Research

Goals:

Introduction about Flask and compare Flask with another web framework like Django. I use developing a mini web application to illustrate the contrasting how Flask would do it versus Django.

1. What is Flask technology?

a. Flask Overview

Flask is a Python API for developing web applications. Armin Ronacher devised it. Flask's architecture is more explicit than Django's framework and is also easier to understand since it takes fewer base code to execute a basic web application. (Madeira)

A Web-Application Platform, also known as a Web Framework, is a series of modules and libraries that allow developers to build applications rapidly. It does not require the developers to put much effort into writing low-level code such as protocols, thread management, and so on. Flask built on the WSGI toolkit and the Jinja2 template engine. (Amigos-Maker)

b. Some of the reason the python web developer should use Flask Framework

Python includes tools for developers to use to build code or software within a given structure. It allows them to concentrate on the logic of the program rather than other information. Although Python has many frameworks, including Web2Py, TurboGears, and Django, developers tend to use Flask for web application creation. Flask ranked as the most successful system by 47% of survey respondents in the Python Developers Survey. Flask is a Python platform that is both lightweight and extensible. It is referred to as a micro framework because it lacks the extra functionality which comes with a full-stack Python framework, such as Django.

There are some reasons why the python web developer loves Flask Framework.

Quick to learn: One of the reasons we like Flask is that it is a non-fussy, easy-to-use system. Any developer who is new or seasoned will quickly learn it and begin using it to create web applications. The well-documented source code is also helpful for beginners studying Python. It does not overburden developers with technical information. Until going on to full-stack frameworks like Django, we suggest that novice developers or others with less Python knowledge learn Flask and experiment with building small bits of web applications. (Bitsadmin)

Highly scalable: When developing a web application in a fast-paced environment, it is often preferable to start small and then scale up. It is where a system like Flask will help. Flask is very scalable. As it grows in popularity, it can handle a large number of requests every day. Consider the website Pinterest. By using Flask, it was able to process up to 12 billion requests a day. Flask modularized code, allowing developers to separate it into discrete bits reused as the code base expands. (Bitsadmin)

Flexible project layout: Flask's core function is flexibility. It is helpful when developers must interact with several workflows and systems. Flask provides a web platform extension. It helps developers to configure their apps in whatever way they choose. It helps us to reduce problems that can occur as a result of the rigidity of other systems. It frees them from having to focus on unique technological components. Instead, developers can use any modules they choose to design their web applications. Flask's degree of freedom applies to us. (Bitsadmin)

Facilitates experimentation: While we're on the subject of liberty, we should mention that Flask is an excellent platform for playing with web applications. Flask is much more adaptable to new digital developments in the software industry than the monolithic form of a framework like Django. Flask can help with quicker deployment in an agile work setting where developers are continually developing their product. So, if a developer wants to add more features to a product, they can use Flask to complete it quickly. (Bitsadmin)

Better code and extension power: Since the code base is limited, developers have more control of their codes. Additionally, developers prefer Flask because it helps them to pick and select which components to use. Flask empowers users to make more choices when considering elements for a web application. They also have total leverage over their extensions. For example, if a web application does not need database access or form validation, the developer can change to remove certain features. However, instead include those that apply to their application. (Bitsadmin)

c. **Flask Environment**

Python projects occur in virtual worlds. Each project exists in a distinct virtual world. This avoids kit clashes. Python packages can not be built on a system-wide basis.

Create a Virtual Environment

Launch a terminal (see below how to open one quickly). Then, mount python3-venv.

On Ubuntu Linux, use the following command:

sudo apt-get install python3-venv

First, create a project directory with the command

$ mkdir flaskexample

cd flaskexample

Then you can create a new virtual environment with the command:

$ python3 -m venv venv

Activate Virtual Environment

The virtual environment has been created, but it's not yet active.

Activate the virtual environment on Linux, use the command:

source venv/bin/activate

On Microsoft Windows use this instead:

$ venv\Scripts\activate

d. Structure of the Flask Application

Explain the structure of a Flask application

Project Tree

Create your project structure using the following directory hierarchy:

```
run.py
▾ bin/
▾ bookshelf/
    __init__.py
    ▾ admin/
        controllers.py
        __init__.py
    ▾ main/
          controllers.py
          __init__.py
```

```
▾ docs/
▾ tests/
```

Figure 1- Project Tree

Source from https://damyan.blog/post/flask-series-structure/

Brief Description of the Application Structure

- bin – in this folder you could place your scripts, that will be executed on the command line;
- docs – in this folder you could place project related documentation files;
- tests – this folder contains your unit tests
- [app_name] – contains the application itself
- run.py – this file is used to run the Flask application, where the contents of the file

The Application Folder

Based on the blueprint principle in Flask, the program is designed to be modular. Flask blueprints allow developers to automate and organize applications while also providing a central place for Flask extensions to log operations on applications. The blueprint is identical to the Flask application object in that it is used to create or expand an application. (How to Structure ...)

2. Some technology inside Flask

2.1 Flask – SQLite

SQLite support is integrated into Python. The SQlite3 module is provided with the Python distribution. Please see this page for a comprehensive tutorial on using the SQLite database in Python.

https://www.tutorialspoint.com/sqlite/sqlite_python.htm

2.2 Flask – SQLAlchemy

Implementing raw SQL in Flask web apps to run database CRUD operations can be time-consuming. Alternatively, SQLAlchemy, a Python framework, is a versatile OR

Mapper that offers full SQL control and versatility to web developers.
Flask-SQLAlchemy is a Flask extension that applies SQLAlchemy support to your Flask program.

2.3 Flask – File Uploading

It is very easy to manage file uploads in Flask. It includes an HTML form with the type attribute set to 'multipart/form-data' and the file posted to a URL. The URL handler retrieves a file from the request.files[] object and saves it to the specified location.

Before being saved to its final location, each uploaded file is first saved in a temporary location on the server. The filename property of the request.files[file] object can be used to access the name of the destination file, which can be hard-coded. However, using the secure filename() feature to obtain a safe version is preferred.

In the Flask object's configuration settings, you can decide the path to the default upload folder and the maximum file size that can be uploaded.

app.config['UPLOAD_FOLDER']	Defines path for upload folder
app.config['MAX_CONTENT_PATH']	Specifies the maximum size of file uploaded – in bytes

Source from https://www.tutorialspoint.com/flask/flask_file_uploading.htm

2.4 Flask – Mail

A web-based program is often expected to have the ability to send email to users/clients. The Flask-Mail extension makes it very convenient to set up a simple GUI for every email server. (Flask Tutorial)

Methods of Mail class

No	Methods & Description
1	send() Sends contents of Message class object
2	connect() Opens connection with the mail host
3	send_message() Sends message object

3. What are template engines in the Flask?

Templating allows data to be represented in a variety of ways by combining prototype and data. The prototype is a document of placeholders that will be filled in with real data as the template is interpreted by the template engine. The template will provide control structures such as for loops of declarations. Flask's template engine of choice is Jinja2. Jinja2 enables developers to generate various output result files based on a single template text file. (Flask Tutorial)

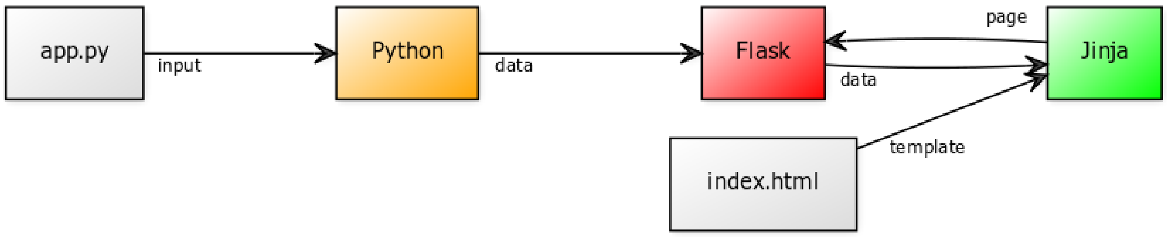

Figure 2- Template Structure

Source from https://pythonbasics.org/flask-tutorial-templates/

4. Compare Flask with another web framework like Django?

Advantage and Disadvantage between two python Frameworks.

Main contrasts:

Flask offers ease of use, versatility, and fine-grained power. It is the objective of it to let you decide how you want to implement things. Django provides an all-inclusive experience, with an admin screen, database applications, an ORM, and directory layout for your software and projects.

You should probably choose Flask, whether you're interested in the experience and learning experiences, or if you want more discretion over which elements to use, such as which databases to use and how to communicate with them. If you're just worried about the end goal, Django is the way to go. Especially if you're working on a basic application like a news site, an e-store, or a blog, and you want there to be a single, obvious way of doing stuff.

Both structures are gaining traction, as seen by the amount of StackOverflow questions for each in the picture below.

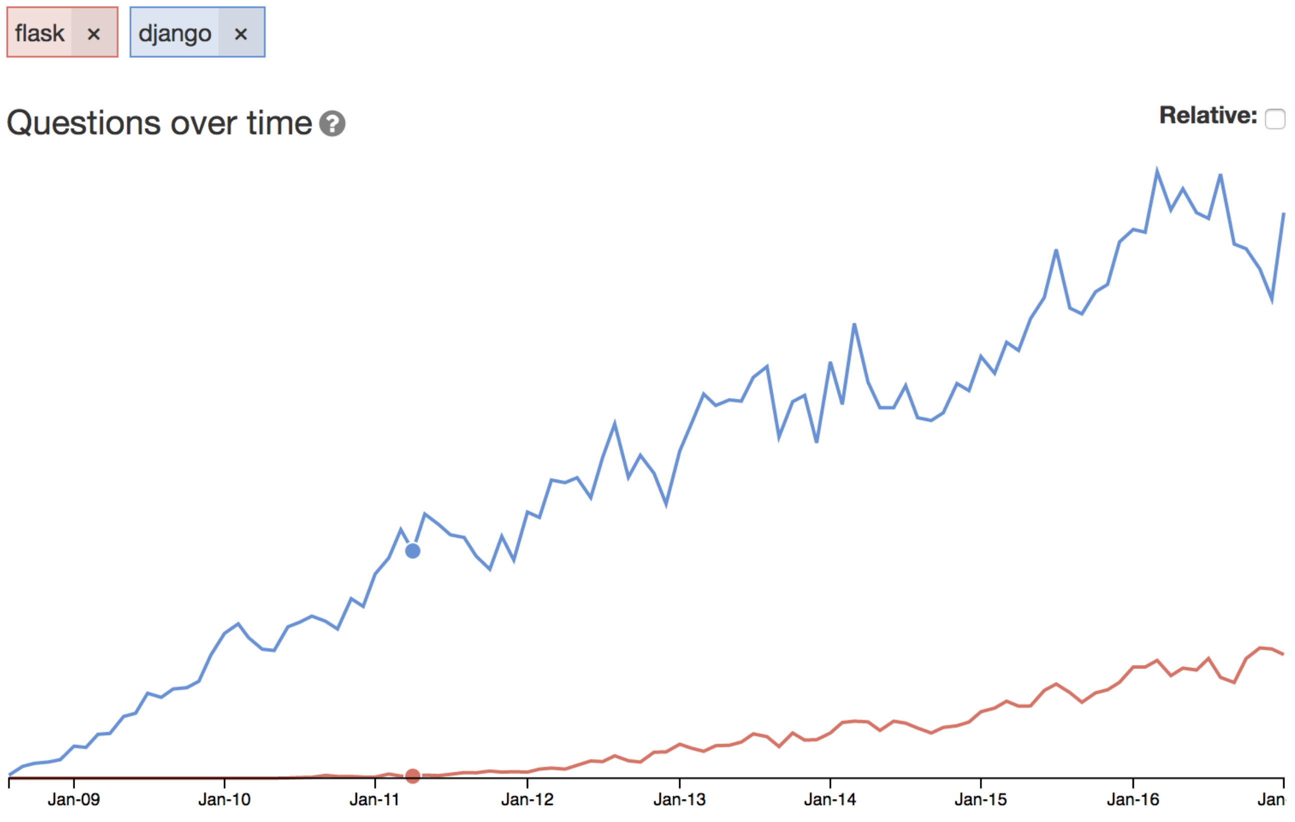

Figure 3- Questions relevant to both frameworks

Source from https://www.codementor.io/@garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v

5. How to use developing a mini web application to illustrate the contrast between how Flask would do it vs. Django?

When trying a new technique, the first thing many people do is follow the shortest series of steps that generates the result "Hello, World!" Below, I will go through the steps to create "Hello, World!" applications with Flask and Django. Seeing two Hello World projects would provide us with a greater understanding of both the two systems and enable us to explain some of the components inside them. (Dwyer)

<u>Flask</u>

Create a file with any name but the best place is hello.py (absolutely not set as flask.py due to the same name flask)

```python
from flask import Flask
app = Flask("my website")

@app.route("/")
def hello_world():
   return "Hello world"

@app.route("/hello/<string:name>")
def hello(name):
    return "Hello {}".format(name)
```

(GeeksforGeeks)

Then run the application by executing some statement as below

```
$ export FLASK_APP=hello.py
$ flask run

 * Running on http://127.0.0.1:5000/
```

The most impressive point here is that only 10 lines, in exactly 1 file, that run into the web page.

The total number of actions to do: 3 - create a file, fill in the content, type commands to run.

Django

Django does not make a web in a file, it uses built-in commands to generate files, ie we only have to type commands, not type a lot of code. Type the following 3 commands to run the web page:

```
$ django-admin startproject project1
$ cd project1/
$ ./manage.py startapp hello
$ ./manage.py runserver

Starting development server at http://127.0.0.1:8000/
```

We've got our Django's sample site right away - not even typed in any code. To have the same result as the Flask above, need to edit 2 files:

File hello / views.py, write functions like flask:

```
from django.http import HttpResponse

def hello_world(request):
    return HttpResponse("Hello world")

def hello(request, name):
    return HttpResponse("Hello {}".format(name))
```

File project1 / urls.py add 3 lines:

```
from hello import views

urlpatterns = [  # already
    path('admin/', admin.site.urls),  # already
    path('', views.hello_world, name='hello_world'),
    path('hello/<str:name>', views.hello, name='hello'),
] # already
```

Now run ./manage runserver will have the same result as Flask.

The total number of lines of code to add to Django is 8 - exactly the same number of lines Flask has to write - excluding blank lines. Django has to run more than 2 commands - but only needs to do it at the start of the project. Flask writes code in one file, and Django has to edit 2 files.

The main difference between Django and Flask is the way the URL is written. The Flask user creates a Flask object, calls it app, and then uses decorator

```
@app.route("/hello/<string:name>")
```

to append the path / hello / name to the function right below it (function hello). Decorator is not a simple concept here, although the user can just close his eyes and then type @ follow and roughly understand that attaching a path is fine.
Django chooses a solution to concatenate the path with the function via file urls.py, import module views from the directory hello / (created at runtime "startapp hello"), then assign the path by calling the function path:

```
   path('hello/<str:name>', views.hello, name='hello'),
```

Django is a bit longer than Flask, it must write name = "hello", without / at the beginning of the URL, and str is written as str, not a string like Flask.
In the beginning, it was difficult to see why Django wrote its own file urls.py when the @ flask was very neat. Then by the time you have 20 URLs, you will understand why. How do I list all URLs in my app? Flask URLs are scattered where, each is at the top of a

function, if each function is 50 lines long, it will be a lot of work - either with the search function or with the IDE to support it. With Django, all are neatly contained in urls.py.

Add a page to use the template

Flask

Create a folder named templates right next to file hello.py, then create index.html file containing HTML code and Jinja2 template.

```
{% for fw in frameworks %}
  {% if fw|length > 5 %}
    <b>{{ fw }}</b>
  {% else %}
    {{ fw }}
  {% endif %}
{% endfor %}
```

Let edit code as the flowing

```
from flask import Flask, render_template
@app.route("/web")
def frameworks():

      return render_template("index.html", frameworks=["Flask",
"Django"])
```

Django

Create a directory called templates / hello in the directory hello, then create index.html and copy content exactly like the Flask template.

Add the following function to hello / views.py

```
# Note that render is already imported in the view.py file
right after creating the app
def frameworks(request):

      return render(request, "hello/index.html", {"frameworks":
["Flask", "Django"]})
```

And add 1 line to project1 / urls.py:

```
 path('web', views.frameworks, name='frameworks'),
```

Also, have to add 'hello' to the file project1 / settings.py In the list of INSTALLED_APPS

```
INSTALLED_APPS = [
     ...,
     'hello',

 ]
```

Flask's default template in jinja2, which is the equivalent of the Django template, even though the syntax is very similar. Django has to add a line to the settings.py file, but this has to be done only once this should have been done after create-app started.

Django app

When running the initial “startproject” command, it generates the files needed for a Django project urls.py, settings.py, manager.py…, and it only has to run once in a project. you create folders manually.

Django introduces the concept of having multiple apps in one project. Flask has a similar concept, called Blueprint. It's not necessary for a 6-10 line example, but when making a feature-rich website blueprint is Flask's official organization, even in the Flask tutorial uses this concept. So only different learning before or after learning, not Flask is not used.

Flask does not force (does not instruct) the user to organize the code, when the project grows up, with 3-5000 lines of code, it must generate a new file, must be arranged and organized so that it is reasonable - but how If it is reasonable, not everyone can immediately find a way. The Flask project, when it falls into the hands of a programmer who does not have much experience in web development, will lead to code in one place, finding it tiring. Flask does not instruct the user in the first place, if encountered with someone with no organizational experience, it will become disorganized.

Django starts out by placing each in a set of places, which may seem a bit clunky at first, but tidy will pay off a clear benefit later. You can see 10 Django projects, the structure is the same, as Django's template, but 10 Flask projects each one is a different kind.

6. Conclusion about Flask & Django?

Flask can provide more benefits than other frameworks. However, there are several drawbacks. Flask, for example, is more vulnerable to security threats than systems such as Django. Furthermore, Flask may not be the ideal platform for quickly evolving complex applications. As a result, it is critical to choose the system with caution.

Flask is a reasonable choice if you need to, the application is clear and straightforward. A static web application, for example, might use Flask, but a dynamic program, such as eCommerce or a news site, would benefit from full-stack frameworks. Developers desire more influence of their system. There are not enough web developers on the squad. (Singh)

Django is a heavier platform than Flask — if you're new to web programming, it can be difficult to find out which components are responsible for what features and what you need to modify to get the desired results. However, once you've become used to Django, the extra work it does can be really helpful and save you time when it comes to setting up tedious, dull elements of a web application. (Dwyer)

It can be not easy to choose between the two systems at times. However, even as you get through their more sophisticated features, such as models, the two remain very close in many ways (many job advertisements ask for "Django or Flask experience" as a result). As a result, switching between the two is simple whether you ever need or want to. (Dwyer)

Work Cited

Amigos-Maker. “What Is Flask Used For?” *DEV Community*, 16 Nov. 2019, www.dev.to/amigosmaker/what-is-flask-used-for-2do5.

Bitsadmin. “Here Is Why We Are Big Fans of Python’s Flask Framework.” *Benchmark IT Solutions*, 2 June 2020, www.benchmarkit.solutions/blog/here-is-why-we-are-big-fans-of-pythons-flask-framework.

Dwyer, Gareth. “Flask vs. Django: Why Flask Might Be Better.” *Codementor*, 13 Feb. 2017, www.codementor.io/@garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v.

“Flask Tutorial - Tutorialspoint.” *Tutorialspoint*, www.tutorialspoint.com/flask. Accessed 16 Apr. 2021.

GeeksforGeeks. “Python | Introduction to Web Development Using Flask.” *GeeksforGeeks*, 18 May 2020, www.geeksforgeeks.org/python-introduction-to-web-development-using-flask.

“How to Structure a Flask Application.” *Damyan*, Adamyan.blog/post/flask-series-structure. Accessed 21 Apr. 2021.

Madeira, Tiago. “Why Use Python for Web Development?” *Blog | Imaginary Cloud*, 14 Jan. 2021, www.imaginarycloud.com/blog/why-use-python-for-web-development.

Singh, Vijay. “Flask vs Django in 2021: Which Framework to Choose?” *Hackr.Io*, www.hackr.io/blog/flask-vs-django. Accessed 16 Apr. 2021.