Flask Vs Django

Django is a full-stack web framework, whereas Flask is a micro and lightweight web framework. The features provided by Django help developers to build large and complex web applications. On the other hand, Flask accelerates development of simple web applications by providing the required functionality

Features	Flask	Django
Type	Lightweight micro framework.	Full-stack framework.
Template Engine	Flask is based on the Jinja2 format motor.	Built-in template engine.
Admin	Does not have a feature to manage administration tasks, but it does have an extension: Flask-Admin.	Django already comes with an admin panel
Database	Has many libraries (e.g., SQLAlchemy, PyMongo, and PonyORM) and extensions (e.g., Flask-Peewee, Flask-Pony, and Flask-Alembic) available.	Built-in ORM, that enables developers to work with several relational database systems.
Development	Provides more control over the components the developer wishes to implement.	Since it integrates all the "batteries", developers have the required tools at their disposal for quick implementation.
Flexibility	High flexibility - Flask enables developers to build and add functionalities to simple applications in a flexible way.	Low flexibility - Developers are not as free to use other plugins and libraries as they are with Flask.
Learn	Easier to learn.	High learning curve.

Features of Flask

Here, are important features of Flask

- Integrated support for unit testing.
- RESTful request dispatching.
- Uses a Ninja2 template engine.
- It is based on Werkzeug toolkit.
- Support for secure cookies (client-side sessions).
- Extensive documentation.
- Google app engine compatibility.
- APIs are nicely shaped and coherent
- Easily deployable in production

Features of Django

Here are important features of Django:

- Offers Model - View – Controller (MVC) Architecture.
- Predefined libraries for imaging, graphics, scientific calculations, etc.
- Supports for multiple databases.
- Cross-platform operating system.
- Site optimization across specialized servers
- Support for front-end tools like Ajax, jQuery, Pyjamas, etc.
- Supports multi-Language and multi-characters.

Advantages of Flask

- Here, are pros/benefits of using Flask
- Higher compatibility with latest technologies
- Technical experimentation
- Easier to use for simple cases
- Codebase size is relatively smaller
- High scalability for simple applications,
- Easy to build a quick prototype
- Routing URL is easy
- Easy to develop and maintain applications
- Database integration is easy
- Small core and easily extensible
- Minimal yet powerful platform
- Lots of resources available online especially on GitHub

Advantages of Django

- Here, are pros/benefits of Django framework:
- Django is easy to set up and run
- It provides an easy to use interface for various administrative activities.
- It offers multilingual websites by using its built-in internationalization system
- Django allows end-to-end application testing
- Allows you to document your API with an HTML output
- REST Framework has rich support for several authentication protocols
- It is used for rate-limiting API requests from a single user.
- Helps you to define patterns for the URLs in your application
- Offers built-in authentication system
- Cache framework comes with multiple cache mechanisms.
- High-level framework for rapid web development
- A complete stack of tools
- Data modelled with Python classes

Disadvantage of Flask

- Here, are cons/drawback of Flask
- Slower MVP development in most cases,
- Higher maintenance costs for more complex systems
- Complicated maintenance for larger implementations.
- Async may be a little problem
- Lack of database and ORM
- Setting up a large project requires some previous knowledge of the framework
- Offers limited support and smaller community compared to Django

Disadvantage of Django

- Here, are cons/drawback of the Django framework
- It is a monolithic platform.
- High dependence on Django ORM. Broad Knowledge required.
- Fewer Design decisions and Components.
- Compatibility with the latest technologies
- A higher entry point for simple solutions
- The larger size of the code
- Too bloated for small projects
- Underpowered templating and ORM
- Templates failed silently
- Auto reload restarts the entire server
- High learning curve
- Documentations does not cover real-world scenarios
- Only allows you to handle a single request per time.
- Routing requires some knowledge of regular expressions
- Internal subcomponents coupling
- You can deploy components together, which can create confusion.

Code (Django) (https://github.com/sohail-datascientist/Django-Hello-World)

Creating a Django Application requires few commands on the CMD like,

- First of all check whether it is installed or not, checking through *(pip list)*
- ***Django-admin startproject project_name*** **(for starting to create an application)**
- Executing this command will create a folder with some of the file like, *_init.py, views.py, urls.py, etc.*
- In the folder there will be a file named as manage.py using it we can create the application.
- ***Python manage.py startapp app_name***
- Now we need to manage internal files like giving the paths and setting the reference
- First we have to add URL to the urls.py file of our project (refer the URL of our application)

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls),
path('',include('Task01.urls'))
]
```

- After this setting of URL we need to go to the URL file our application (It is created by user it's not automatically created)
- Also in that urls.py file we need to add paths of the file where we show the data , like in the views.py we need to display the data at our screen so we can add that path in urls.py like:

```
from django.urls import path
from .import views
urlpatterns = [
path('', views.Task_1)
]
```

- In views we can create a program like:

```
from django.shortcuts import render, HttpResponse
def Task_1(request):
        return HttpResponse("Hello The World of Aleins")
```

- *In setting we need to add the name of application in the list of INSTALLD_APPS*

After completing this we need to run server and execute the program at the given URL.

- ***Python manage.py runserver***

Code (Flask) (https://github.com/sohail-datascientist/Flask-Hello-World)

- Initially check that the flask is available or not, if it is not available the install it using ***pip install flask***
- After it create a folder in and in that folder place a file with the ***.py*** extension
- In .py file write the following code:

```
from flask import Flask
app=Flask(__name__)
@app.route("/")
def Task_01():
    return "Hello World of Programmers"
```

- Now copy open the folder copy the path and paste it into the CMD Command line
- Now ***set FLASK_APP=File_name_with_.py_extension***
- Executer the last command that is ***flask run***
- Now copy the URL and paste it into the browser your code will execute.