A DATA-DRIVEN PLATFORM TO COMPUTER PERFORMANCE ANALYSIS AND RECOMMENDATION USING AI AND BIG DATA ANALYSIS

Shuo Chen[1] and Yu Sun[2]

[1]Yorba Linda High School, 19900 Bastanchury Rd, Yorba Linda, CA 92886
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

When I was assembling the computer, I found a problem. This problem is that we need to spend a lot of time and energy when we choose a desktop with a configuration and price that we are satisfied with [5]. Some computer websites will only recommend some ordinary desktops to users. Does not allow users to get what they really want, and some other shops that assemble computer mainframes use the characteristics of customers that do not understand computers to increase prices. So I wanted to create a software to help these people who need to assemble a computer to find the most suitable computer efficiently and in accordance with their requirements [6].

This program, according to the needs of users, artificial intelligence application crawler technology can help users find the most suitable computer parts based on big data, and help users get the most cost-effective self-assembled computer host. We applied our application to match a person in need of a computer host with My Platform and conducted a qualitative evaluation of the method [7]. The results showed that My Platform can efficiently and quality match the user's needs and find the best solution for the user.

KEYWORDS

Computer Performance, Big Data Analysis, Recommendation System, DATA Mining.

1. INTRODUCTION

Many people have high requirements for computer productivity, and many laptops cannot meet everyone's needs, so people need to put together a desktop computer according to their next needs, and then they need to choose the most cost-effective and best-performing desktop computer, and I designed a platform that allows people to efficiently find all eligible configurations and provide them to the user, so that the user does not have to spend a very long time to choose each accessory.

There are some problems with the first websites about computer assembly [8]. First, because of how well known these websites are, many items are in short supply and these shortages lead to many people who need a computer not being able to get a satisfactory computer. Second, many people who need a computer don't know where to get the best resources, which leads to many people who will pay high prices for accessories [9]. Many businesses are unable to provide users with a comprehensive range of accessories because of their inventory, in which case users are unable to find the most satisfactory computer configuration.

Our goal is to find all parts that users need faster and more accuracy, I use a web crawler to find the availability of different sites, based on each site to help the user complete a set of what we think is best for the user's computer and show them what this computer needs to work and its efficiency and performance, in addition each component of the computer parts has a tab where the user can replace any of the parts that he is not satisfied with [10]. We do not limit the inventory of a site, all the sources available for crawling are displayed here and can help the user to find all the resources [11].

Experienced computer assemblers can use this site to find available computer parts and get some advice from us. For inexperienced computer assemblers, you can use this site to complete your personal computer assembly and thus learn about computers. Since my site is open to everyone, I will measure the extent to which this site is helpful and satisfying to everyone through a user survey.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the experiment and designing the sample (for example, web scraper); Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the tracking system, a few challenges have been identified as follows.

2.1. Solving problems for users

The first problem is how to solve the problem of finding the complete version for the user, because there are many different parts to build a PC and some of them are not needed, only certain core items are necessary and others are considered according to the user's budget, so I need to identify the key requirements of the user and find the relevant parts and then give the complete PC that is within the user's budget [15].

2.2. Choosing presentation type

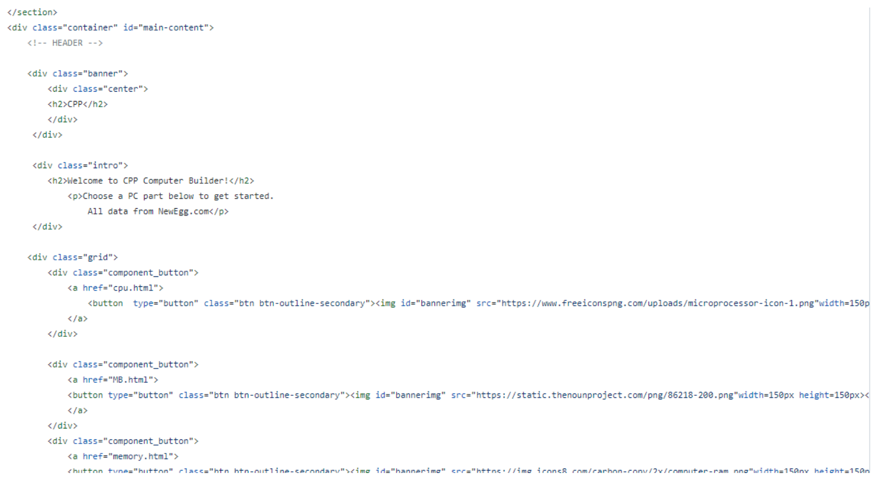
The second question is when I find a suitable PC for the user, what kind of presentation should I use to show it to the user, when a person with some experience sees the configuration, even if it is not shown, but if a person who does not know anything about computers does not show them the corresponding data, it is not possible for them to use the computer with confidence.

2.3. How to implement the website

Our last challenge is how to implement this website? How to obtain more resources through different websites? Different resources will lead to different results for users. If the data obtained is not enough, it is impossible to provide users with the most complete configuration according to user requirements.

3. SOLUTION

Overall the system has three components, including User-interaction end, web-frame structure, and maintenance back-end. The user interaction end takes user input and sets triggers for the program to record user selection, where the user can choose different types of components they like [14]. Each of the selections will be then passed to the web server.



```
</section>
<div class="container" id="main-content">
    <!-- HEADER -->

    <div class="banner">
        <div class="center">
        <h2>CPP</h2>
        </div>
     </div>

     <div class="intro">
        <h2>Welcome to CPP Computer Builder!</h2>
            <p>Choose a PC part below to get started.
                All data from NewEgg.com</p>
     </div>

    <div class="grid">
        <div class="component_button">
            <a href="cpu.html">
                <button  type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://www.freeiconspng.com/uploads/microprocessor-icon-1.png"width=150p
            </a>
        </div>

        <div class="component_button">
            <a href="MB.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://static.thenounproject.com/png/86218-200.png"width=150px height=150px><
            </a>
        </div>
        <div class="component_button">
            <a href="memory.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://img.icons8.com/carbon-copy/2x/computer-ram.png"width=150px height=150p
```

Figure 1. Code 1

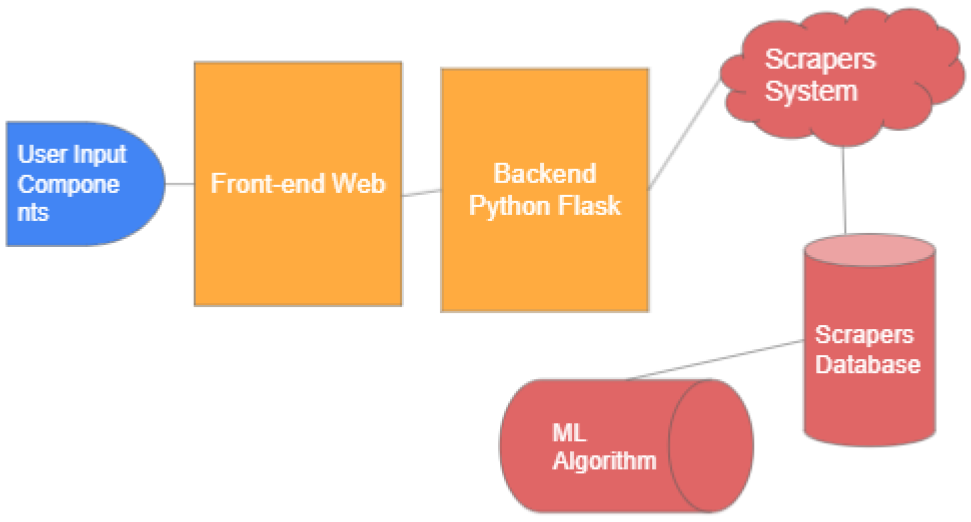
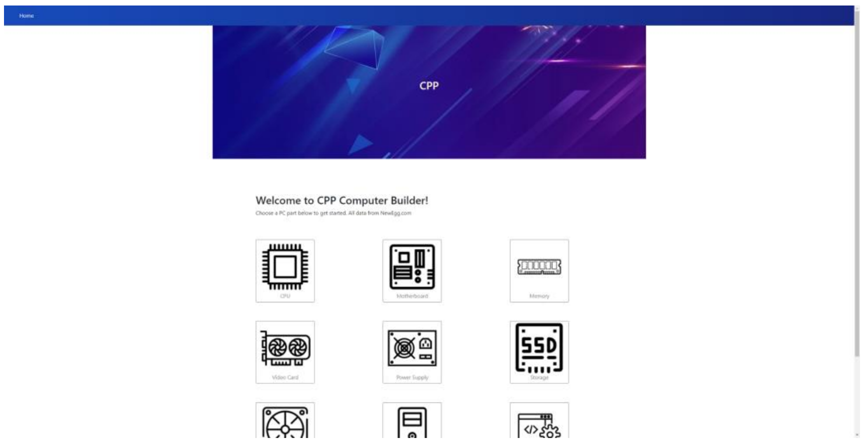


Figure 2. The overview of the system

We have different pages for our website, where one of them being the index (homepage) while the rest being a selection page for each pc component. Each of the PC component pages will enable users to view different part products in this category and enable them to make a selection. The server will track the users selection from each page, make sure the user is seeing the correct total amount/redirected to the corresponding pages, though this way we can better understand the user's preferences and recommend a computer that better matches the user's personal preferences.Lastly, we also have a machine learning service, where the server will be updating the database (what components are available) and provide this information to the front end through web service.

<form input = “xxx”, url = “/get_cpu”>
abc.com/get_cpu&xxx
request: from 172.18.0.1

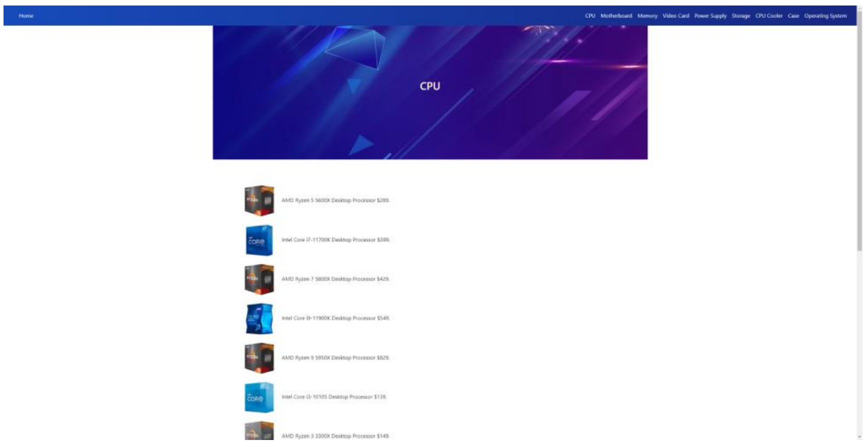
In the previous section we discussed having an index page to display the overall information for a user, here’s how we actually implemented it. First we use HTML with header xxx, then we have bodies, inside bodies we have the components for each part of the PC [13]. The user's input (user selection) is recorded through xx.js, where the HTML includes this JavaScript at line xxx. This JavaScript also sends the recorded information to the back-end through Ajax.



Welcome to CPP Computer Builder!

Choose a PC part below to get started. All data from NewEgg.com

Figure 3. Screenshot of CPP computer builder


Home
CPU Motherboard Memory Video Card Power Supply Storage CPU Cooler Case Operating System
CPU

AMD Ryzen 5 5600X Desktop Processor $289.

Intel Core i7-11700K Desktop Processor $399.

AMD Ryzen 7 5800X Desktop Processor $429.

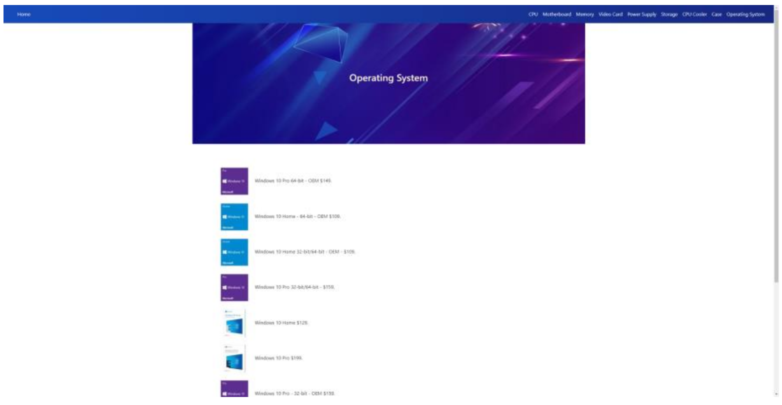
Intel Core i9-11900K Desktop Processor $549.

AMD Ryzen 9 5950X Desktop Processor $829.

Intel Core i3-10105 Desktop Processor $139.

AMD Ryzen 3 3300X Desktop Processor $149.

Figure 4. Screenshot of CPU



Windows 10 Pro 64-bit - OEM $149.

Windows 10 Home - 64-bit - OEM $109.

Windows 10 Home 32-bit/64-bit - OEM - $109.

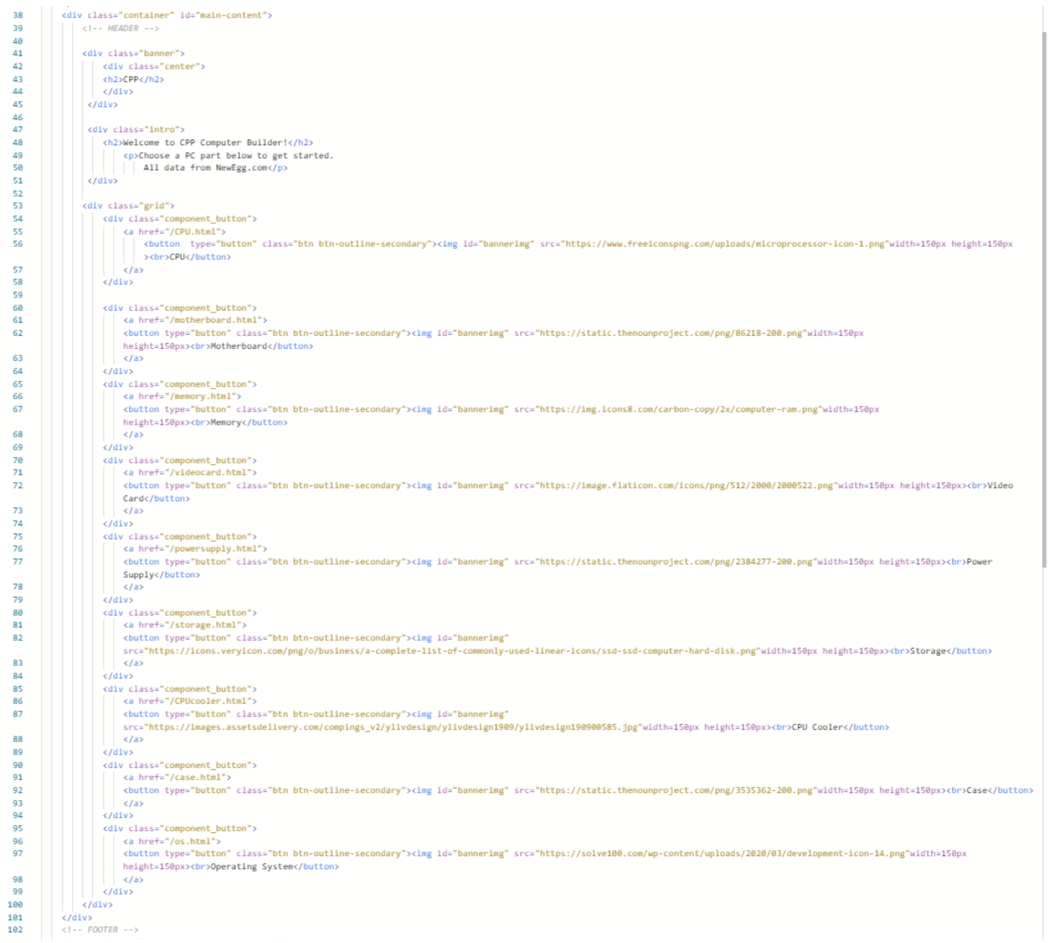
Windows 10 Pro 32-bit/64-bit - $159.

Windows 10 Home $129.

Windows 10 Pro $199.

Windows 10 Pro - 32-bit - OEM $159.

Figure 5. Screenshot of Operation System

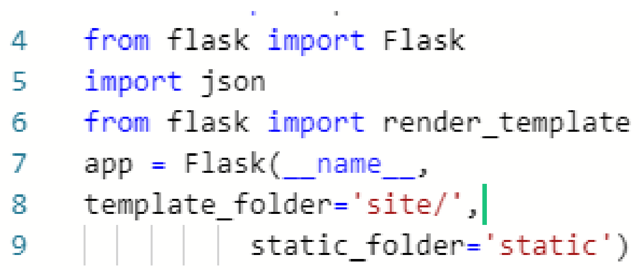


```html
<div class="container" id="main-content">
    <!-- HEADER -->

    <div class="banner">
        <div class="center">
        <h2>CPP</h2>
        </div>
     </div>

     <div class="intro">
        <h2>Welcome to CPP Computer Builder!</h2>
            <p>Choose a PC part below to get started.
                All data from NewEgg.com</p>
     </div>

    <div class="grid">
        <div class="component_button">
            <a href="/CPU.html">
                <button  type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://www.freeiconspng.com/uploads/microprocessor-icon-1.png"width=150px height=150px
                ><br>CPU</button>
            </a>
        </div>

        <div class="component_button">
            <a href="/motherboard.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://static.thenounproject.com/png/86218-200.png"width=150px
            height=150px><br>Motherboard</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/memory.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://img.icons8.com/carbon-copy/2x/computer-ram.png"width=150px
            height=150px><br>Memory</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/videocard.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://image.flaticon.com/icons/png/512/2000/2000522.png"width=150px height=150px><br>Video
            Card</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/powersupply.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://static.thenounproject.com/png/2384277-200.png"width=150px height=150px><br>Power
            Supply</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/storage.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg"
            src="https://icons.veryicon.com/png/o/business/a-complete-list-of-commonly-used-linear-icons/ssd-ssd-computer-hard-disk.png"width=150px height=150px><br>Storage</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/CPUcooler.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg"
            src="https://images.assetsdelivery.com/compings_v2/ylivdesign/ylivdesign1909/ylivdesign190900585.jpg"width=150px height=150px><br>CPU Cooler</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/case.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://static.thenounproject.com/png/3535362-200.png"width=150px height=150px><br>Case</button>
            </a>
        </div>
        <div class="component_button">
            <a href="/os.html">
            <button type="button" class="btn btn-outline-secondary"><img id="bannerimg" src="https://solve100.com/wp-content/uploads/2020/03/development-icon-14.png"width=150px
            height=150px><br>Operating System</button>
            </a>
        </div>
    </div>
</div>
<!-- FOOTER -->
```

Figure 6. Code 2-Front end html (index.html)

This image describes the very first page of the website, index.html. It is made up with different layers of divs and html components. For each of the user's parts selection a separate div is created, inside it contains a “<a href=””>” tag which links to different urls to display the specific parts. For example, the storage button contains “<a href=”storage.html”>” which means by clicking on this button the user will be redirected to the storage page, to look at potential storage items.

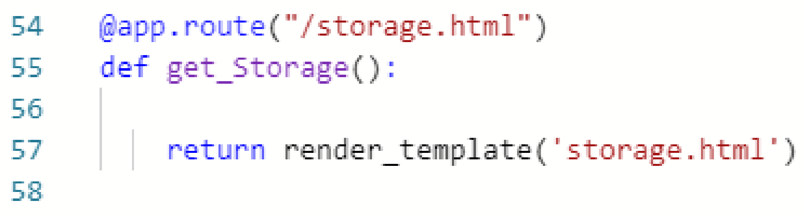
This image talks about how the front page looks and how all the buttons work, when a user clicks on a button it will send a request to.



```python
4   from flask import Flask
5   import json
6   from flask import render_template
7   app = Flask(__name__,
8   template_folder='site/',
9               static_folder='static')
```

Figure 7. Code 3-Flask import

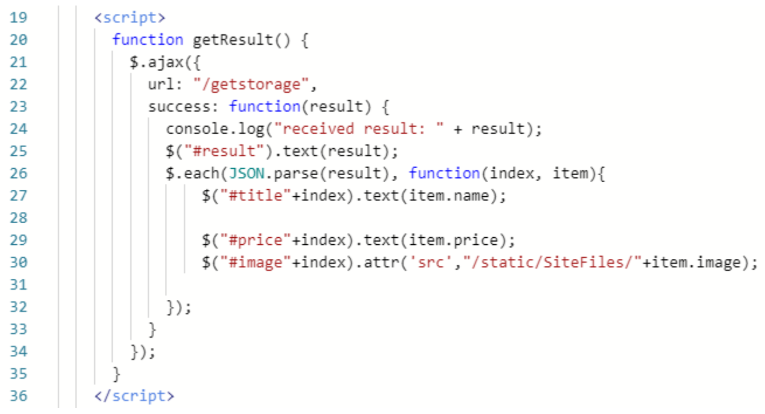
After the storage button is clicked, a request will be sent to the flask server. Flask is a lightweight web server package written in python. Line 4-8 described the import of such a package and the initiation of the object instance. The template_folder parameter decides where the server will look for different templates and the static folder will be where the server looks for static files.



```python
@app.route("/storage.html")
def get_Storage():

    return render_template('storage.html')

```

Figure 8. Code 3-APP route (1)

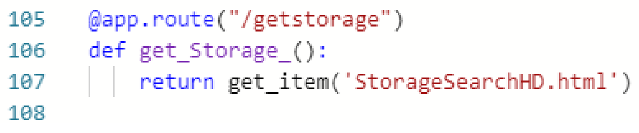
Inside flask, the user's url is parsed through view functions. We used app.route in this case to manage the views of the server. To continue with the example above, the user click on storage button, then it will be redirected to this function here, where it will return a rendered template of "storage.html".



```html
<script>
  function getResult() {
    $.ajax({
      url: "/getstorage",
      success: function(result) {
        console.log("received result: " + result);
        $("#result").text(result);
        $.each(JSON.parse(result), function(index, item){
            $("#title"+index).text(item.name);

            $("#price"+index).text(item.price);
            $("#image"+index).attr('src',"/static/SiteFiles/"+item.image);

        });
      }
    });
  }
</script>
```

Figure 9. Code 4 –Script

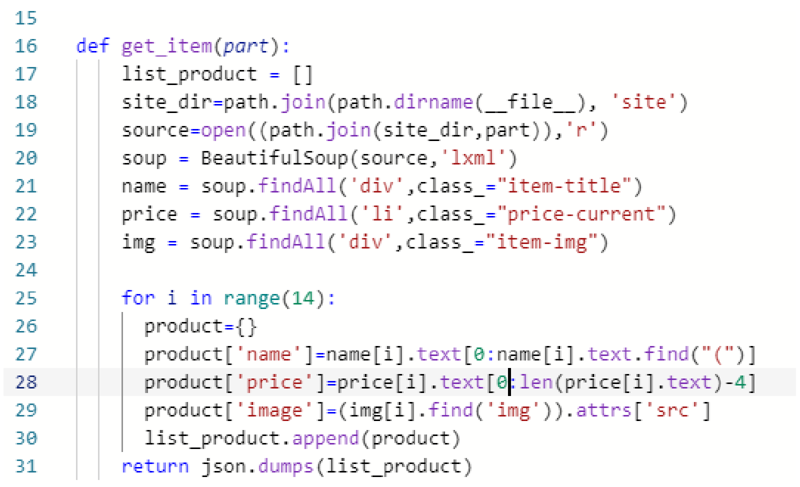
Inside the storage.html, besides the conventional header and footer, it uses a jquery ajax function to request for the specific items to be displayed in the front end. By calling a get request to "/getstroage", it received the list of available storage item from the server.



```python
@app.route("/getstorage")
def get_Storage_():
    return get_item('StorageSearchHD.html')

```

Figure 10. Code 5-APP route (2)

On the server end, once received the request of "/getstorage", it is parsed again using the view function. This time it is redirected to another one where it returns "get_item('storagesearchHD.html", which is a function that package the pre generated html and parse into readable json by the front end.



```python

def get_item(part):
    list_product = []
    site_dir=path.join(path.dirname(__file__), 'site')
    source=open((path.join(site_dir,part)),'r')
    soup = BeautifulSoup(source,'lxml')
    name = soup.findAll('div',class_="item-title")
    price = soup.findAll('li',class_="price-current")
    img = soup.findAll('div',class_="item-img")

    for i in range(14):
        product={}
        product['name']=name[i].text[0:name[i].text.find("(")]
        product['price']=price[i].text[0:len(price[i].text)-4]
        product['image']=(img[i].find('img')).attrs['src']
        list_product.append(product)
    return json.dumps(list_product)
```

Figure 11. Code 6-Get item

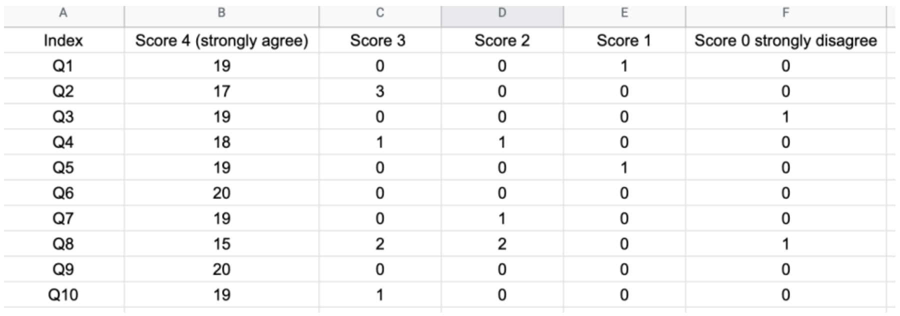
In the get_item function, it opens local saved results, parses each component by the tag and its class. Our results as of now come directly from the new egg, hence after studying their naming patterns we extracted the image, price, and item title to be displayed. For each item, it is packed into a python dictionary, and passed to the front end as a json file.

In the above we discussed having a web server taking care of all URL requests, the implementation of the web server is done through flask. There are 9 URLs in total where each URL is dispatched through the dispatcher to each view function.

4. Experiment

4.1. Experiment 1

We have 30 users doing the survey in total, the results show below:



A	B	C	D	E	F
Index	Score 4 (strongly agree)	Score 3	Score 2	Score 1	Score 0 strongly disagree
Q1	19	0	0	1	0
Q2	17	3	0	0	0
Q3	19	0	0	0	1
Q4	18	1	1	0	0
Q5	19	0	0	1	0
Q6	20	0	0	0	0
Q7	19	0	1	0	0
Q8	15	2	2	0	1
Q9	20	0	0	0	0
Q10	19	1	0	0	0

Figure 12. Table of result

The results shows the system have a high score review from the users

According to my survey, the user's response to this website is that this website is easier to view and can find all the data directly and quickly. A lot of users think they don’t need to learn anything to know how to use this web, it is steet forward to the point.

According to my survey, some users are foreign websites and should have more data so that they can access all the data more quickly and conveniently, so that they can get better information.

The way to solve this problem is to use crawler technology to find more data and store it in our database so that users can get more information.

4.2. Experiment 2

In the second experiment, we ask users to rate the most satisfy components they bought from the website, most of them choose cpu, which matches the algorithm we designed which make sure the performance of the CPU first, the second high score component is GPU.

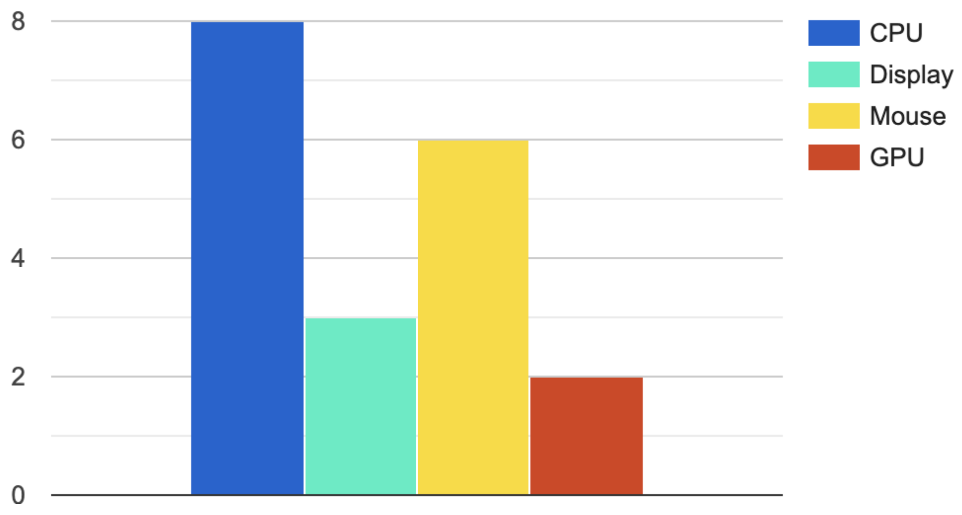


Figure 13. Score of each part

According to the two experiments, users prefer my website because my website is simpler and clearer, allowing users to directly find the information they want, and get all the data and prices from it. In addition, my website can provide him with datas.Also my website provides components lists with high performance with CPU and GPU.

5. RELATED WORK

The content of the web has increasingly become a focus for academic research [2]. Computer programs are needed in order to conduct any large-scale processing of web pages, requiring the use of a web crawler at some stage in order to fetch the pages to be analysed. The processing of the text of web pages in order to extract information can be expensive in terms of processor time. Consequently a distributed design is proposed in order to effectively use idle computing resources and to help information scientists avoid the need to employ dedicated equipment. A system developed using the model is examined and the advantages and limitations of the approach are discussed.

This paper describes Mercator, a scalable, extensible Web crawler written entirely in Java [3]. Scalable Web crawlers are an important component of many Web services, but their design is not well-documented in the literature. We enumerate the major components of any scalable Web crawler, comment on alternatives and trade-offs in their design, and describe the particular components used in Mercator. We also describe Mercator's support for extensibility and customizability. Finally, we comment on Mercator's performance, which we have found to be comparable to that of other crawlers for which performance numbers have been published.

Broad Web search engines as well as many more specialized search tools rely on Web crawlers to acquire large collections of pages for indexing and analysis [4]. Such a Web crawler may interact with millions of hosts over a period of weeks or months, and thus issues of robustness, flexibility, and manageability are of major importance. In addition, I/O performance, network resources, and OS limits must be taken into account in order to achieve high performance at a reasonable cost. In this paper, we describe the design and implementation of a distributed Web crawler that runs on a network of workstations. The crawler scales to (at least) several hundred pages per second, is resilient against system crashes and other events, and can be adapted to various crawling applications. We present the software architecture of the system, discuss the performance bottlenecks, and describe efficient techniques for achieving high performance. We also report preliminary experimental results based on a crawl of 120 million pages on 5 million hosts.

6. CONCLUSIONS

I found that when we need a computer, the laptops on the market cannot meet our needs. At this time we need to assemble a host computer that meets our needs, but it is a big challenge for a person who does not understand computers [1]. Because there are various accessories on the Internet, it is difficult to successfully assemble a computer that meets your needs if you don’t know the computer. Now there are many websites that have some automatic configuration tools, but they can’t provide users with them because of their imperfect inventory. The best service, so I thought of a way. I can use crawler technology to search the inventory of all websites on a large scale, and then according to the inventory and combined with the needs of the user, I recommend a computer that suits him and give him all the resources. Users, let users have more choices.

Sometimes it is limited by manufacturing capacity and no website provides usable products [12]. At this time, my solution is to give all available resources to the user for him to choose from, or if the user is not in a hurry, then he can wait until the resources are available before proceeding.

In the future, if possible, I hope to cooperate with different platforms to access limited stocks and obtain more resources for users.

REFERENCES

[1] Witten, Ian H., et al. "Practical machine learning tools and techniques." DATA MINING. Vol. 2. 2005.
[2] Thelwall M. A web crawler design for data mining. Journal of Information Science. 2001;27(5):319-325. doi:10.1177/016555150102700503
[3] Heydon, A., Najork, M. Mercator: A scalable, extensible Web crawler. World Wide Web 2, 219–229 (1999). https://doi.org/10.1023/A:1019213109274
[4] V. Shkapenyuk and T. Suel, "Design and implementation of a high-performance distributed Web crawler," Proceedings 18th International Conference on Data Engineering, 2002, pp. 357-368, doi: 10.1109/ICDE.2002.994750.
[5] Suchman, Lucy. "Configuration." Inventive methods. Routledge, 2012. 62-74.
[6] Pedersen, Lasse Heje. Efficiently inefficient. Princeton University Press, 2015.
[7] Patton, Michael Quinn. Qualitative evaluation methods. Vol. 381. Beverly Hills, CA: Sage publications, 1980.
[8] Hardt, Michael, and Antonio Negri. Assembly. Oxford University Press, 2017.
[9] Lighting, Home. "Accessories." Kenroy International wall lantern (1995): 42.
[10] Ross, Stephen A. "Options and efficiency." The Quarterly Journal of Economics 90.1 (1976): 75-89.
[11] Olston, Christopher, and Marc Najork. Web crawling. Now Publishers Inc, 2010.
[12] Morgan, Peter. "The concept of capacity." European Centre for Development Policy Management (2006): 1-19.
[13] Musciano, Chuck, and Bill Kennedy. HTML & XHTML: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.", 2002.
[14] Cox, David R. "Interaction." International Statistical Review/Revue Internationale de Statistique (1984): 1-24.
[15] Laffan, Brigid, and Michael Shackleton. "The budget." Policy-making in the European Union (2000): 191-212.