Build a Stock Market Web App in Flask

ABSTRACT

Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

What is Web Framework?

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

What is Flask?

Flask is a web application framework written in Python. It is developed by **Armin Ronacher**, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application. Some of the popular Flask extensions are discussed later in the tutorial.

Flask – Application

In order to test **Flask** installation, type the following code in the editor as **Hello.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return 'Hello World’

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

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.

Flask constructor takes the name of **current module (__name__)** as argument.

The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.

Sr. No.	Parameters & Description

1	**host** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally
2	**port** Defaults to 5000
3	**debug** Defaults to false. If set to true, provides a debug information
4	**options** To be forwarded to underlying Werkzeug server.

```
app.run(host, port, debug, options)
```

All parameters are optional

The above given **Python** script is executed from Python shell.

```
Python Hello.py
```

A message in Python shell informs you that

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open the above URL **(localhost:5000)** in the browser. **‘Hello World’** message will be displayed on it.

Debug mode

A **Flask** application is started by calling the **run()** method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable **debug support**. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.

The **Debug** mode is enabled by setting the **debug** property of the **application** object to **True** before running or passing the debug parameter to the **run()** method.

```
app.debug = True
app.run()
app.run(debug = True)
```

HTTP methods

Http protocol is the foundation of data communication in world wide web. Different methods of data retrieval from specified URL are defined in this protocol.

The following table summarizes different http methods −

Sr.No.	Methods & Description
1	**GET** Sends data in unencrypted form to the server. Most common method.
2	**HEAD** Same as GET, but without response body
3	**POST** Used to send HTML form data to server. Data received by POST method is not cached by server.
4	**PUT** Replaces all current representations of the target resource with the uploaded content.
5	**DELETE**

	Removes all current representations of the target resource given by a URL

By default, the Flask route responds to the **GET** requests. However, this preference can be altered by providing methods argument to **route()** decorator.

In order to demonstrate the use of **POST** method in URL routing, first let us create an HTML form and use the **POST** method to send form data to a URL.

Save the following script as login.html

```
<html>
<body>
<form action="" method="POST">
    <div class="columns">
        <div class="column is -9">
        <input class="input" type="text" name="ticker"
placeholder="Ticker(PNB.NS)">
        </div>
        <div class="column is-3">
            <button class="button is-primary">Search</button>
        </div>
    </div>
</form>
   </body>
</html>
```

Now enter the following script in Python shell.

```
from flask import Flask, request, render_template
from package import fetch

app = Flask(__name__)


@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        ticker = request.form["ticker"]
        info = fetch.company_info(ticker)
        return render_template("home.html", info=info)
    return render_template("home.html")


if __name__ == "__main__":
    app.run(port=5000)
```

After the development server starts running, open **home.html** in the browser, enter name in the text field and click **Search**.

Form data is POSTed to the URL in action clause of form tag.

http://localhost/login is mapped to the **login()** function. Since the server has received data by **POST** method, value of 'nm' parameter obtained from the form data is obtained by −

```
ticker = request.form['ticker']
```

It is passed to **'/success'** URL as variable part. The browser displays a **welcome** message in the window.

Change the method parameter to **'GET'** in **home.html** and open it again in the browser. The data received on server is by the **GET** method. The value of 'ticker' parameter is now obtained by −

```
ticker = request.args.get('ticker')
```

Here, **args** is dictionary object containing a list of pairs of form parameter and its corresponding value. The value corresponding to 'ticker' parameter is passed on to '/success' URL as before.

Programming

Given below is the Python code of main.py-

```
from flask import Flask, request, render_template
from package import fetch

app = Flask(__name__)


@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        ticker = request.form["ticker"]
        info = fetch.company_info(ticker)
        return render_template("home.html", info=info)
    return render_template("home.html")


if __name__ == "__main__":
    app.run(port=5000)
```

Given below is to write another Python code of fetch.py-

```
import yfinance as yf


def company_info(ticker):
    ticker = yf.Ticker(ticker)
    return ticker.info
```

Code of template **(index.html)** is given below –

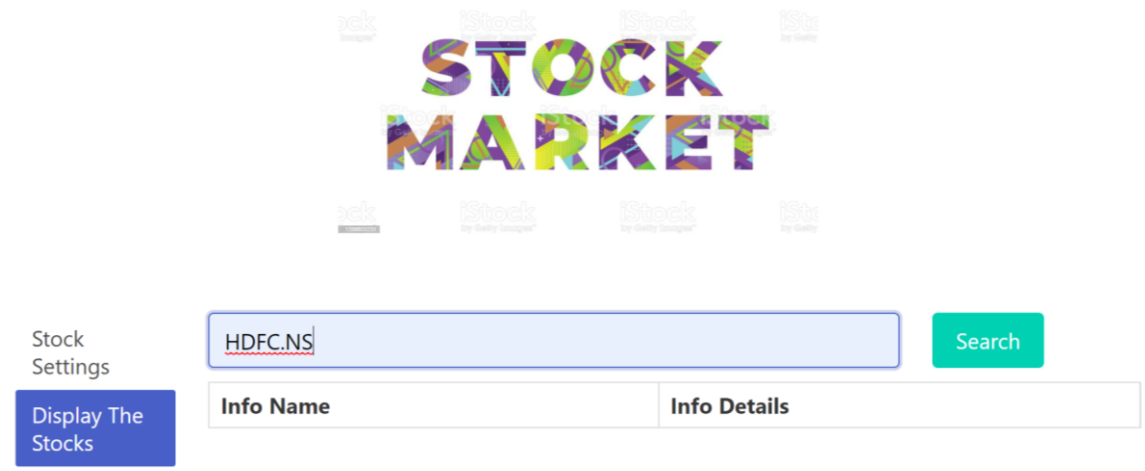
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Stock Market</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bulma@0.9.3/css/bulma.min.css">
  </head>
  <body>
  <p style="text-align:center;"><img
src="https://media.istockphoto.com/vectors/stock-market-concept-retro-
colorful-word-art-illustration-vector-id1288625231" alt="stock-market"
height="120" width="350"></p>
  <section class="section">
    <div class="columns">
      <div class="column is-2">
        <aside class="menu">
  <ul class="menu-list">
    <li><a>Stock Settings</a></li>
```

```
    <li>
      <a class="is-active">Display The Stocks</a>
    </li>
  </ul>

        </aside>
      </div>
      <div class="column">
        {%block body%}
        {%endblock%}
      </div>
    </div>
  </section>
  </body>
</html>
```

Code of template **(home.html)** is given below –

```
{%extends 'index.html'%}

{%block body%}
<form action="" method="POST">
    <div class="columns">
        <div class="column is -9">
        <input class="input" type="text" name="ticker"
placeholder="Ticker(PNB.NS)">
        </div>
        <div class="column is-3">
            <button class="button is-primary">Search</button>
        </div>
    </div>
</form>
<div class="container" style="margin-top:10px;">
    <table class="table is-bordered is-striped is-narrow is-hoverable is-
fullwidth">
        <thead>
        <th>Info Name</th>
        <th>Info Details</th>
        </thead>
        <tbody>
        {% for i in info%}
        <tr>
            <td>
                {{i}}
            </td>
            <td>
                {{info[i]}}
            </td>
        </tr>
        {%endfor%}
        </tbody>
    </table>
</div>
{%endblock%}
```

Run the Python script and enter the URL **http://localhost:5000/** in the browser



When the **Search** button is clicked, form data is rendered on **home.html** in the form of HTML table.

Conclusion

Flask may have more advantages than other frameworks. However, there are downsides to it too. For example, Flask is more vulnerable to security risks as compared to frameworks such as Django. Also, Flask may not be the right framework to use for developing complex apps at a fast pace. So, it’s important to choose the framework carefully.