    - Die erhaltene Fehlermeldung im Browser oder im Terminal notieren. Kurz erklären, warum dieser Fehler auftritt.
    - Den Fehler beheben, indem die Registrierung der App wiederhergestellt wird.

🚀 Schüler-Projekt: Aufbau der "Community Recipe Sharing Platform"

Konzept für den heutigen Tag: Heute beginnen die praktischen Arbeiten am Hauptprojekt für die Schüler, der **"Community Recipe Sharing Platform"**. Die in diesem Skript erlernten Grundlagen des Django-Projekt-Setups und der ersten URL-Definitionen werden direkt angewendet. Das Ziel ist es, die grundlegende Struktur der Plattform zu legen und einen ersten Anlaufpunkt im Web zu schaffen.

Aufgabe:

1. **Neues Django-Projekt initialisieren:**
    - In einem neuen, leeren Verzeichnis (außerhalb anderer Übungsprojekte) ein frisches Django-Projekt erstellen. Benennen Sie es **`recipe_platform`**. Achten Sie darauf, die empfohlene Projektstruktur (ohne Punkt am Ende des `startproject`-Befehls) zu verwenden.
    - In das neu erstellte Hauptverzeichnis (`recipe_platform`) navigieren.
2. **Erste Django-App erstellen:**
    - Innerhalb des `recipe_platform`-Projekts eine neue Django-App erstellen. Nennen Sie diese App **`recipes`**. Diese App wird in Zukunft alle Funktionen rund um die Rezeptverwaltung enthalten.
3. **App registrieren:**
    - Öffnen Sie die `settings.py` im inneren `recipe_platform`-Ordner (z.B. `recipe_platform/recipe_platform/settings.py`).
    - Fügen Sie den Namen Ihrer neuen App (`'recipes'`) zur `INSTALLED_APPS`-Liste hinzu.

4. **Erster "Willkommen"-View und URL-Routing:**
    - Öffnen Sie die `views.py` der `recipes`-App (z.B. `recipe_platform/recipes/views.py`).
    - Fügen Sie eine einfache View-Funktion namens `welcome_recipes` hinzu. Diese Funktion soll lediglich den Text "Willkommen auf der Rezept-Plattform!" als `HttpResponse` zurückgeben.
    - Erstellen Sie eine `urls.py`-Datei im `recipes`-App-Verzeichnis (falls noch nicht vorhanden: `recipe_platform/recipes/urls.py`).
    - In dieser `recipes/urls.py`-Datei ein URL-Pattern definieren, das den leeren Pfad `''` (oder einen anderen intuitiven Pfad wie `'start/'`) Ihrer `welcome_recipes`-View-Funktion zuordnet. Geben Sie dieser Route den Namen `'home'`.
    - Öffnen Sie die Haupt-`urls.py` Ihres Projekts (`recipe_platform/recipe_platform/urls.py`).
    - Binden Sie die `recipes.urls` hier ein. Verwenden Sie dabei ein klares Präfix, zum Beispiel `'recipes/'`, sodass die finale URL, um Ihre Willkommensnachricht zu sehen, `http://127.0.0.1:8000/recipes/` (oder `http://127.0.0.1:8000/recipes/start/`, falls Sie `'start/'` in der App-URL verwendet haben) lautet.
5. **Funktionalität überprüfen:**
    - Den Django-Entwicklungsserver von Ihrem äußeren `recipe_platform`-Verzeichnis aus starten: `python manage.py runserver`.
    - Öffnen Sie Ihren Webbrowser und navigieren Sie zur erstellten URL.
    - Stellen Sie sicher, dass der Text "Willkommen auf der Rezept-Plattform!" korrekt angezeigt wird.
    - Beheben Sie eventuelle Fehler, bevor Sie fortfahren.

Ziele für das Schüler-Projekt heute: Eine funktionierende Basis-Struktur des Django-Projekts mit einer ersten App und einem erreichbaren View schaffen. Dies ist der erste Schritt zum Aufbau der Community Recipe Sharing Platform.