Stock Monitor With Redis Flask

Abstract

We will use flask and redis for this. Flask is a good python web micro framework which lets you focus only on things you need. There is more focus on the modularity of your code base. Redis is a key-value data store that can be used as a database. Redis is an excellent choice for caching and for constant real-time analysis of data coming in, hence redis is great tool to build a platform.

What is Redis?

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server, since the keys can contain strings, hashes, lists, sets and sorted sets. Redis is written in C. This tutorial provides good understanding on Redis concepts, needed to create and deploy a highly scalable and performance-oriented system.

What is Flask Python?

Flask is a web framework, it's a Python module that lets you develop web applications easily. it's a microframework that doesn't include an ORM (Object Relational Manager) or such features. It does have many cool features like url routing, template engine. It is a WSGI web app framework.

What is web framework?

A Web Application Framework or a simply a Web Framework represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on.

What is Flask?

Flask is a web application framework written in Python. It was developed by Armin Ronacher. Flask is based on the Werkzeug WSGI toolkit and the Jinja2 template. Flask is often referred to as a micro framework. It is designed to keep the core of the application simple and scalable. Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.

WSGI:

The Web Server Gateway Interface (Web Server Gateway Interface, WSGI) has been used as a standard for Python web application development. WSGI is the specification of a common interface between web servers and web applications.

Werkzeug:

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeug as one of its bases.

Jinja2:

jinja2 is a popular template engine for Python. A web template system combines a template with a specific data source to render a dynamic web page. This allows you to pass Python variables into HTML templates Lets create a stock market web app!

Steps in Creating a Stock Market Web app:

Step1: Installing Flask

Step2: Creating a base app

Step3: Using HTML templates

Step4: installing redis,redis-om,redis modules

Step5: Setting up redis database for stocks

Step6: Displaying the stock items with time series.

Step1: Installing Flask:

In this step, you'll activate your Python environment and install Flask using the pip package installer. Syntax:

```
pip install flask
```

Step2: Creating a base application:

In this step, you'll make a small web application inside a Python file and run it to start the server, which will display some information on the browser.

Code:

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

#from flask import Flask Flask constructor takes the name of current module (__name__) as argument.#

app = Flask(__name__)

# The route() function of the Flask class is a decorator,

# which tells the application which URL should call the associated function.

@app.route('/')
```

```
# ‘/’ URL is bound with hello_world() function.
def hello_world():
    return 'Hello World'
# main driver function
if __name__ == '__main__':
 # run() method of Flask class runs the application on the local development server.
 app.run()
```

Output:

```
 * Serving Flask app "hello" (lazy loading)
* Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
* Debugger is active!
* Debugger PIN: 813-894-335
```

The application is running locally on the URL http://127.0.0.1:5000/, 127.0.0.1 is the IP that represents your machine’s localhost and :5000 is the port number.

Open a browser and type in the URL http://127.0.0.1:5000/, you will receive the string Hello, World! as a response, this confirms that your application is successfully running.

Step3:Using of HTML Templates:

Currently your application only displays a simple message without any HTML. Web applications mainly use HTML to display information for the visitor.

Flask provides a render_template() helper function that allows use of the Jinja template engine. This will make managing HTML much easier by writing your HTML code in .html files as well as using logic in your HTML code. You’ll use these HTML files, (templates) to build all of your application pages, such as the main page where you’ll display the current blog posts, the page of the stocks, the page where the user can add a new stocks, and so on.

Code:

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
```

```
def index():
    return render_template('base.html')
```

• The index() view function returns the result of calling render_template() with index.html as an argument, this tells render_template() to look for a file called index.html in the templates folder. Both the folder and the file do not yet exist, you will get an error if you were to run the application at this point.

• So before run the application you should create templates folder and HTML file.

• In addition to the templates folder, Flask web applications also typically have a static folder for hosting static files, such as CSS files, JavaScript files, and images the application uses.

• You can create a style.css style sheet file to add CSS to your application. First, create a directory called static inside your main directory.

```
<!doctype html>
<html lang="en">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
    <title>
            {% block title %}

            {% endblock %}
    </title>
    </head>
<body>
    <div class="navbar">
    <a  href="{{url_for('home')}}">Home</a>
    <a  href="{{url_for('stocks_page')}}">Stocks</a>
    </div>
        {% block content %}

        {% endblock %}
</body>
</html>
```

However, the following highlighted parts are specific to the Jinja template engine:

• {% block title %} {% endblock %}: A block that serves as a placeholder for a title, you'll later use it in other templates to give a custom title for each page in your application without rewriting the entire section each time.

• {{ url_for('index')}}: A function call that will return the URL for the index() view function. This is different from the past url_for() call you used to link a static CSS file, because it only takes one argument, which is the view function's name, and links to the route associated with the function instead of a static file.

• {% block content %} {% endblock %}: Another block that will be replaced by content depending on the child template (templates that inherit from base.html) that will override it.

• Using Template inheritance you will avoid typing the same code again and again

Step4:Installing redis,redis_om:

Start by installing the extension with **pip install flask-redis** Once that's done, configure it within your Flask config.

Usage

Setup

To add a Redis client to your application:

import redis

from flask import Flask, render_template

db = redis.StrictRedis(host='127.0.0.1', port='6379', db=0, charset="utf-8", decode_responses=True)

Then install redis-om with pip install redis _om to create python data models to store stock items.

Step5: Creating data models:

then create data model to store stock items and display the stock list on the app.

```
from typing import Optional,List
import redis
from pydantic import EmailStr, ValidationError,PositiveInt
from redis_om import JsonModel,Field
import json


class Item(JsonModel):

    name :str=Field(index=True)
    price : PositiveInt=Field(index=True)
    barcode :str=Field(index=True)
    description :str=Field(index=True)
```

Store the stocks as json elements in stocks.json with the following code

[{“name”:”HDFC”,”price”:500,”barcode”:”123123123123”,”description”:”none”,
“name”:”SBI”,”price”:1000,”barcode”:”152152152152”,”description”:”none”,
“name”:”tcs”,”price”:2000,”barcode”:”1010001000101”,”description”:”none”,}]

Step6:Displaying the stocks with time series:

```
@app.route("/stocks")
def stocks_page():

   item=Item.find().all()

   return render_template('stocks.html', items=item)
```

output:

run your **main.py** file and click on the link that it provides after running. If it doesn’t show you any link, open your browser and on the url box, type http://0.0.0.0:5000/.

While routing to stocks page, I am getting an error,

:

RedisModelError

```
redis_om.model.model.RedisModelError: Your Redis instance does not have either the RediSearch module or RedisJSON module installed. Querying requires that your Redis instance has one of these modules installed.
```

Traceback (most recent call last)

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *2095*, in `__call__`

```
return self.wsgi_app(environ, start_response)
```

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *2080*, in `wsgi_app`

```
response = self.handle_exception(e)
```

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *2077*, in `wsgi_app`

```
response = self.full_dispatch_request()
```

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *1525*, in `full_dispatch_request`

```
rv = self.handle_user_exception(e)
```

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *1523*, in `full_dispatch_request`

```
rv = self.dispatch_request()
```

File "C:\Users\sugum\AppData\Roaming\Python\Python310\site-packages\flask\app.py", line *1509*, in `dispatch_request`

```
return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
```

File "d:\app\main.py", line *37*, in `stocks_page`

```
item=Item.find().all()
```