Flow Monitoring

1 Problem and Usage Scenarios

1.1 Background

Traffic statistics play a very important role in social life, mainly reflected in public safety management, statistical analysis of commercial market data and reasonable allocation of resources and other aspects. For many large places with dense crowds, the flow statistics technology can provide an important basis for managers to manage and make decisions in public places. For example, security risks can be analyzed in public places. It can be used to analyze market supply and demand in business analysis. Therefore, real-time and effective statistics of people flow are of great significance in many social fields.

In railway stations, airports and other places with dense passenger flow, through the statistics of passenger flow data, security personnel in each region can be appropriately allocated to prevent the occurrence of accidents; Through the statistics of the flow of people in shopping malls and supermarkets, it can avoid the accidents caused by the large population density, and can provide effective basis for site selection, so as to win more interests; Through the statistics of the flow of people in the library and other places, students can understand the current situation of the library through the client, and teachers can manage the library more reasonably and effectively according to the flow of people, so as to provide students with a more convenient and more humanized learning conditions.

In real life, many fields will take the method of manual operation to count the flow of people, in order to obtain accurate real-time flow statistics. However, this manual counting method has many disadvantages: tedious work, waste of time manpower and material resources. In real life, this kind of method can not achieve the effectiveness, real-time and systematic counting of people flow, and it is easy to produce wrong statistical data when people flow is dense.

1.2 Introduction of the System

Based on this background, compared with the manual counting method, the Flow Monitoring system based on the Internet of Things and AWS designed by our team can save a lot of manpower and financial resources, and can run stably for a long time, avoiding the unreliable statistical data caused by manual operation under the state of overwork.

Flow Monitoring can be used in high-traffic areas such as shopping malls, supermarkets, train stations, airports, hotels, schools, and so on. Flow Monitoring provides visitor traffic statistics for these locations. Specifically, by deploying sensors on Doors or gates, the system counts the time visitors pass through the entrances and provides administrators of these locations with a website that allows them to view visitor traffic data collected by Flow Monitoring.

In this report, the design, implementation, challenges and other relevant instructions of the system will be introduced in detail.

2 Design

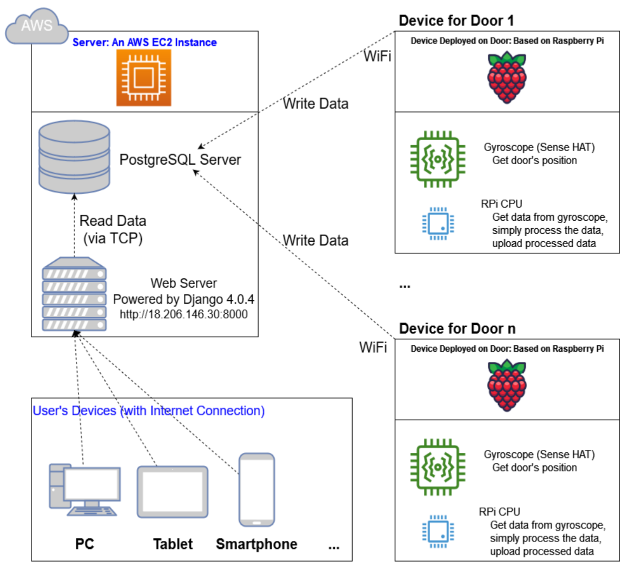
2.1 a "Thing" in the Internet of Things

Thing	Door
Connectivity	WLAN
Sensing (sensors)	Gyroscope on Raspberry Pi Sense HAT
Data Collection	Record details of a person passing the door. The data contains name of the door, passing date and time.
Acting (actuator)	Speaker, connected on RPi.
Communicating	Establish TCP communication with the database through the Psycopg2 library using a wireless network.
Computing	Judge whether the Angle change of the current door reaches a certain value (50 degrees), if the Angle change exceeds this value, it is considered that someone passes the door; When someone comes through, the speaker makes a short prompt sound and gets the current exact time uploaded to the database

2.2 the IoT architecture

Sensing Layer	The system senses the attitude of the door at different times through the gyroscope fixed on the door, that is, its Angle in the coordinate system of PITCH, roll and YAW.
Network Layer	As a communication channel, the network layer transmits the data collected in the perception layer to other connected devices, such as the cloud database. In fact, considering the future scalability of the system and the reduction of database write pressure, the collected data is not directly uploaded to the database, but is simply filtered and processed by the RPi before uploading to the database.
Data Processing Layer	The data processing layer is divided into two parts. In the first part, data collected from sensors are simply screened and sorted by RPi, and then uploaded to the cloud database through Wi-Fi. The second part is done by servers in the cloud. In this step, the server retrieves data uploaded by the RPi from the database, analyzes the data and forms a line chart of visitor numbers on a daily basis.
Application Layer	The application layer is divided into two parts. The first part relies on THE RPi implementation, which is mainly used to help deploy the RPi device on the door. A prompt is played when the person installing the device starts and sets up the program on the RPi. When a visitor passes through the door, the loudspeaker connected to the RPi will sound a prompt tone for maintenance personnel to detect the stopped RPi in time. The second part is the web page designed and implemented by our team. Users can use a browser on any device, such as a smartphone or PC, or even a Kindle, to access real-time visitor traffic statistics at any time of day on the site.

2.3 Infrastructure


AWS
Server: An AWS EC2 Instance
PostgreSQL Server
Read Data (via TCP)
Web Server Powered by Django 4.0.4 http://18.206.146.30:8000
Write Data
Write Data
WiFi
WiFi
Device for Door 1
Device Deployed on Door: Based on Raspberry Pi
Gyroscope (Sense HAT) Get door's position
RPi CPU Get data from gyroscope, simply process the data, upload processed data
...
Device for Door n
Device Deployed on Door: Based on Raspberry Pi
Gyroscope (Sense HAT) Get door's position
RPi CPU Get data from gyroscope, simply process the data, upload processed data
User's Devices (with Internet Connection)
PC
Tablet
Smartphone
...

2.4 Techniques

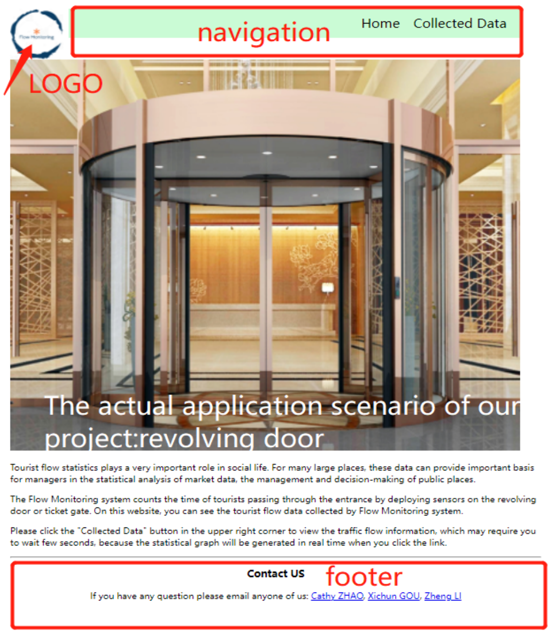
PostgreSQL	PostgreSQL is a free, open source database system known for its stability, ease of maintenance, and low performance under high load. PostgreSQL has grown in popularity over the past decade, becoming the fourth most popular database system after MySQL, Oracle and SQL Server, Apple, IMDB (the Internet Movie Database), Instagram, Skype, and more all use it. We chose this database system because Flow Monitoring is a project that involves a lot of reads and writes to the database (especially writes, think of commuting subway stations), and the database needs to be stable enough to minimize performance degradation under high loads.
AWS EC2	EC2 is a virtual machine service provided by AWS and has very flexible application scenarios. We chose this AWS service because it's easier to use and has richer and more complete functionality than Elastic Beanstalk, even though it's a bit of a performance waste.
Amazon CloudWatch	Cloud Watch provides detailed operating data of the server (i.e. EC2 instance), which is very intuitive and convenient for monitoring whether the server is running well or not. We can use it to obtain some performance indicators of the project.
django	Django is an open source Web application framework based on Python that is easy, fast and secure to develop. It can also be deployed on virtual machines. Since our site only needs to display data, it's very convenient to just call Django views. and process data.

3 Solution Implementation

3.1 Web Implementation

Web pages:

The following two figures show the details of Home page and Collected data page.


navigation
Home
Collected Data
LOGO
The actual application scenario of our project:revolving door
footer

Tourist flow statistics plays a very important role in social life. For many large places, these data can provide important basis for managers in the statistical analysis of market data, the management and decision-making of public places.

The Flow Monitoring system counts the time of tourists passing through the entrance by deploying sensors on the revolving door or ticket gate. On this website, you can see the tourist flow data collected by Flow Monitoring system.

Please click the "Collected Data" button in the upper right corner to view the traffic flow information, which may require you to wait few seconds, because the statistical graph will be generated in real time when you click the link.

Contact US

If you have any question please email anyone of us: Cathy ZHAO, Xichun GOU, Zheng LI

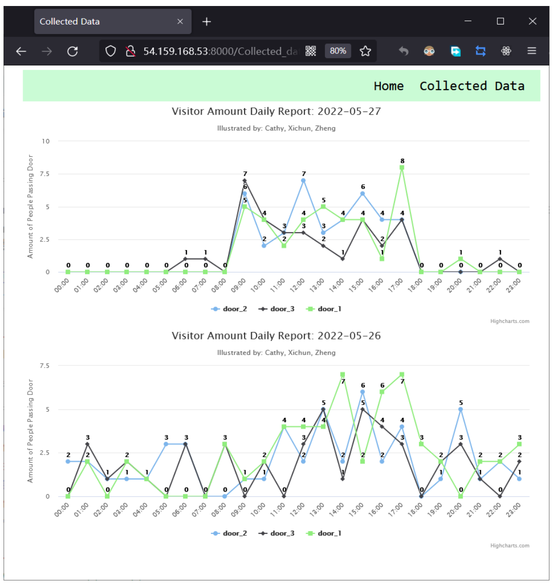

Collected Data
54.159.168.53:8000/Collected_da
Home
Collected Data
Visitor Amount Daily Report: 2022-05-27
Illustrated by: Cathy, Xichun, Zheng
Amount of People Passing Door
door_2
door_3
door_1
Highcharts.com
Visitor Amount Daily Report: 2022-05-26
Illustrated by: Cathy, Xichun, Zheng
Amount of People Passing Door
door_2
door_3
door_1
Highcharts.com

Fig. 3.1.1 Home Page screenshot

Fig. 3.1.2 Collected Data screenshot

Database Connection:

Create a cloud database and change the database configuration in the setting.py file to connect to the cloud

```
DATABASES = {
    'default': {
        # 'ENGINE': 'django.db.backends.sqlite3',
        # 'NAME': BASE_DIR / 'db.sqlite3',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'doorsmove',#数据库名称
        'USER':'postgres',#拥有者，这个一般没修改
        'PASSWORD':'123456',#密码，自己设定的
        'HOST':'localhost',#默认的就没写
        'PORT':'5432',
    }
}
```

Fig. 3.1.3 database connection

Code of showing line chart data:

http://<server_IP_address>:8000/Collected_data.html

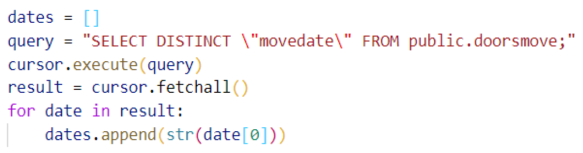
This page shows Flow Monitoring's simple analysis and visualization of visitor flow data in the database, which is responsible for the file testdb.py. The process of testdb.py showing line charts in the collected data page for users is divided into four parts:

In the first part, the program queries the database for how many gates that our system is currently monitoring, and what the gate names are.

```
doors = []
query = "SELECT DISTINCT \"position\" FROM public.doorsmove;"
cursor.execute(query)
result = cursor.fetchall()
for door in result:
    doors.append(door[0])
```

Fig 3.1.4 Source code for collecting doors monitered

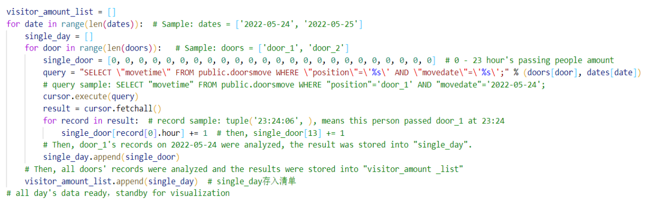
In the second part, the program queries the database for how many days and how much flow information that our system has stored since it was started.



```
dates = []
query = "SELECT DISTINCT \"movedate\" FROM public.doorsmove;"
cursor.execute(query)
result = cursor.fetchall()
for date in result:
    dates.append(str(date[0]))
```

Fig 3.1.5 Source coe for collecting date recorded

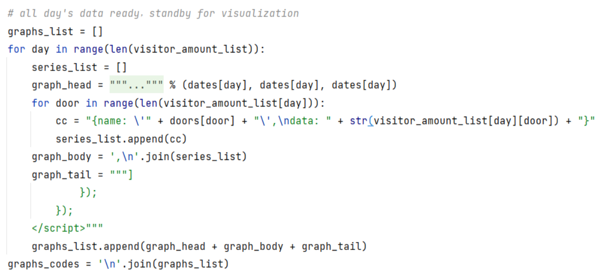
In the third part, the program queries the daily traffic records according to the information obtained in the first two parts, and generates the HTML code of the line chart according to the passenger flow information.



```
visitor_amount_list = []
for date in range(len(dates)):  # Sample: dates = ['2022-05-24', '2022-05-25']
    single_day = []
    for door in range(len(doors)):   # Sample: doors = ['door_1', 'door_2']
        single_door = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  # 0 - 23 hour's passing people amount
        query = "SELECT \"movetime\" FROM public.doorsmove WHERE \"position\"=\'%s\' AND \"movedate\"=\'%s\';" % (doors[door], dates[date])
        # query sample: SELECT "movetime" FROM public.doorsmove WHERE "position"='door_1' AND "movedate"='2022-05-24';
        cursor.execute(query)
        result = cursor.fetchall()
        for record in result:  # record sample: tuple('23:24:06', ), means this person passed door_1 at 23:24
            single_door[record[0].hour] += 1  # then, single_door[13] += 1
        # Then, door_1's records on 2022-05-24 were analyzed, the result was stored into "single_day".
        single_day.append(single_door)
    # Then, all doors' records were analyzed and the results were stored into "visitor_amount _list"
    visitor_amount_list.append(single_day)  # single_day存入清单
# all day's data ready, standby for visualization
```

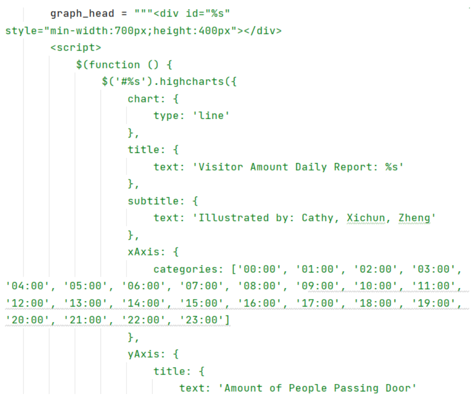
Fig 3.1.6 Source code for getting and analyzing daily data

When obtaining the daily passenger flow information, query and analyze the traffic records of each door on that day in turn. After the analysis is over, the code generates the line chart which based on the daily analysis results, and the amount of line charts are equel to the amount of days in the database.



```
# all day's data ready, standby for visualization
graphs_list = []
for day in range(len(visitor_amount_list)):
    series_list = []
    graph_head = """...""" % (dates[day], dates[day], dates[day])
    for door in range(len(visitor_amount_list[day])):
        cc = "{name: \'" + doors[door] + "\',\ndata: " + str(visitor_amount_list[day][door]) + "}"
        series_list.append(cc)
    graph_body = ',\n'.join(series_list)
    graph_tail = """]
            });
        });
    </script>"""
    graphs_list.append(graph_head + graph_body + graph_tail)
graphs_codes = '\n'.join(graphs_list)
```

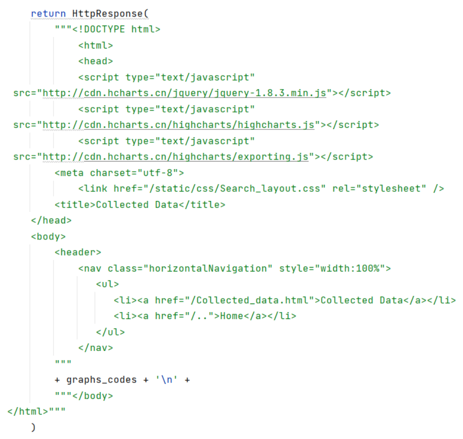
Fig 3.1.7.a. Generate daily diagrams based on analyzation result, each item in graphs_list is a graph's code



```
    graph_head = """<div id="%s"
style="min-width:700px;height:400px"></div>
    <script>
        $(function () {
            $('#%s').highcharts({
                chart: {
                    type: 'line'
                },
                title: {
                    text: 'Visitor Amount Daily Report: %s'
                },
                subtitle: {
                    text: 'Illustrated by: Cathy, Xichun, Zheng'
                },
                xAxis: {
                    categories: ['00:00', '01:00', '02:00', '03:00',
'04:00', '05:00', '06:00', '07:00', '08:00', '09:00', '10:00', '11:00',
'12:00', '13:00', '14:00', '15:00', '16:00', '17:00', '18:00', '19:00',
'20:00', '21:00', '22:00', '23:00']
                },
                yAxis: {
                    title: {
                        text: 'Amount of People Passing Door'
```

Fig 3.1.7.b. Part of long string graph_head

The fourth part it generates the HTML code in the third part and return the final HTTPResponse object.

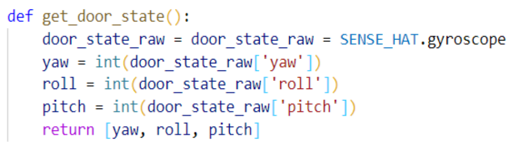


```
    return HttpResponse(
        """<!DOCTYPE html>
            <html>
            <head>
            <script type="text/javascript"
   src="http://cdn.hcharts.cn/jquery/jquery-1.8.3.min.js"></script>
            <script type="text/javascript"
   src="http://cdn.hcharts.cn/highcharts/highcharts.js"></script>
            <script type="text/javascript"
   src="http://cdn.hcharts.cn/highcharts/exporting.js"></script>
        <meta charset="utf-8">
            <link href="/static/css/Search_layout.css" rel="stylesheet" />
        <title>Collected Data</title>
    </head>
    <body>
        <header>
            <nav class="horizontalNavigation" style="width:100%">
               <ul>
                  <li><a href="/Collected_data.html">Collected Data</a></li>
                  <li><a href="/..">Home</a></li>
               </ul>
            </nav>
        """
        + graphs_codes + '\n' +
        """</body>
</html>"""
    )
```

Fig 3.1.8 Source code for generating entire HTTP Response message

3.2 Raspberry Pi Sensor Implementation

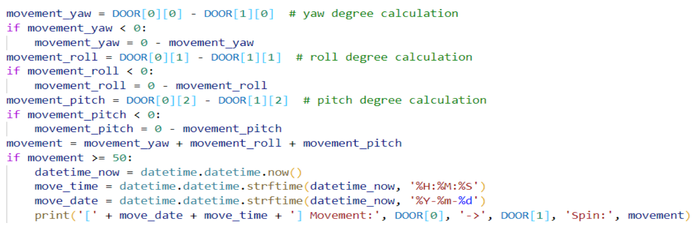
The RPi connects to the Sense HAT gyroscope via GPIO, and the program gets gyro information every second via the function get_door_state().



```
def get_door_state():
    door_state_raw = door_state_raw = SENSE_HAT.gyroscope
    yaw = int(door_state_raw['yaw'])
    roll = int(door_state_raw['roll'])
    pitch = int(door_state_raw['pitch'])
    return [yaw, roll, pitch]
```

Fig 3.2.1. Source code of function get_door_state()

The information returned by the gyroscope indicates the current RPi, which is the attitude information of the door. The program compares the current position of the door to the position one second ago. The two sets of positions are stored when the difference between the two positions is large enough to assume that someone pushed the door and passed through.



```
movement_yaw = DOOR[0][0] - DOOR[1][0]  # yaw degree calculation
if movement_yaw < 0:
    movement_yaw = 0 - movement_yaw
movement_roll = DOOR[0][1] - DOOR[1][1]  # roll degree calculation
if movement_roll < 0:
    movement_roll = 0 - movement_roll
movement_pitch = DOOR[0][2] - DOOR[1][2]  # pitch degree calculation
if movement_pitch < 0:
    movement_pitch = 0 - movement_pitch
movement = movement_yaw + movement_roll + movement_pitch
if movement >= 50:
    datetime_now = datetime.datetime.now()
    move_time = datetime.datetime.strftime(datetime_now, '%H:%M:%S')
    move_date = datetime.datetime.strftime(datetime_now, '%Y-%m-%d')
    print('[' + move_date + move_time + '] Movement:', DOOR[0], '->', DOOR[1], 'Spin:', movement)
```

Fig 3.2.2. Source code of judging whether there is a person passing the door, if yes, print relating information.

When someone is thought to have passed, the RPi sends a record to the database, including the location of the RPi and the date and time the visitor passed through the door. The action of sending a record is done by the function upload_enter().

```python
def upload_enter(door_place, enter_date, enter_time):
    db = psycopg2.connect(database='doorsmove',
                          user='postgres',
                          password='123456',
                          host='18.206.146.30',
                          port='5432')
    head = db.cursor()  # why call it "head"? Think about head of a HDD
    sql_sentence = """INSERT INTO public.doorsmove(
        "position", movedate, movetime)
        VALUES ('%s', '%s', '%s');""" % (door_place, enter_date, enter_time)
    head.execute(sql_sentence)
    db.commit()
    head.close()
    db.close()
    return
```

Fig 3.2.3. Source code of function upload_enter()

The program allows for multiple people passing through the door quickly and one at a time. As uploading data often takes a long time (according to the test, it usually takes 0.8-2.1 seconds for a upload), it affects the normal count in the case of multiple people passing through the door quickly and continuously. Here, multithreading technology is applied to try to alleviate this problem.

```python
if movement >= 50:
    datetime_now = datetime.datetime.now()
    move_time = datetime.datetime.strftime(datetime_now, '%H:%M:%S')
    move_date = datetime.datetime.strftime(datetime_now, '%Y-%m-%d')
    print('[' + move_date + move_time + '] Movement:', DOOR[0], '->', DOOR[1], 'Spin:', movement)
    commit_thread = threading.Thread(target=upload_enter, args=(door_position, move_date, move_time))
    commit_thread.start()
```

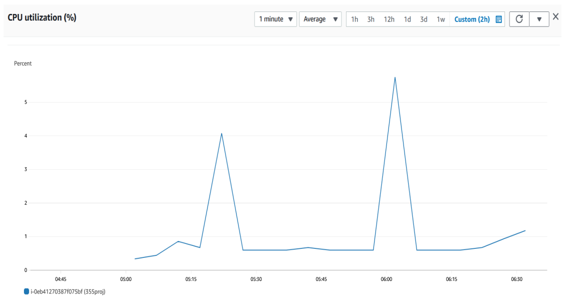
Fig 3.2.4. Source code of multi-threading

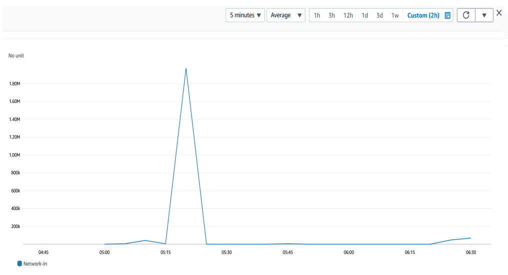
3.3 Real-time performance

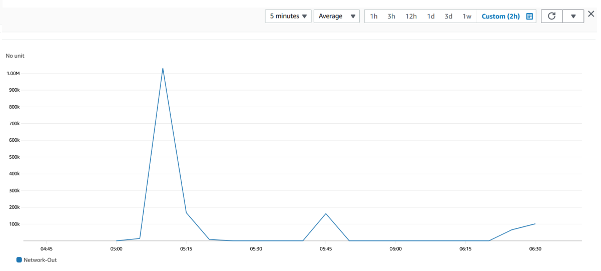
3.3.1 Server performance

We wrote some scripts to run a stress test to our database server (which also works as our web server). We didn't make stress test for our web server, because this site is prepared for apartment administrators, mall managers or other people who have similar jobs. However, there will be lots of informations uploaded to database from doors.

Here are some server performance metrics procided by AWS Cloud Watch, it can be seen that the server is able to make through the stress test. We simulated 50 people passing three doors in a short time. The test was executed 6 rounds. From the data provided by Cloud Watch, the server can easily bear the workload we estimated.

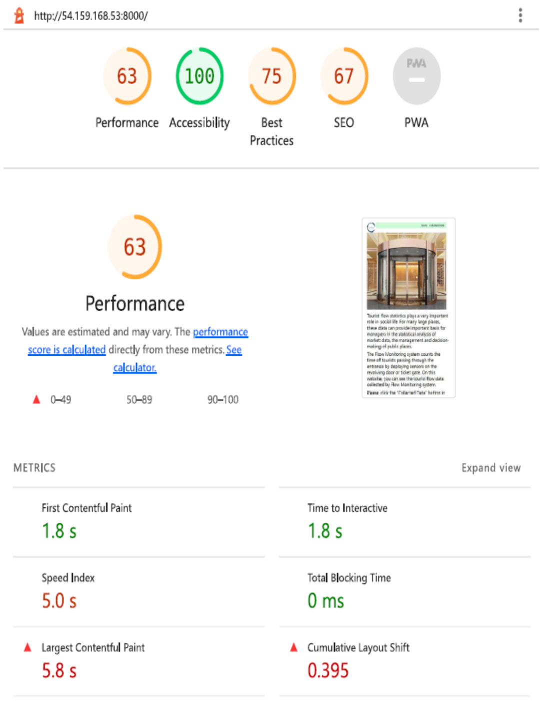






3.3.2 Webpage performance

We used Chrome Lighthouse 9.5.0 to perform a test for our two webpages, here are parts of our test reports.



http://54.159.168.53:8000/

63 Performance | 100 Accessibility | 75 Best Practices | 67 SEO | PWA

63

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 | 50–89 | 90–100

METRICS | Expand view

First Contentful Paint 1.8 s

Time to Interactive 1.8 s

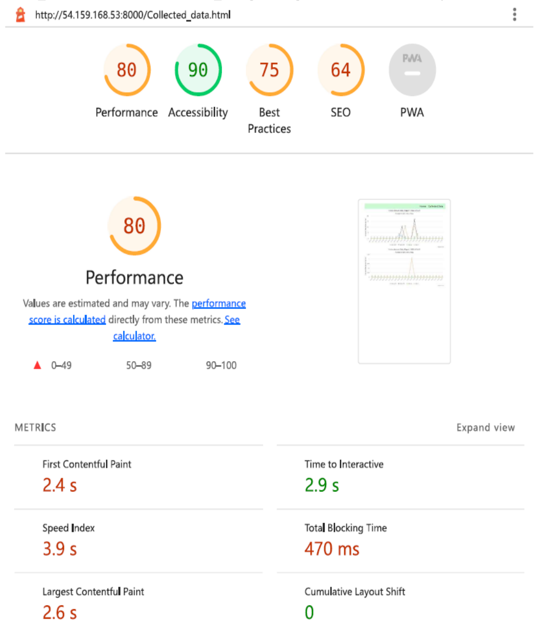
Speed Index 5.0 s

Total Blocking Time 0 ms

▲ Largest Contentful Paint 5.8 s

▲ Cumulative Layout Shift 0.395

Fig 3.3.3 Test Report for homepage, generated by Chrome Lighthouse



http://54.159.168.53:8000/Collected_data.html

80 Performance | 90 Accessibility | 75 Best Practices | 64 SEO | PWA

80

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 | 50–89 | 90–100

METRICS | Expand view

First Contentful Paint 2.4 s

Time to Interactive 2.9 s

Speed Index 3.9 s

Total Blocking Time 470 ms

Largest Contentful Paint 2.6 s

Cumulative Layout Shift 0

Fig 3.3.4. Test Report for "Collected Data" page, generated by Chrome Lighthouse

4 Challenges and Areas for improvement

4.1 The phenomenon of walking in groups

At the entrance of some public places, it is inevitable that there are many people walking together, which will seriously affect the accuracy of the statistics of people flow. When there is a complex situation of multiple people sticking together in the scene, such as multiple people sticking together closely or adults holding children, the statistical effect of the system will be greatly reduced.

4.2 Time Issue

For the number of people counting link is not mature enough, to solve the basic function of the statistics of the flow of people, but to be further improved. In the paper the main programming of design to meet the design requirements, complete system as the goal, not too much to consider the length of the program execution time of algorithm, or whether there is a delay problems, etc., can follow-up in terms of programming optimization to further shorten the running time of the system needed to complete the number of statistical process, makes the system more real-time and accuracy.
In the collected_data.html page, the Web server sends multiple query requests to the database in order to generate visitor traffic statistics: Suppose that in a single visit, Float Monitoring has been deployed for two days to collect visitor traffic statistics on four entries. This visit to Collected_data.html causes the server to send 2*5=10 query requests to the database. This obviously takes a lot of time to prepare the response to the visitor, putting a lot of strain on both the Web server and the database server.

5 Conclusion

The problem of the flow of people in public places (such as banks, libraries, shopping malls, classrooms, etc.) is becoming more and more prominent. The system can effectively count the number of personnel, with the characteristics of real-time, security, low cost, can be widely used in public real-time statistics of the number of people.

Reference:
[1] The design and implementation of real-time traffic statistics and early warning system based on big data [J]. Wang Min. Information and Computer (Theory Edition). 2021(08)
[2] The design and application of travel information analysis system based on wireless communication network [J]. Liu Jie, Hu Xianbiao, Fu Dandan, Chen Mingwei. Highway Transportation Technology. 2009(S1)
[3] The "most Popular database" ranking is based on https://db-engines.com/en/ranking_trend