LearnDjango

Flask vs. Django im Jahr 2025: Ein umfassender Vergleich von Python-Webframeworks

Aktualisiert11. September 2025

▶ Inhaltsverzeichnis

Flask und Django sind die beiden führenden Python-Webframeworks. Beide ermöglichen es Entwicklern, schnell Websites zu erstellen, verfolgen dabei aber grundverschiedene Ansätze. In diesem Artikel betrachten wir die Funktionen und die Funktionsweise der einzelnen Frameworks und erläutern, warum Entwickler sich für das eine oder das andere entscheiden. Um diese Unterschiede zu verdeutlichen, entwickeln wir drei verschiedene Projekte von Grund auf – eine Hello-World-App, eine persönliche Website und eine Aufgabenliste –, damit Sie sich selbst ein Bild von der Funktionsweise machen und die beste Entscheidung für Ihre Bedürfnisse treffen können.

Was ist ein Web-Framework?

Datenbankbasierte Websites haben bemerkenswert ähnliche Anforderungen: URL-Routing, Logik, Datenbankanbindung, Darstellung von HTML-Vorlagen, Benutzerauthentifizierung usw. In den Anfängen des World Wide Web mussten Entwickler all diese Komponenten selbst implementieren, bevor sie überhaupt mit der eigentlichen Website-Entwicklung beginnen konnten.

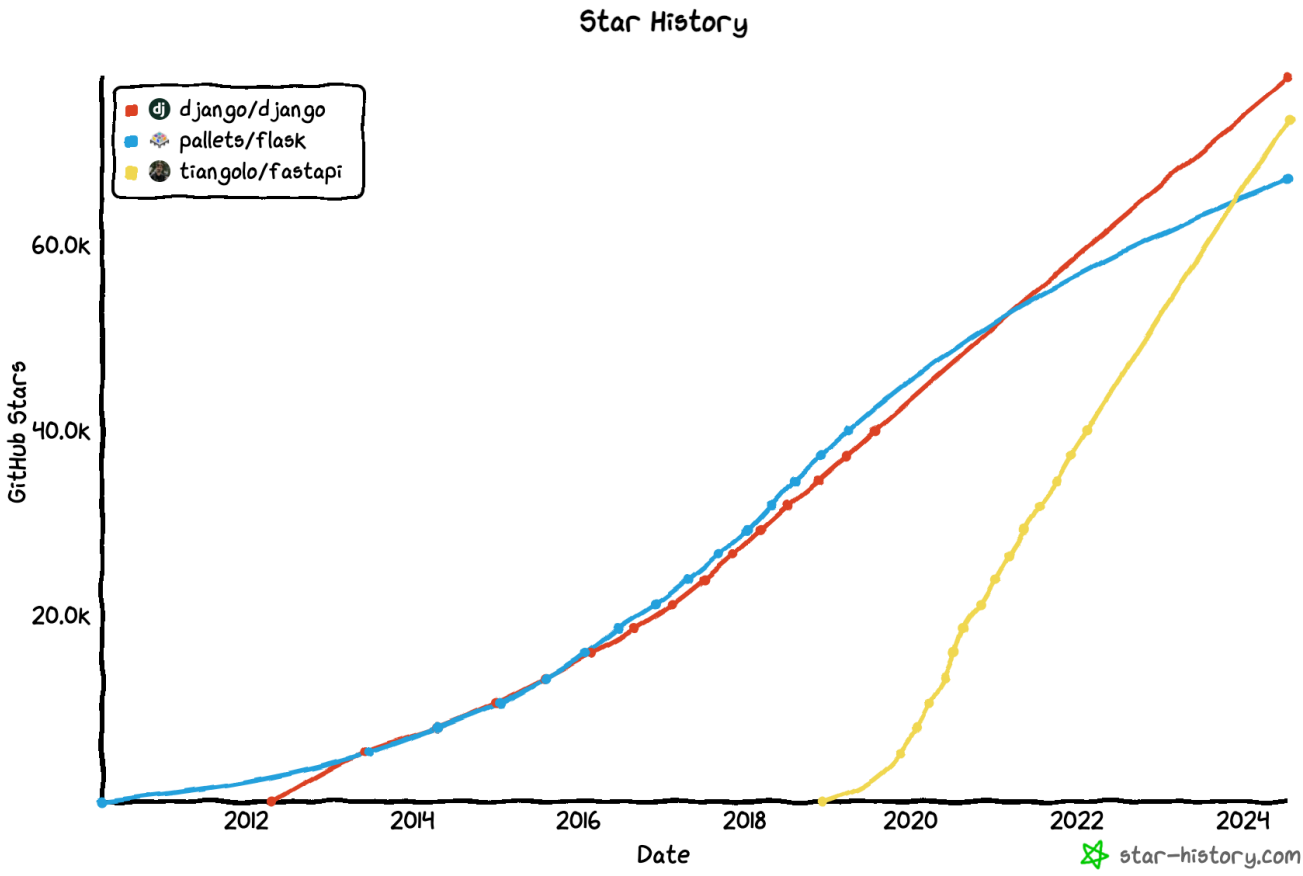
Bald entstanden Open-Source-Webframeworks, die es Entwicklergruppen ermöglichten, gemeinsam an dieser Herausforderung zu arbeiten, Best Practices auszutauschen, Code zu überprüfen und generell das Rad nicht jedes Mal neu erfinden zu müssen, wenn jemand eine neue

Website erstellen wollte. Es gibt Webframeworks für jede gängige Programmiersprache. Bekannte Beispiele sind Ruby on Rails (in Ruby geschrieben), Laravel (in PHP geschrieben), Spring (in Java geschrieben) und die beiden hier vorgestellten Python-Frameworks Flask und Django.

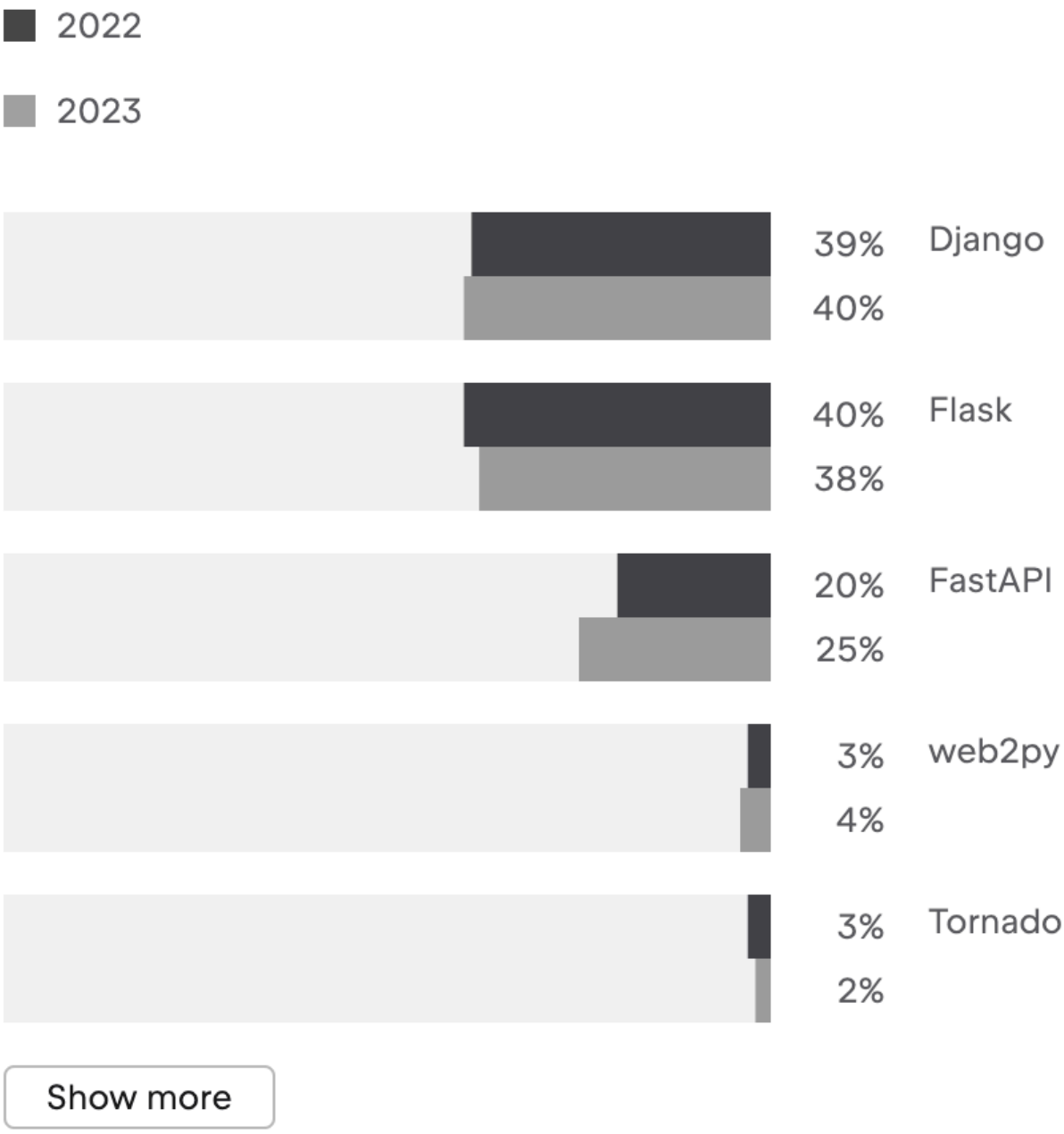
Wenn Sie noch nicht viel Programmiererfahrung haben, mag der Unterschied zwischen Python-Framework und -Bibliothek zunächst verwirrend sein. Beide bezeichnen Software, unterscheiden sich aber in ihrer Komplexität: Eine Bibliothek konzentriert sich auf ein spezifisches Problem, während ein Framework eine größere Herausforderung angeht und dafür oft viele kleinere Bibliotheken einbindet. Wie wir sehen werden, verwenden sowohl Flask als auch Django zahlreiche Python-Bibliotheken.

Welche ist beliebter?

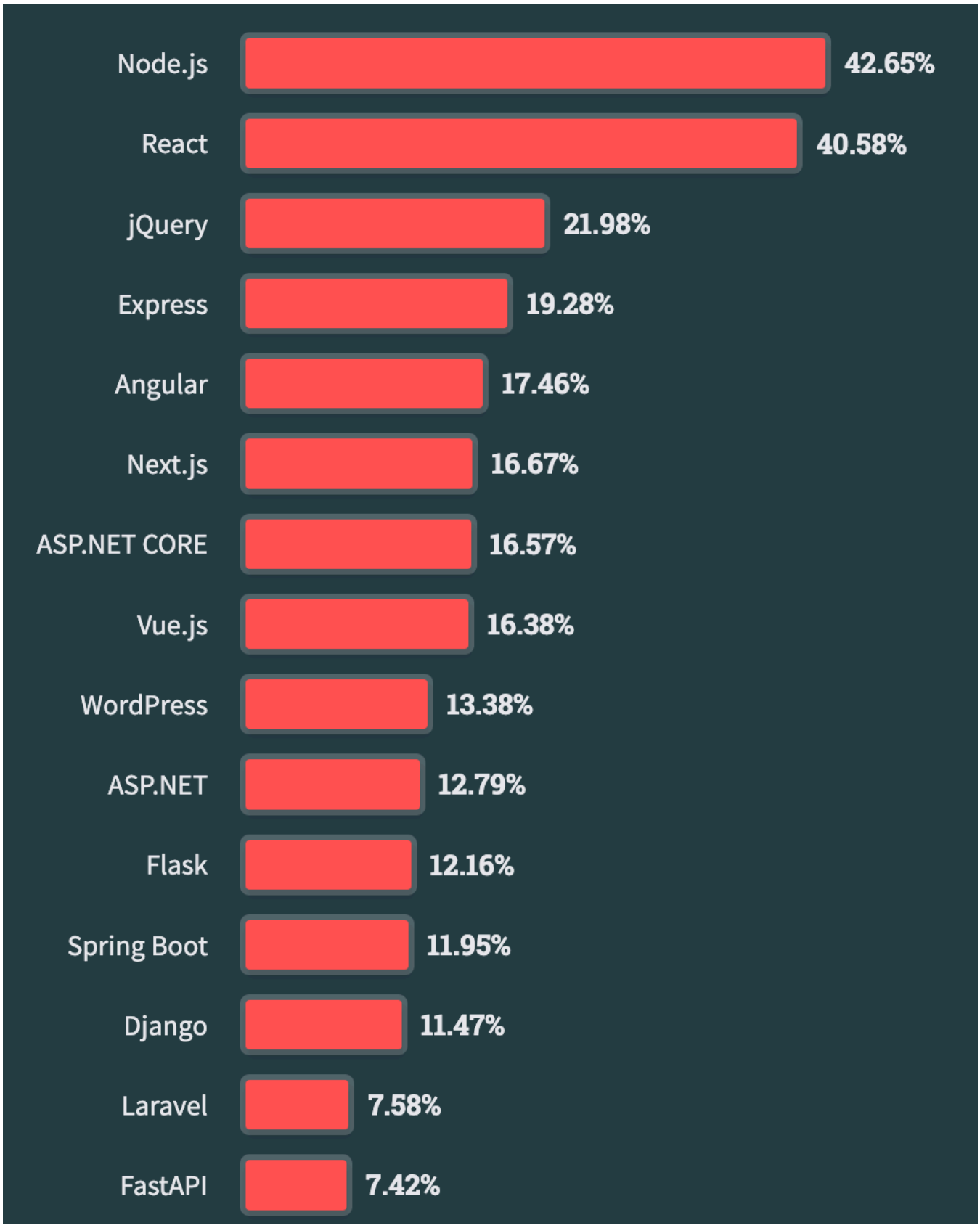
Wenn wir uns die GitHub-Sterne ansehen , liegen Flask und Django relativ gleichauf, aber wir können das explosive Wachstum von FastAPI erkennen , das sich mittlerweile klar unter den Top 3 der Python-Webframeworks etabliert hat.



Laut der Python Developers Survey 2023 liegt Django im Jahr 2023 knapp vor Flask, wobei FastAPI ebenfalls an Bedeutung gewinnt.



Laut einer Stack Overflow-Umfrage aus dem Jahr 2023 unter Entwicklern aller Programmiersprachen liegt Flask leicht vorn, wird aber fast unmittelbar von Django gefolgt, während FastAPI etwas zurückliegt.



Diese Vergleiche sind zwar interessant, um Trends zu erkennen, berücksichtigen aber viele Aspekte nicht. Nur weil ein Web-Framework populär ist, heißt das beispielsweise nicht, dass es auch von etablierten Unternehmen und professionellen Entwicklern eingesetzt wird, oder ist es nur etwas, mit dem Anfänger gerne experimentieren?

Unabhängig vom Vergleichsmaßstab ist klar, dass Flask und Django derzeit die beiden führenden Python-Webframeworks sind.

Jobs

Wenn Sie einen Job als Python-Webentwickler suchen, ist Django die bessere Wahl. Auf großen Jobportalen wie Indeed.com gibt es fast doppelt so viele Stellenangebote für Django-Entwickler wie für Flask- Entwickler.

Diese Diskrepanz dürfte jedoch darauf zurückzuführen sein, dass Django eine deutlich spezialisiertere Lösung als Flask darstellt. Ein Startup oder Unternehmen kann nahezu alle seine Dienste ausschließlich mit Django betreiben, während Flask aufgrund seines geringen Ressourcenbedarfs häufig in Kombination mit anderen Technologien eingesetzt wird.

Der vielversprechendste Ansatz ist, zunächst Python wirklich zu beherrschen und dann Webentwicklungskenntnisse mit Django oder Flask (idealerweise beides!) hinzuzufügen.

Gemeinschaft

Django verfügt über die größere und besser organisierte Community der beiden Frameworks. Über 1.800 Entwickler tragen zum Django-Quellcode bei, im Vergleich zu etwa 550 bei Flask. Auf Stack Overflow finden sich rund 212.500 Fragen zu Django, verglichen mit etwa 31.500 Fragen zu Flask .

Django veranstaltet außerdem jährliche Konferenzen in den USA , Europa und Australien . Flask kann nicht auf ein vergleichbares Konferenzniveau zurückblicken, obwohl beide Frameworks auf PyCon- Veranstaltungen rege diskutiert werden.

Was ist eine Flasche?

Flask ist ein bewusst minimalistisches und flexibles Mikro-Framework, dessen Nutzen dadurch aber keineswegs eingeschränkt wird. Wie wir sehen werden, bringt diese Designentscheidung sowohl Stärken als auch Schwächen mit sich.

Flask entstand 2010 als Aprilscherz von Armin Ronacher und wurde vom Sinatra Ruby-Framework inspiriert. Flask sollte so einfach sein, dass es in eine einzige Python-Datei passte, und trotz seines

humorvollen Ursprungs erfreute es sich aufgrund seiner Einfachheit und Flexibilität schnell großer Beliebtheit.

Flask selbst hat eine recht kleine Codebasis und stützt sich stark auf zwei wichtige Abhängigkeiten: **Werkzeug** und **Jinja** , die beide ursprünglich von Armin Ronacher entwickelt wurden.

Werkzeug ist ein WSGI-Toolkit (Web Server Gateway Interface), das die Kernfunktionalität für Flask bereitstellt. Es verarbeitet HTTP-Anfragen und -Antworten, bietet ein URL-Routing-System, einen integrierten Entwicklungsserver, einen interaktiven Debugger, einen Testclient und Middleware. Jinja ist eine Template-Engine zur Generierung dynamischer HTML-Dokumente mit eigener Syntax für grundlegende Logik, Variablen, if/else-Schleifen, Template-Vererbung und mehr.

Obwohl Flask keine spezifische Struktur vorschreibt, wird es häufig im Model-View-Controller (MVC)-Muster verwendet, das auch bei anderen Web-Frameworks wie Ruby on Rails üblich ist.

- **Modell** : Interagiert mit der Datenbank und verarbeitet die Datenlogik.
- **Ansicht** : Rendert (in der Regel) HTML-Vorlagen mit Daten für den Benutzer.
- **Controller** : Verarbeitet Benutzereingaben, interagiert mit dem Modell und wählt die darzustellende Ansicht aus.

Flasks Mikroframework-Architektur bedeutet, dass es einige wenige Aufgaben extrem gut erledigt und für die restlichen Funktionen auf Drittanbieterbibliotheken (und den Entwickler) zurückgreift. Dieser Ansatz eignet sich gut für kleinere Webanwendungen, die nicht den vollen Funktionsumfang von Django benötigen. Am anderen Ende des Spektrums bevorzugen erfahrene Programmierer, die die vollständige Kontrolle über ihre Anwendung wünschen, oft Flask. Dies bedeutet jedoch, dass sie mehr Designentscheidungen treffen müssen als bei einem umfassenden Framework wie Django.

Was ist Django?

Django ist ein leistungsstarkes Python-Webframework, das schnelle Entwicklung und ein klares, pragmatisches Design fördert. Es wurde bei der Zeitung Lawrence Journal-World entwickelt und 2005 veröffentlicht. „Leistungsstark“ bedeutet, dass Django den Programmieraufwand bei der Webanwendungsentwicklung minimiert, indem es für die meisten Anwendungsfälle integrierte Funktionen bereitstellt, darunter ein ORM (Objektrelationales Mapping), URL-Routing, eine Template-Engine, Formularverarbeitung, ein Authentifizierungssystem, eine Administrationsoberfläche und robuste Sicherheitsfunktionen. Im Gegensatz zu Flask, wo Entwickler

diese Funktionen selbst auswählen und implementieren müssen, sind sie bei Django standardmäßig enthalten.

Django wird von der gemeinnützigen Django Software Foundation verwaltet und verfügt über eine große und engagierte Community, die an neuen Versionen, umfangreicher Dokumentation, aktiven Online-Communities und regelmäßigen Community-Konferenzen arbeitet.

Django folgt einer Variante der MVC-Architektur, dem sogenannten Model-View-Template (MVT)-Muster, das die Trennung der Belange betont:

- **Modell** : Verarbeitet Daten und Geschäftslogik, einschließlich Methoden zur Interaktion mit den Daten.
- **Ansicht** : Verarbeitet Geschäftslogik und interagiert mit dem Modell und der Vorlage. Sie verarbeitet außerdem Benutzeranfragen.
- **Templates** : Sie rendern die Benutzeroberfläche, üblicherweise als HTML mithilfe der Django-Templating-Sprache.

Eine vierte Komponente, **URLs** , ist ebenfalls enthalten und dient der URL-Weiterleitung. Dabei wird eine Benutzeranfrage einer bestimmten Ansicht zugeordnet, die dann eine Antwort generiert.

Flask: Hallo Welt

Python sollte bereits auf Ihrem Computer installiert sein, daher müssen wir lediglich eine virtuelle Umgebung erstellen und Flask installieren.

```
# Windows
$ python -m venv .venv
$ .venv\Scripts\Activate.ps1
(.venv) $ python -m pip install flask

# macOS/Linux
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ python -m pip install flask
```

Erstellen Sie mit Ihrem Texteditor eine neue Datei namens `**hello.py**`. Flask benötigt bekanntermaßen nur fünf Zeilen für eine Hello World-Webseite.

```
# app.py
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Dieser Code importiert die `Flask`Klasse am Anfang und erstellt `app`in der nächsten Zeile eine Instanz. Der `route()`Dekorator teilt Flask mit, welche URL die Funktion auslösen soll; hier ist sie auf die Startseite gesetzt `/`. Die Funktion gibt dann einen HTML-String zwischen Absatz -Tags mit unserer Nachricht `hello_world`zurück .`<p>`

Um den Code auszuführen, verwenden Sie den `flask run`Befehl.

```
(.venv)
$ flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deploymen
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Wenn Sie die Adresse `127.0.0.1:5000`in Ihrem Webbrowser aufrufen, wird die Meldung angezeigt. Flask verwendet `5000`standardmäßig Port 10.

Das ist so einfach wie nur möglich und verdeutlicht sowohl Flasks Wurzeln als Versuch, Webanwendungen mit nur einer einzigen Datei zu erstellen, als auch seine inhärente Flexibilität.

Django: Hallo Welt

Die Django-Dokumentation bietet keine vergleichbare Schnellstartanleitung, aber wir können mit nur wenigen zusätzlichen Codezeilen ein ähnliches Ergebnis erzielen. Tatsächlich ist dies unter erfahrenen Django-Entwicklern zu einer Art Wettbewerb geworden, und es gibt sogar ein eigenes Repository, [django-microframework] , das sich diesen Bemühungen widmet. Wir entscheiden uns für Option 2, die zwar nicht die prägnanteste, aber leichter verständlich ist als einige der anderen Ansätze.

Navigieren Sie zu einem neuen Verzeichnis, beispielsweise `**django**`einem Verzeichnis namens „Desktop“, und erstellen Sie eine virtuelle Umgebung, die Django enthält.

```
# Windows
> cd onedrive\desktop\code
> mkdir django
> cd django
> python -m venv .venv
> .venv\Scripts\Activate.ps1
(.venv) > python -m pip install django

# macOS
% cd ~/desktop/code
% mkdir django
% cd django
% python3 -m venv .venv
% source .venv/bin/activate
(.venv) % python3 -m pip install django
```

Erstellen Sie in Ihrem Texteditor eine `**hello_django.py**`Datei mit folgendem Code:

```
# hello_django.py
from django.conf import settings
from django.core.handlers.wsgi import WSGIHandler
from django.core.management import execute_from_command_line
from django.http import HttpResponse
from django.urls import path

settings.configure(
    ROOT_URLCONF=__name__,
    DEBUG=True,
)
```

```
def hello_world(request):
    return HttpResponse("Hello, Django!")


urlpatterns = [path("", hello_world)]

application = WSGIHandler()

if __name__ == "__main__":
    execute_from_command_line()
```

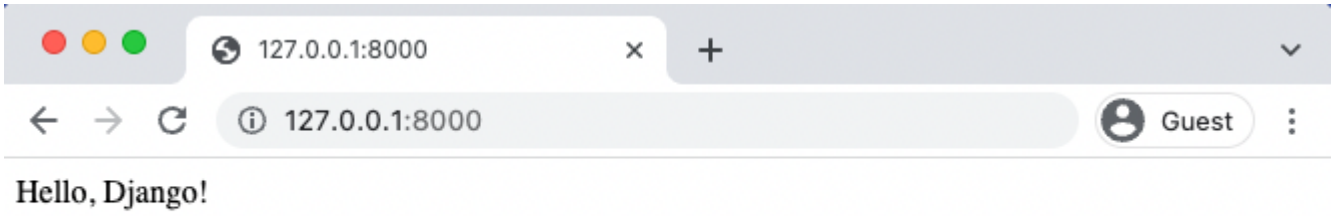
Django ist für größere Webanwendungen konzipiert und verwendet typischerweise eine globale `**settings.py**`Konfigurationsdatei. Wir können jedoch alle benötigten Elemente in einer einzigen Datei importieren. Die wichtigsten Referenzpunkte sind die `**hello_world**`Funktion, die den String „Hello, Django!“ zurückgibt, und die `**urlpatterns**`Definition unserer URL-Routen, insbesondere `/`**""**`home/home`, was einen leeren String bedeutet und somit die Startseite darstellt.

Starten Sie den in Django integrierten Server mit dem `**runserver**`Befehl

```
(.venv) > python hello_django.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
July 17, 2024 - 13:48:54
Django version 5.0, using settings None
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Navigieren Sie zum Standardport von Django, 8000, http://127.0.0.1:8000/ , um die Meldung „Hello, Django!“ zu sehen.



Django benötigte zwölf Codezeilen, Flask hingegen nur fünf. Beide Beispiele sind jedoch als Schnellstartanleitungen gedacht; sie entsprechen nicht der Strukturierung einer realen Flask- oder Django-Anwendung.

Flask Persönliche Website

Jetzt erstellen wir eine persönliche Website mit einer Startseite und einer Über-uns-Seite. Dies bietet die Möglichkeit, Vorlagen einzuführen und einige der Muster zu wiederholen, die wir bei der Definition von Routen in Flask kennengelernt haben.

Aktualisieren Sie die `app.py`Datei wie folgt:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Die Funktionen `view` `home`und `about``devices` geben nun jeweils eine Vorlage zurück. Wir verwenden außerdem `route()`wieder den Dekorator, um den URL-Pfad für jede Funktion zu definieren. Zusätzlich haben wir `debug=True`unten eine Zeile hinzugefügt, sodass der Entwicklungsserver nun im Debug-Modus läuft.

Im nächsten Schritt erstellen wir unsere beiden Templates. Flask sucht die Template-Dateien in einem `templates`Verzeichnis, also erstellen Sie dieses jetzt.

```
(.venv) $ mkdir templates
```

Fügen Sie darin die beiden Dateien mit folgendem Code hinzu:

1. Erstellen Sie Vorlagen in `templates/`:

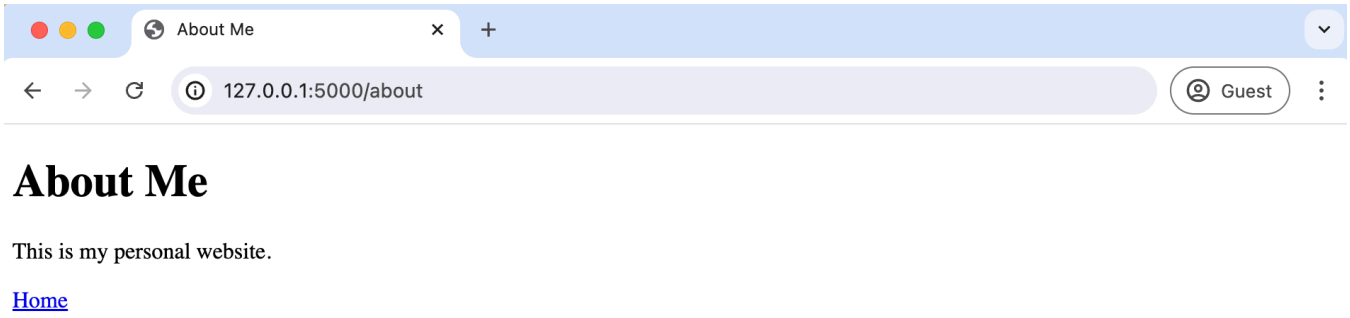
```
<!-- templates/home.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Personal Website</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <a href="{{ url_for('about') }}">About Me</a>
</body>
</html>
```

```
<!-- templates/about.html -->
<!DOCTYPE html>
<html>
<head>
    <title>About Me</title>
</head>
<body>
    <h1>About Me</h1>
    <p>This is my personal website.</p>
    <a href="{{ url_for('home') }}">Home</a>
</body>
</html>
```

Jede Datei verwendet die Methode url_for , um Links basierend auf dem Namen der Ansichtsfunktion zu definieren.

Starten Sie den Server erneut `flask run`und navigieren Sie zur Startseite:

Klicken Sie anschließend auf den Link „Über mich“.



Dies ist ein einfaches Beispiel, aber es verdeutlicht, wie Templates und Views in Flask interagieren. Wir verwenden noch eine einzige Python-Hauptdatei, die alles steuert. Sobald wir jedoch mehr Seiten haben und Logik einführen, ist der Ansatz mit einer einzigen Datei nicht mehr sinnvoll, und es wird Zeit, die Flask-Anwendung anders zu strukturieren. In der Flask-Community gibt es einige gängige Muster, die letztendliche Entscheidung liegt aber beim Entwickler.

Django: Persönliche Website

Django ist für umfangreiche Webanwendungen konzipiert, daher bietet die Erstellung einer persönlichen Website die Möglichkeit, dies in der Praxis zu erleben. Wir beginnen mit der Erstellung eines Projekts, dem zentralen Knotenpunkt unserer Website, mithilfe des `startproject` entsprechenden Befehls.

```
(.venv) $ django-admin startproject django_project .
```

Wir haben dem Projekt `django_project` hier einen Namen gegeben. Der Punkt im Namen `.` bedeutet, dass der neue Ordner und die Dateien im aktuellen Verzeichnis installiert werden. Ohne Punkt erstellt Django ein neues Verzeichnis und fügt den Projektordner und die Dateien dort hinzu .

So sollte Ihr Verzeichnis nun aussehen. Die `hello_django.py` Datei bleibt erhalten und kann entweder dort bleiben oder vollständig gelöscht werden: Wir benötigen sie nicht mehr. Es gibt einen komplett neuen `django_project` Ordner mit mehreren Dateien und einer `manage.py` Datei zum Ausführen von Django-Befehlen.

```
├── django_project
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── hello_django.py
└── manage.py
```

Wir möchten nun mithilfe dieser `**startapp**`Befehle eine App erstellen, die wir „app“ nennen werden `**pages**`. Ein einzelnes Django-Projekt besteht typischerweise aus mehreren Apps für unterschiedliche Funktionen. Wenn wir beispielsweise die Benutzerregistrierung hinzufügen, sollte der entsprechende Code in einer eigenen App implementiert werden, ebenso wie der Code für Zahlungen usw. Dies hilft Entwicklern, ihren Code besser zu strukturieren.

```
(.venv) $ python manage.py startapp pages.
```

Dieser Befehl erstellt ein `**pages**`Verzeichnis mit folgenden Dateien:

```
└── pages
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Im ersten Schritt aktualisieren wir die `**django_project/settings.py**`Datei, um Django über unsere neue App zu informieren. Dies ist eine globale Einstellungsdatei für das gesamte Projekt.

```
# django_project/settings.py
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
```

```
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "pages",  # new
]
```

Als Nächstes aktualisieren wir die `**django_project/urls.py**`Datei. Jede eingehende URL-Anfrage wird zuerst auf diese Datei zugegriffen und anschließend entweder verarbeitet oder an eine bestimmte Anwendung weitergeleitet. In diesem Fall möchten wir Anfragen an die `**pages**`Anwendung senden. Dazu importieren wir die Datei `**include**`und legen einen neuen Pfad unter ` <home>` fest `""`, was der Startseite entspricht. Django bindet standardmäßig die URL-Konfiguration für die integrierte Administrationsoberfläche ein, eine leistungsstarke visuelle Möglichkeit zur Interaktion mit Ihrer Datenbank.

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

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("pages.urls")),  # new
]
```

Innerhalb der `**pages**`App benötigen wir eine Ansicht und eine URL-Datei. Beginnen wir mit der Ansicht unter `**pages/views.py**`.

```
# pages/views.py
from django.shortcuts import render


def home(request):
    return render(request, "home.html")


def about(request):
    return render(request, "about.html")
```

In Django empfangen Views Webanfragen und geben Webantworten zurück. Der `**request**`Parameter ist ein Objekt mit Metadaten zur Benutzeranfrage. Wir definieren hier zwei funktionsbasierte Views, `view` `**home**`und `**about**``view`, die die Kurzform `contains`

verwenden `render`, um eine Vorlage mit einem `HttpResponse`an den Benutzer zurückgesendeten Objekt zu kombinieren. Die beiden Vorlagen sind `view` `home.html`und `view` `about.html`.

Für die Templates können wir ein `templates`Verzeichnis innerhalb des Projekts erstellen `pages`, darin ein weiteres Verzeichnis mit dem App-Namen und schließlich unsere Template-Dateien. Dieser Ansatz beseitigt jegliche Bedenken hinsichtlich einer möglichen Beeinträchtigung des Django-Template-Loaders in größeren Projekten.

```
(.venv) $ mkdir pages/templates
(.venv) $ mkdir pages/templates/pages
```

Fügen Sie anschließend in Ihrem Texteditor zwei neue Dateien hinzu: `pages/templates/pages/home.html`und `pages/templates/pages/about.html`.

```
<!-- pages/templates/pages/home.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Personal Website</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <a href="{% url 'about' %}">About Me</a>
</body>
</html>
```

```
<!-- pages/templates/pages/about.html -->
<!DOCTYPE html>
<html>
<head>
    <title>About Me</title>
</head>
<body>
    <h1>About Me</h1>
    <p>This is my personal website.</p>
    <a href="{% url 'home' %}">Home</a>
</body>
</html>
```

Im letzten Schritt müssen die URLs für diese beiden Seiten konfiguriert werden. Erstellen Sie dazu `urls.py` innerhalb der `pages` App eine Datei mit folgendem Code.

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

urlpatterns = [
    path("", views.home, name="home"),
    path("about/", views.about, name="about"),
]
```

Ganz oben importieren wir unsere Ansichten und legen dann für jede einen URL-Pfad fest. Die Syntax besteht aus der Definition des URL-Pfads, dem Ansichtsnamen und optional dem Hinzufügen einer URL, `name` die es uns ermöglicht, in unseren Vorlagen auf die einzelnen Pfade zu verlinken.

Starten Sie den lokalen Django-Server mit dem `runserver` Befehl.

```
(.venv) $ python manage.py runserver
```

Sie können die Startseite sehen:

Welcome to My Website

About Me

Klicken Sie auf „Über mich“, um zur Über-mich-Seite weitergeleitet zu werden:

About Me

This is my personal website.

Home

Wie Sie sehen können, benötigte Django mehr Gerüst als Flask, dieser Ansatz bietet jedoch eine konsistente Struktur, die gut skalierbar ist.

Detaillierter Vergleich

Der tatsächliche Vergleich dieser Web-Frameworks hängt von den Anforderungen Ihres Projekts ab. Entwickeln Sie eine klassische Webanwendung mit Datenbankanbindung, CRUD-Funktionalität (Erstellen, Lesen, Aktualisieren, Löschen) und Benutzerauthentifizierung? Dann bietet Django integrierte Lösungen für all diese Anforderungen. Flask hingegen erfordert die Installation zahlreicher Drittanbieterbibliotheken: `**Flask-SQLAlchemy**`für die Datenbankanbindung, `**Flask-Migrate**`die Verwaltung von Datenbankmigrationen, `**Flask-WTF**`Formulare `**WTForms**`, `**Flask-Login**`Benutzerauthentifizierung, `**FLask-Mail**`E-Mail-Unterstützung, `**Flask-Security**`Sicherheitsfunktionen, `**Flask-Admin**`eine Administrationsoberfläche zur Verwaltung von Anwendungsdaten, `**Flask-Caching**`Caching, `**Flask-BCrypt**`Passwort-Hashing usw.

Die Stärke von Django liegt darin, dass man sich um all das nicht kümmern muss. Die benötigten Komponenten sind integriert, getestet und werden von der Community unterstützt. Bei Flask hingegen sind die Drittanbieterbibliotheken nicht so eng eingebunden und erfordern eine manuelle Installation durch den Entwickler. Das bietet zwar mehr Flexibilität, setzt aber auch mehr Programmiererfahrung voraus.

Abschluss

Letztendlich können Sie mit Flask oder Django für Ihre Webanwendungen nichts falsch machen. Beide sind ausgereift, skalierbar und gut dokumentiert. Der Unterschied liegt im Ansatz, und am besten finden Sie heraus, was Ihnen besser gefällt, indem Sie beide anhand komplexerer Projekte ausprobieren.

Wenn Sie mehr über Django erfahren möchten, empfehle ich Ihnen „Django für Anfänger“. Dort lernen Sie, wie Sie sechs zunehmend komplexere Webanwendungen erstellen, inklusive Tests und Deployment. Für Flask gibt es das Flask Mega-Tutorial als kostenlose Online-Version. Außerdem sind zwei Kurse auf TestDriven.io empfehlenswert: „TDD mit Python, Flask und Docker“ und „Authentifizierung mit Flask, React und Docker“ . Falls Sie Videos bevorzugen, finden Sie auf Udemy zahlreiche Flask-Kurse. Der beste Videokurs, den ich kenne, ist „ Eine SaaS-App mit Flask und Docker erstellen“ .

Bleiben Sie auf dem Laufenden

you@domain.com Abonnieren