

بسمه تعالی

پروژه: تست کاراشاب

مجری: مرضیه لطیفی

Backend:

Main.py

۱- در قسمت backend ابتدا یک دیتابیس از نوع sqlite3 با نام memdb ایجاد و در آن جدولی به نام memory تعریف کردیم تا مقادیر حافظه RAM در آن ذخیره شوند:

```
۱- conn=sqlite3.connect('memdb')
۲- cursor=conn.cursor()
۳- cursor.execute("""CREATE TABLE IF NOT EXISTS memory(
۴-     timestamp INT,
۵-     total INT,
۶-     available INT,
۷-     used INT
۸- )""")
۹- conn.commit()
۱۰- conn.close()
```

نکته: باید توجه داشت برای انجام این کار ابتدا باید کتابخانه sqlite3 را نصب و در برنامه با دستور زیر فراخوانی کرد:

```
import sqlite3
```

۲- در ادامه با استفاده از حلقه while عملیات خواندن مقادیر حافظه و قرار دادن آن در دیتابیس را انجام و این عملیات را در هر دقیقه یکبار تکرار می کنیم. برای خواندن مقادیر حافظه RAM از کتابخانه psutil استفاده میکنیم. psutil یک کتابخانه بین پلتفرمی برای بازیابی اطلاعات برای فرآیندهای در حال اجرا و استفاده از سیستم (CPU، حافظه، دیسک، شبکه، حسگرها) در پایتون است. psutil در حال حاضر از پلتفرم های FreeBSD, OpenBSD, NetBSD ، Windows، Linux ، macOS Sun Solaris و AIX پشتیبانی می کند.

```
timestamp=int(time.time())
svmem =psutil.virtual_memory()
total= svmem.total//1000000
available= svmem.available//1000000
used= svmem.used//1000000
conn=sqlite3.connect('memdb')
cursor=conn.cursor()
cursor.execute("INSERT INTO memory VALUES(?, ?, ?, ?);", (timestamp,
total, available, used))
conn.commit()
conn.close()
time.sleep(60)
```

در این قسمت از کد برنامه، با استفاده از `psutil.virtual_memory()` به حافظه دسترسی پیدا کرده و با `svmem.total` ، `svmem.available` و `svmem.used` به مقادیر `total` ، `available` و `used` مدنظر پروژه می‌رسیم. مقادیر بدست آمده را بر `۱۰۰۰۰۰۰` تقسیم میکنیم تا اعداد با واحد مگابایت ذخیره گردند.

باید توجه داشت که همانند مرحله قبل، ابتدا کتابخانه را نصب و سپس در پروژه استفاده می‌کنیم:

```
import psutil
```

کتابخانه دیگری که در این قسمت از برنامه مورد استفاده قرار دادیم کتابخانه `time` است که زمان سیستم را با متد `time` برمیگرداند و به این ترتیب زمان سیستم را هم در دیتابیس می‌توان ذخیره نمود:

```
import time
```

```
timestamp=int(time.time())
```

۳-در نهایت با دستور زیر به دیتابیس وصل شده و مقادیر بدست آمده از مرحله قبل را در دیتابیس ذخیره میکنیم:

```
conn=sqlite3.connect('memdb')
cursor=conn.cursor()
cursor.execute("INSERT INTO memory VALUES(?, ?, ?, ?);", (timestamp,
total, available, used))
conn.commit()
conn.close()
```

۴- و مراحل ۲ و ۳ را هر دقیقه یکبار تکرار میکنیم:

```
time.sleep(60)
```

API:

App.py

برای ایجاد API مورد نظر پروژه از فریمورک flask استفاده نمودیم؛ فریمورک را نصب و در پروژه با دستور زیر مورد استفاده قرار دادیم:

```
from flask import Flask , jsonify
```

برای قسمت routing از متد GET و پارامتر `</int:num>` استفاده کردیم که کاربر با وارد کردن عدد دلخواه `n` در آدرس، `n` مقدار خروجی دریافت کند.

```
@app.route('/<int:num>', methods = ['GET'])
```

تابع اصلی در این قسمت تابع `index(num)` است که با دریافت مقدار `num` به عنوان پارامتر ورودی، دستورات را اجرا میکند:

```
def index(num):  
    #connect to database(memdb) and fetch all result rows  
    conn=sqlite3.connect('memdb')  
    query = conn.execute('SELECT * FROM memory')  
    data=query.fetchall()  
    m=[]  
    if num<len(data):  
        b=len(data)+1  
        a=len(data)-num  
    #add num rows in m and return m  
    for itm in data[a:b]:  
        m.append({  
            'Time': itm[0],  
            'Total':itm[1] ,  
            'available':itm[2],  
            'used':itm[3]  
        })  
    return jsonify(m)  
else:  
    return('The number is too large!')
```

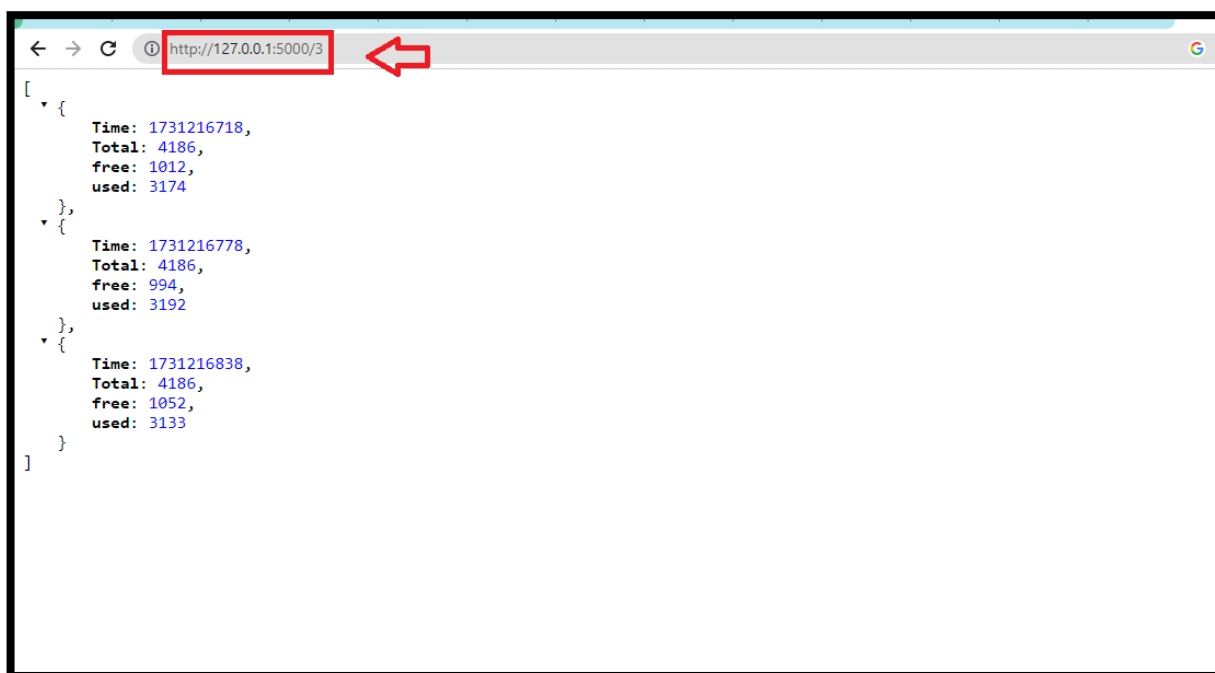
در این تابع ابتدا یک اتصال به دیتابیس `memdb` برقرار میکنیم؛ سپس تمام سطرهای موجود در جدول `memory` را با دستورات زیر واکنشی میکنیم:

```
query = conn.execute('SELECT * FROM memory')  
data=query.fetchall()
```

در ادامه باید بررسی کنیم که آیا در مدت زمان اجرای پروژه، به تعداد `num` ارسالی توسط کاربر، سطر در جدول دیتابیس وجود دارد یا خیر.

اگر شرط $\text{num} \leq \text{len}(\text{data})$ برقرار بود با اضافه کردن تمام سطرهاى موجود در بازه a تا b به لیست خالى m، و jsonify کردن تمام اعضاى m به خروجى json مورد نظر مى رسیم.

در غیر اینصورت پیام "the number is too large!" را چاپ مى کنیم تا کاربر متوجه شود num وارد شده از تعداد سطرهاى دیتابیس بیشتر است. خروجى به دست آمده برای مقدار $\text{num}=3$ و $\text{num}=5$ بعد از اجرای app به صورت زیر است:



```
[
  {
    Time: 1731216718,
    Total: 4186,
    free: 1012,
    used: 3174
  },
  {
    Time: 1731216778,
    Total: 4186,
    free: 994,
    used: 3192
  },
  {
    Time: 1731216838,
    Total: 4186,
    free: 1052,
    used: 3133
  }
]
```



```
[
  {
    Time: 1731216597,
    Total: 4186,
    free: 1042,
    used: 3143
  },
  {
    Time: 1731216658,
    Total: 4186,
    free: 1045,
    used: 3140
  },
  {
    Time: 1731216718,
    Total: 4186,
    free: 1012,
    used: 3174
  },
  {
    Time: 1731216778,
    Total: 4186,
    free: 994,
    used: 3192
  },
  {
    Time: 1731216838,
    Total: 4186,
    free: 1052,
    used: 3133
  }
]
```