Create Cricket Score API using Web Scraping in Flask

Last Updated : 20 Mar, 2024

Cricket is one of the famous outdoor sport played worldwide. There are very few APIs providing live scoreboards and none of them are free to use. Using any of the scoreboards available we can create API for ourselves. This method not only works for Cricket Scoreboard but also for any information available online. Following is the flow in which this blog would guide to create an API and deploy it.

- Setting up the App Directory
- Web Scrape data from NDTV Sports.
    - Beautiful Soup in Python would be used.
- Create an API.
    - Flask would be used.
- Heroku would be used for deployment,

Setting up the App Directory

Step 1: Create a Folder (eg. CricGFG).

Step 2: Set up the virtual environment. Here we create an environment **.env**

```
python -m venv .env
```

Step 3: Activate the environment.

```
.env\Scripts\activate
```

Getting the Data

Step 1: In Python, we have Beautiful Soup which is a library to pull out data from HTML files. To install Beautiful Soup, run a simple command;

```
pip install beautifulsoup4
```

```
(.env) D:\CricGFG>pip install beautifulsoup4
Collecting beautifulsoup4
  Using cached beautifulsoup4-4.9.3-py3-none-any.whl (115 kB)
Collecting soupsieve>1.2; python_version >= "3.0"
  Using cached soupsieve-2.2.1-py3-none-any.whl (33 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.9.3 soupsieve-2.2.1
WARNING: You are using pip version 20.2.3; however, version 21.2.4 is available.
You should consider upgrading via the 'd:\cricgfg\.env\scripts\python.exe -m pip install --upgrade pip' command.

(.env) D:\CricGFG>
```

Similarly, install the Requests module of Python.

```
pip install requests
```

We would use the NDTV Sports Cricket Scorecard to fetch the data.

Step 3: Following are the steps for Scraping data from the Web Page. To get the HTML text from the web page;

html_text = requests.get('https://sports.ndtv.com/cricket/live-scores').text

To represent the parsed object as a whole we use the BeautifulSoup object,

```
soup = BeautifulSoup(html_text, "html.parser")
```

Note: It is recommended to run and check the code after each step to know about the difference and thoroughly understand the concepts.

Python

```python
from bs4 import BeautifulSoup
import requests

html_text = requests.get('https://sports.ndtv.com/cricket/live-scores').text
soup = BeautifulSoup(html_text, "html.parser")
print(soup)
```

We will further find all the required divs and other tags with their respective classes.

Python

```
from bs4 import BeautifulSoup
import requests

html_text = requests.get('https://sports.ndtv.com/cricket/live-scores').text
soup = BeautifulSoup(html_text, "html.parser")
sect = soup.find_all('div', class_='sp-scr_wrp')
section = sect[0]
description = section.find('span', class_='description').text
location = section.find('span', class_='location').text
current = section.find('div', class_='scr_dt-red').text
link = "https://sports.ndtv.com/" + \
    section.find('a', class_='scr_ful-sbr-txt').get('href')
```

The next section of the code has our data that is our result. If for any of the reasons that code is not present in the HTML file, it would lead to an error, so including that part in a try and except block.

Complete Code:

Python3

```
from bs4 import BeautifulSoup
import requests

html_text = requests.get('https://sports.ndtv.com/cricket/live-scores').text
soup = BeautifulSoup(html_text, "html.parser")
sect = soup.find_all('div', class_='sp-scr_wrp ind-hig_crd vevent')

section = sect[0]
description = section.find('span', class_='description').text
location = section.find('span', class_='location').text
current = section.find('div', class_='scr_dt-red').text
link = "https://sports.ndtv.com/" + section.find(
    'a', class_='scr_ful-sbr-txt').get('href')

try:
    status = section.find_all('div', class_="scr_dt-red")[1].text
```

```
        block = section.find_all('div', class_='scr_tm-wrp')
        team1_block = block[0]
        team1_name = team1_block.find('div', class_='scr_tm-nm').text
        team1_score = team1_block.find('span', class_='scr_tm-run').text
        team2_block = block[1]
        team2_name = team2_block.find('div', class_='scr_tm-nm').text
        team2_score = team2_block.find('span', class_='scr_tm-run').text
        print(description)
        print(location)
        print(status)
        print(current)
        print(team1_name.strip())
        print(team1_score.strip())
        print(team2_name.strip())
        print(team2_score.strip())
        print(link)
    except:
        print("Data not available")
```

Output:

Live score England vs India 3rd Test,Pataudi Trophy, 2021

Headingley, Leeds

England lead by 223 runs

Day 2 | Post Tea Session

England

301/3 (96.0)

India

78

https://sports.ndtv.com//cricket/live-scorecard/england-vs-india-3rd-test-leeds-enin08252021199051

Creating the API

We will use Flask which is a micro web framework written in Python.

```
pip install Flask
```

Following is the starter code for our flask application.

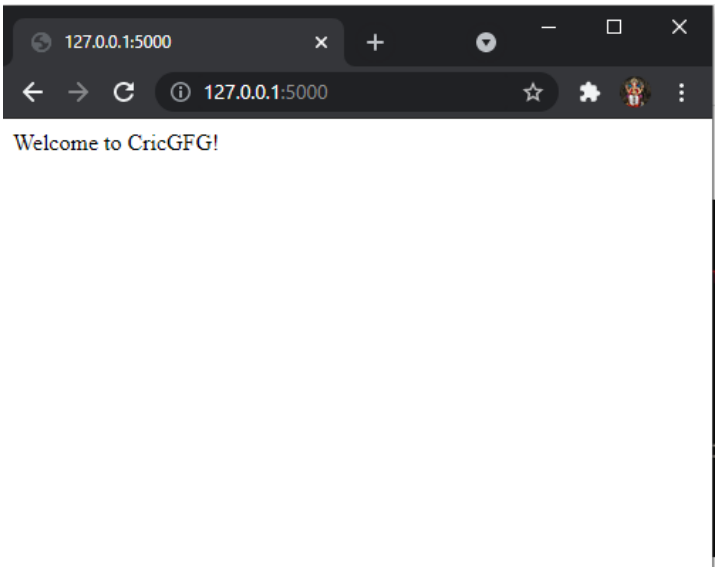
Python3

```python
# We import the Flask Class, an instance of
# this class will be our WSGI application.
from flask import Flask

# We create an instance of this class. The first
# argument is the name of the application’s module
# or package. __name__ is a convenient shortcut for
# this that is appropriate for most cases.This is
# needed so that Flask knows where to look for resources
# such as templates and static files.
app = Flask(__name__)

# We use the route() decorator to tell Flask what URL
# should trigger our function.
@app.route('/')
def cricgfg():
    return "Welcome to CricGFG!"

# main driver function
if __name__ == "__main__":

    # run() method of Flask class runs the
    # application on the local development server.
    app.run(debug=True)
```

Output:

```
 * Serving Flask app "CricGFG" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 328-502-483
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open localhost on your browser:



We would now add our code of Web Scraping into this and some helper methods provided by Flask to properly return JSON data.

Understanding Jsonify

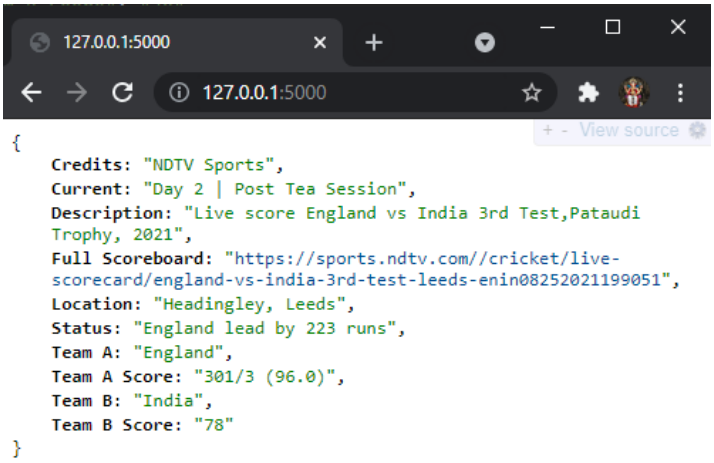
jsonify is a function in Flask. It serializes data to JavaScript Object Notation (JSON) format. Consider the following code:

Python3

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def cricgfg():

    # Creating a dictionary with data to test jsonfiy.
    result = {
        "Description": "Live score England vs India 3rd Test,Pataudi \
```

```
        Trophy, 2021",
        "Location": "Headingley, Leeds",
        "Status": "England lead by 223 runs",
        "Current": "Day 2 | Post Tea Session",
        "Team A": "England",
        "Team A Score": "301/3 (96.0)",
        "Team B": "India",
        "Team B Score": "78",
        "Full Scoreboard": "https://sports.ndtv.com//cricket/live-scorecard\
        /england-vs-india-3rd-test-leeds-enin08252021199051",
        "Credits": "NDTV Sports"
    }
    return jsonify(result)

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

Output:



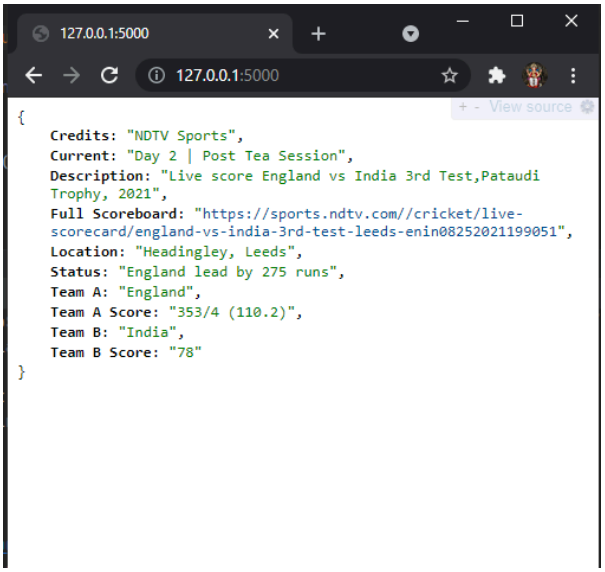
Now it's time to merge all our codes. Let's Start!

Python3

```
import requests
from bs4 import BeautifulSoup
from flask import Flask, jsonify

app = Flask(__name__)


@app.route('/')
def cricgfg():
```

```
    html_text = requests.get('https://sports.ndtv.com/cricket/live-scores').text
    soup = BeautifulSoup(html_text, "html.parser")
    sect = soup.find_all('div', class_='sp-scr_wrp ind-hig_crd vevent')

    section = sect[0]
    description = section.find('span', class_='description').text
    location = section.find('span', class_='location').text
    current = section.find('div', class_='scr_dt-red').text
    link = "https://sports.ndtv.com/" + section.find(
    'a', class_='scr_ful-sbr-txt').get('href')

    try:
        status = section.find_all('div', class_="scr_dt-red")[1].text
        block = section.find_all('div', class_='scr_tm-wrp')
        team1_block = block[0]
        team1_name = team1_block.find('div', class_='scr_tm-nm').text
        team1_score = team1_block.find('span', class_='scr_tm-run').text
        team2_block = block[1]
        team2_name = team2_block.find('div', class_='scr_tm-nm').text
        team2_score = team2_block.find('span', class_='scr_tm-run').text
        result = {
            "Description": description,
            "Location": location,
            "Status": status,
            "Current": current,
            "Team A": team1_name,
            "Team A Score": team1_score,
            "Team B": team2_name,
            "Team B Score": team2_score,
            "Full Scoreboard": link,
            "Credits": "NDTV Sports"
        }
    except:
        pass
    return jsonify(result)

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

Output in the Browser:



Here we have created our own Cricket API.

Deploying API on Heroku

Step 1: You need to create an account on Heroku.

Step 2: Install Git on your machine.

Step 3: Install Heroku on your machine.

Step 4: Login to your Heroku Account

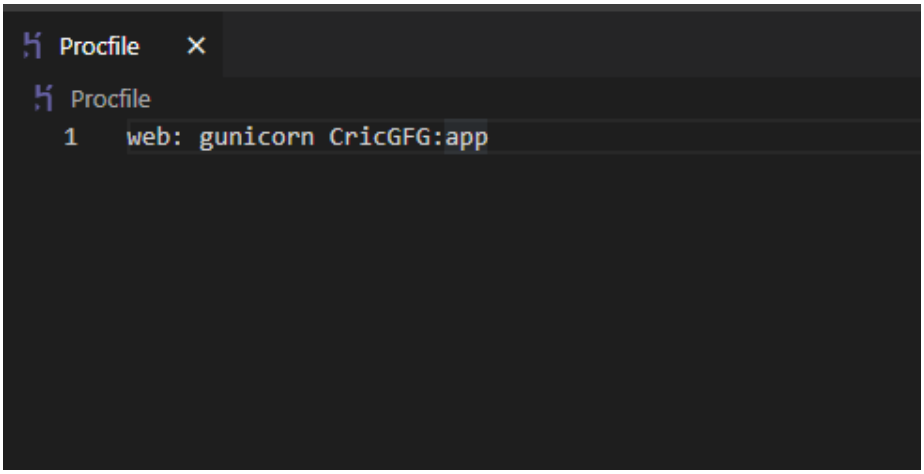
```
heroku login
```

Step 5: Install gunicorn which is a pure-Python HTTP server for WSGI applications. It allows you to run any Python application concurrently by running multiple Python processes.

```
pip install gunicorn
```

Step 6: We need to create a profile which is a text file in the root directory of our application, to explicitly declare what command should be executed to start our app.

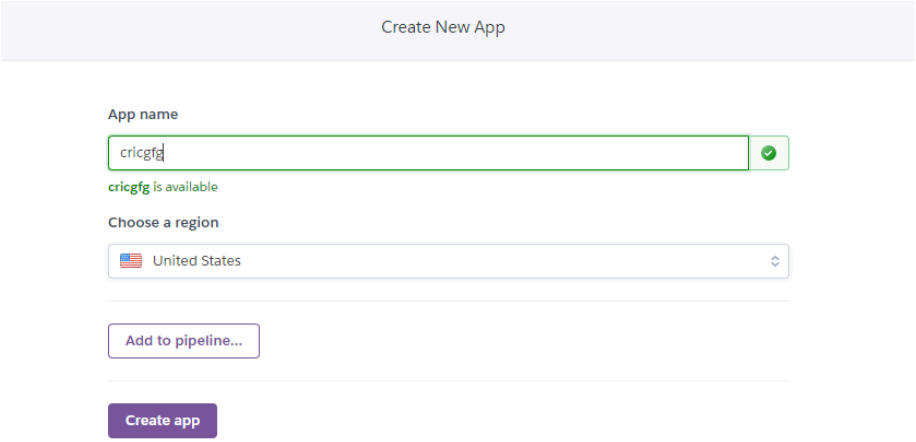
```
web: gunicorn CricGFG:app
```



Step 7: We further create a requirements.txt file that includes all the necessary modules which Heroku needs to run our flask application.

```
pip freeze >> requirements.txt
```

Step 8: Create an app on Heroku, click here.



Step 9: We now initialize a git repository and add our files to it.

```
git init
git add .
git commit -m "Cricket API Completed"
```

```
(.env) D:\CricGFG>git init
Initialized empty Git repository in D:/CricGFG/.git/

(.env) D:\CricGFG>git add .

(.env) D:\CricGFG>git commit -m "Cricket API Completed"
[master (root-commit) faf5ae1] Cricket API Completed
 4 files changed, 64 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 CricGFG.py
 create mode 100644 Procfile
 create mode 100644 requirements.txt
```

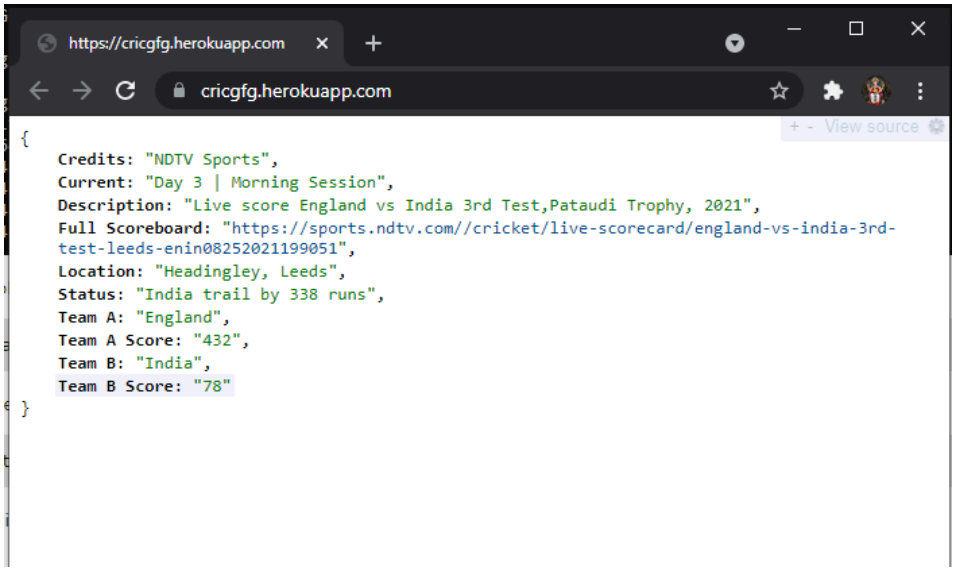
Step 10: We will now direct Heroku towards our git repository.

```
heroku git:remote -a cricgfg
```

Step 11: We will now push our files on Heroku.

```
git push heroku master
```

Finally, our API is now available on https://cricgfg.herokuapp.com/



Looking to dive into the world of programming or sharpen your Python skills? Our **Master Python: Complete Beginner to Advanced Course** is your ultimate guide to becoming proficient in Python. This course covers everything you need to build a solid foundation from fundamental programming concepts to advanced techniques. With **hands-on projects**, real-world examples, and expert guidance, you'll gain the confidence to tackle complex **coding challenges**. Whether you're starting from scratch or aiming to enhance your skills, this course is the perfect fit. Enroll now and master Python, the language of the future!

P kotha... 5

Previous Article

Single Page Portfolio Using Flask

Next Article

Create a Bar Chart From a DataFrame with Plotly and Flask

Similar Reads

Web Scraping using Beautifulsoup and scrapingdog API

In this post we are going to scrape dynamic websites that use JavaScript libraries like React.js, Vue.js, Angular.js, etc you have to put extra efforts. It is an easy but...

5 min read

Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses...

4 min read

Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would...

12 min read

How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application.The Session is the time between the client logs in to...

4 min read

How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is know...

8 min read

Flask URL Helper Function - Flask url_for()

In this article, we are going to learn about the flask url_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used f...

11 min read

Flask API Authentication with JSON Web Tokens

Authentication is the process of verifying the identity of the user. It checks whether the user is real or not. It is used to provide access to resources only to...

7 min read

Create GitHub API to fetch user profile image and number of repositories...

GitHub is where developers shape the future of software, together, contribute to the open-source community, manage Git repositories, etc. It is one of the most...

5 min read

Newspaper scraping using Python and News API

There are mainly two ways to extract data from a website: Use the API of the website (if it exists). For example, Facebook has the Facebook Graph API which...

4 min read

GUI Application for Live Cricket scoreboard Using Python

In this article, we will see how sports.py module is imported and implemented to produce scoreboard of a specified sport like baseball, basketball, cricket and...

3 min read

Article Tags : Blogathon Python Blogathon-2021 Flask Projects +2 More

Practice Tags : python