Shopping Spree

Shop Top Brands All in One Place

December 15, 2019

PREPARED FOR:

Professor Houwei Cao | Department of Computer Science

REPRESENTATIVE

Aemun Ahmar

Neha Bala

Fernanda Tovar

ABSTRACT

Smart Shopping was a simple website we previously created for the compilation of different clothing products from different brands all in one place. The main objective of this website was to make the searching, viewing, and selection of products easy through simple interaction. Our goal for this project was to make the user's experience as convenient as possible by allowing them to browse for products from different brands all in one place. A key feature to achieve that goal was the search bar. Although our website had one already, it wasn't a workable search bar in the traditional sense. The way the previous search bar functioned was only by brand so if you searched the name of a brand featured on the site, it would redirect you to that brand's page. For this project we wanted to expand on that feature and make it workable by allowing the users to actually search for items that they want to find. The main emphasis of this project lies in providing a user friendly search website for effectively showing the desired results using the new and improved search bar.

INTRODUCTION

Problem

As mentioned before, the main goal of this project was to expand on the search bar feature from our previous ecommerce project to make it workable. Our previous search bar only allowed users to search for a brand's name. Once a brand name was entered, the search bar would redirect you to that brand's page on the site.

Definition

On a website, the search bar is a location on the site that allows users to search the site for what they might be interested in finding. A search bar is usually a single-line text box or search icon with the dedicated function of accepting user input to be searched for in a database.

Application

There are all sorts of applications for a search bar. Most, if not all, websites have one in order to make the user's browsing experience a lot more efficient and convenient. People are inherently lazy so by adding a search bar to your website, you relieve the users of the task of navigating through your website. This is especially useful when your website has a lot of content and pages. In addition, manually scanning through the available options or sections of a website to find an answer, word, item, or any other element is incredibly ineffective. It would take a long time, and the users might not even be able to find what they are looking for. In addition to websites, internet browsers also have search bars. The search bar within a browser allows you to search the Internet for what you want to find. They serve as search engines in a way.

Motivation

Our goal when we were first working on this project was to make the user's browsing experience more efficient and convenient. However, without a proper search bar we hadn't completely achieved that goal. That is why for this project we wanted to work on the previous search bar to make the user's experience a lot more convenient.

Goal

For this project, we wanted users to be able to use the search bar to search for items as well. The goal for this search bar was to have it return a list of items that were relevant to the user's search. For example, if a user searched for "leather boots", the search bar would return a list of items that matched "leather boots".

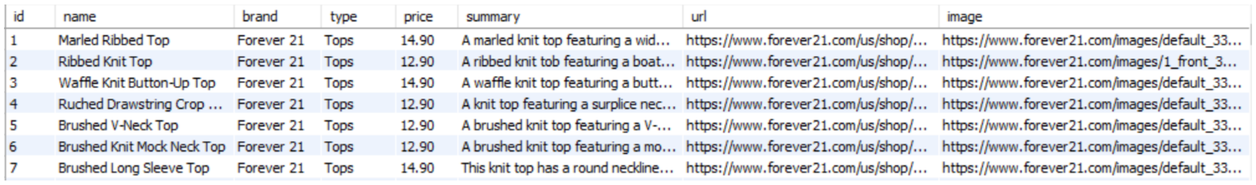
METHODOLOGY

Dataset Description

The dataset that we used for this project was the same one from the previous project. The data was collected from the websites of different brands. We used an external web crawler called ParseHub to crawl the data from those sites. Using ParseHub we were able to get the name, brand, type, price, url, and image url of the selected items. Once the parameters that you want to crawl are selected, ParseHub extracts all of the data and creates a JSON file. In order to use the data from the JSON file, we developed a PHP script to take the data from the file and add them to our SQL database. Below is the PHP script that we developed and the resulting database.

```php
//Read the JSON file
$jsondata = file_get_contents('ProductsListJson.json');

//Decode JSON data
$data = json_decode((utf8_encode($jsondata)), true);
var_dump($data);

//Attempt 3
if(is_array($data))
{
    echo "is array";
    foreach($data as $row)
    {
        //Fetch details of product
        $name = $row["name"];
        $brand = $row["brand"];
        $type = $row["type"];
        $price = $row['price'];
        $summary = $row["summary"];
        $url = $row["url"];
        $image = $row["image"];

        //Inserting fetched data
        $query = "INSERT INTO productslist (name, brand, type, price, summary, url, image) VALUES ('$name', '$brand', '$type',
        '$price', '$summary', '$url', '$image')";

        if(!(mysqli_query($dbconnect, $query)))
        {
            die('Error : Query Not Executed. Please Fix the Issue! ' . mysqli_error($dbconnect));
        } else
        {
            echo "Data Inserted Successully!!!";
        }
    }
```

Figure 1: JSON to database PHP script



id	name	brand	type	price	summary	url	image
1	Marled Ribbed Top	Forever 21	Tops	14.90	A marled knit top featuring a wid...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...
2	Ribbed Knit Top	Forever 21	Tops	12.90	A ribbed knit tob featuring a boat...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/1_front_3...
3	Waffle Knit Button-Up Top	Forever 21	Tops	14.90	A waffle knit top featuring a butt...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...
4	Ruched Drawstring Crop ...	Forever 21	Tops	12.90	A knit top featuring a surplice nec...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...
5	Brushed V-Neck Top	Forever 21	Tops	12.90	A brushed knit top featuring a V-...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...
6	Brushed Knit Mock Neck Top	Forever 21	Tops	12.90	A brushed knit top featuring a mo...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...
7	Brushed Long Sleeve Top	Forever 21	Tops	14.90	This knit top has a round neckline...	https://www.forever21.com/us/shop/...	https://www.forever21.com/images/default_33...

Figure 2: Products Database

One of the main issues we encountered during data collection was that ParseHub could only select a few items at a time. It was very time consuming to get a good amount of data. In addition, there were times when ParseHub wouldn't select the right thing or extract the correct data. In order to make sure it selected the correct things, we had to manually set it up making it very time consuming. As a result, our dataset was relatively small with only 300 items from 5 different brands.

Approach

Programming Languages:

1. ***Front End:***
    a. **HTML/CSS:** used to create the structure of the website
    b. **Javascript:** used to make responsive forms and buttons
2. ***Backend:***
    a. **SQL:** used to create and maintain the database
    b. **PHP:** used to communicate with the database and display products
    c. **JSP:** used to create the search bar functionality

Tools:

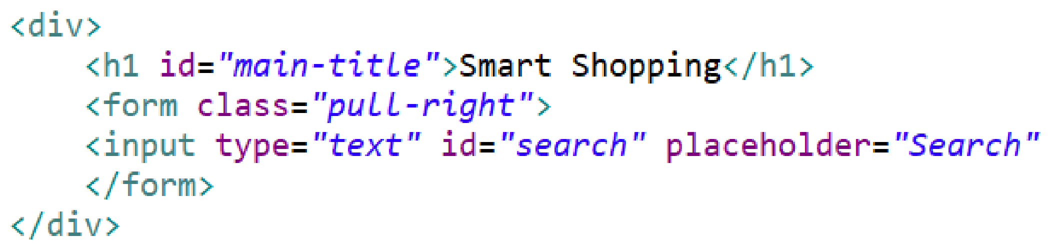
1. **Server:** Apache Tomcat was used to run the PHP and JSP code
2. **Database:** MySQL
3. **IDE:** Eclipse was used to write all of the code

Algorithms Used:

1. **Context Based & Exact Match Approach:** similarity between query terms and certain criteria of item
2. **TFIDF:** used to find important words in query and items
3. **Cosine Similarity:** used to determine how similar the query is to each item

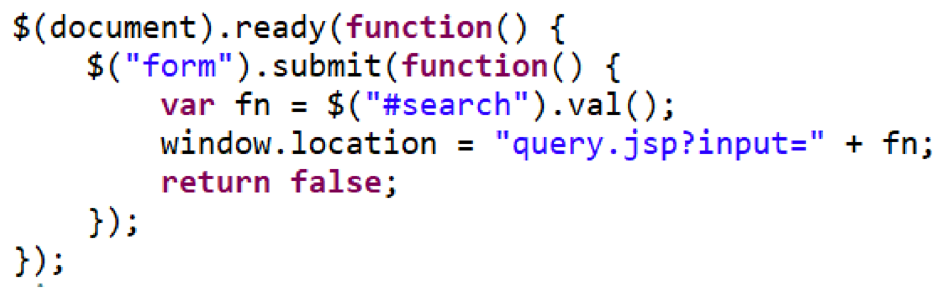
Search Bar:

In order to make the search bar functional we used JavaScript to send over the search bar input to the backend query page. Once the user submits the query, the JavaScript grabs the value using the ID of the search bar, and sends that value over to the JSP page by adding it to the end of its URL as shown in Figure 4 and changing to that page.



```
<div>
    <h1 id="main-title">Smart Shopping</h1>
    <form class="pull-right">
    <input type="text" id="search" placeholder="Search"
    </form>
</div>
```

Figure 3: Search Bar

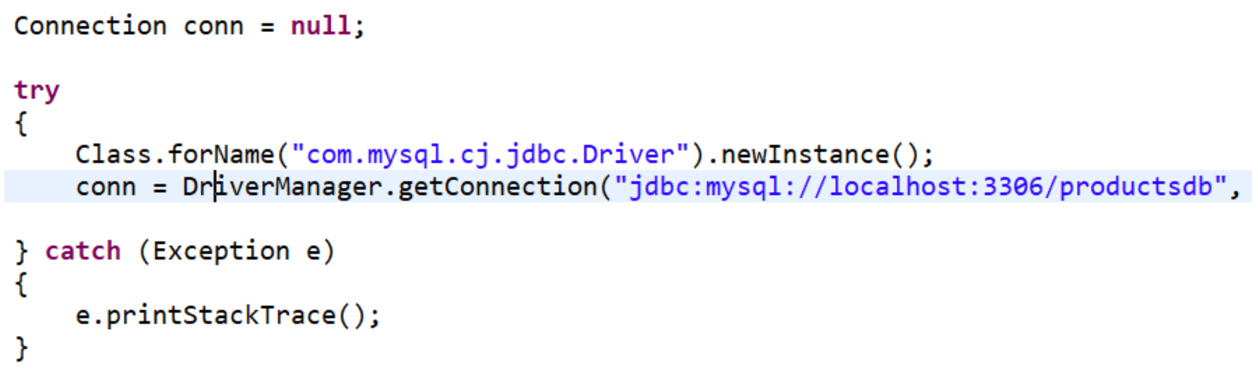


```
$(document).ready(function() {
    $("form").submit(function() {
        var fn = $("#search").val();
        window.location = "query.jsp?input=" + fn;
        return false;
    });
});
```

Figure 4: Search Bar JavaScript

<u>**JSP:**</u>

______For the backend computations we decided to use JSP. JSP is essentially an HTML page that incorporates Java. The JSP page does several things. One of the things it does is connect to the database. Without doing this part we wouldn't be able to compare the query to the items in the dataset. Once connected, we add all of the items in a list to be used by the rest of the program.



```
Connection conn = null;

try
{
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/productsdb",

} catch (Exception e)
{
    e.printStackTrace();
}
```

Figure 5: Database Connection

After the database, the program computes the TF-IDF for all of the products in our list and the query. In order to compute the cosine similarity later on, this step is necessary. To compute the TF-IDF for the products, the program uses the products list from the previous step. To do it for the query, the program must first know what the query is. This is where the URL generated from the search bar come into play. As shown in Figure 6 and 7, the program grabs the input parameter from the URL and sets it to a variable to be used for the rest of the query related computations.

```
localhost:8080/SearchBar/WebContent/query.jsp?input=shoes
```

Figure 6: URL generated by search bar JavaScript for the JSP page

```java
//Ask for search terms
String query = request.getParameter("input").replaceAll("[^a-zA-Z\\s]", "");
```

Figure 7: Code to get query parameter from URL

After computing the TF-IDF for both, we end up TF-IDF vectors for both the products and the query. Once that has been done, the program starts to compute the cosine similarity between the query and each of the items in our list. This is needed in order to generate the ranking list of the most relevant items. Once all the cosine similarities are calculated, we create a similarity matrix in the form of a HashMap. In this matrix we only add items with a cosine similarity higher than 0.250 for reasons explained previously. This HashMap is then sorted from highest similarity to lowest similarity as shown in Figure 8.

```java
LinkedHashMap<Integer, Double> sortedMap = new LinkedHashMap<Integer, Double>();

sortedMap = temp.entrySet().stream().sorted(Collections.reverseOrder(Map.Entry.comparingByValue())).collect(
    Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
```

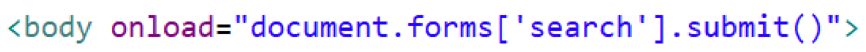
Figure 8: Sorting the ranking list in order of highest similarity to lowest

From here, we add all of the item IDs in the ranking list into an array. We then use that array to generate a URL for the search result page. This step is similar to one of the earlier steps regarding the search bar and is necessary so that the search result page knows which items to display. We generate the URL by recursively submitting multiple inputs within a single form with the value as the item ID using the GET method as shown in Figure 9.

```jsp
<form action = "searchresult.php" method = "get" name = "search">
<% for(int i = 0; i < itemNum.length; i++) { %>
    <input type="hidden" name="item<%out.print(i);%>" value="<%out.print(itemNum[i]);%>">
<% } %>
</form>
```

Figure 9: Generate item ID URL for the search result page

In order to make sure that the user does not see all of things happening in the background, we create the inputs with the type “hidden” so it’s not seen. We also make sure that the inputs are submitted as soon as the query page is loaded so it seems as if the website is skipping over that page when in reality it is has computed everything and submitted it. This is shown in Figure 10. The resulting URL is shown in Figure 11.

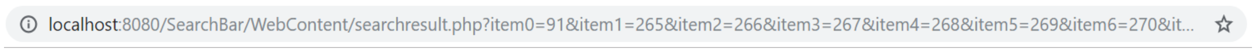


```
<body onload="document.forms['search'].submit()">
```

Figure 10: Submit search result form on load

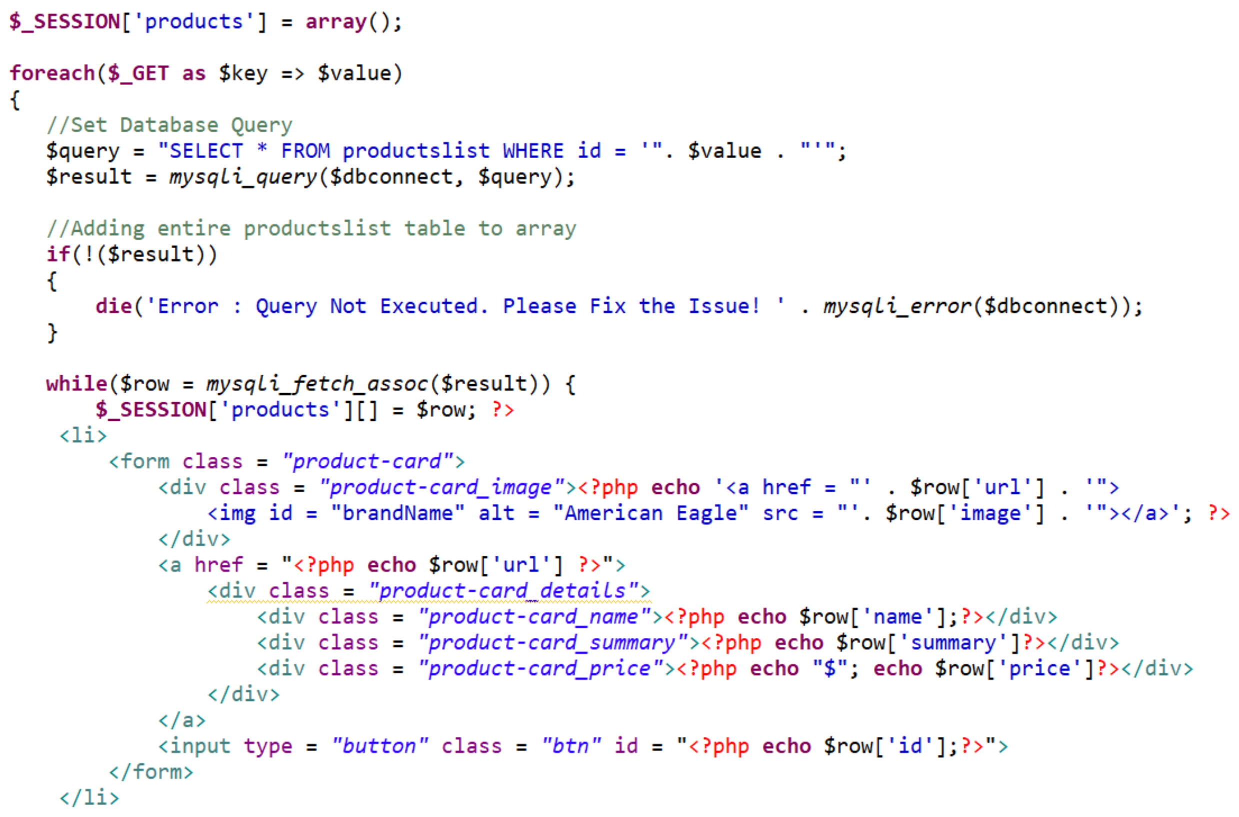
Search Results Page:

For this page we use a combination of PHP and HTML to display all of the results. Once the query page submits all of the forms the URL shown in Figure 11 is created. We use PHP to recursively get each of the parameters. For each parameter, we then retrieve all of the information from the database and then display it using a combination of PHP and HTML as shown in Figure 12.



localhost:8080/SearchBar/WebContent/searchresult.php?item0=91&item1=265&item2=266&item3=267&item4=268&item5=269&item6=270&it...

Figure 11: Item IDs URL



```
$_SESSION['products'] = array();

foreach($_GET as $key => $value)
{
   //Set Database Query
   $query = "SELECT * FROM productslist WHERE id = '". $value . "'";
   $result = mysqli_query($dbconnect, $query);

   //Adding entire productslist table to array
   if(!($result))
   {
       die('Error : Query Not Executed. Please Fix the Issue! ' . mysqli_error($dbconnect));
   }

   while($row = mysqli_fetch_assoc($result)) {
       $_SESSION['products'][] = $row; ?>
    <li>
        <form class = "product-card">
            <div class = "product-card_image"><?php echo '<a href = "' . $row['url'] . '">
                <img id = "brandName" alt = "American Eagle" src = "'. $row['image'] . '"></a>'; ?>
            </div>
            <a href = "<?php echo $row['url'] ?>">
                <div class = "product-card_details">
                    <div class = "product-card_name"><?php echo $row['name'];?></div>
                    <div class = "product-card_summary"><?php echo $row['summary']?></div>
                    <div class = "product-card_price"><?php echo "$"; echo $row['price']?></div>
                </div>
            </a>
            <input type = "button" class = "btn" id = "<?php echo $row['id'];?>">
        </form>
    </li>
```

Figure 12: Retrieve relevant item information and display it

EXPERIMENTS & RESULTS

Evaluation Method

Algorithms Used:

1. **Precision:** measures the percentage of mostly relevant items retrieved

$$Precision = \frac{Relevant\ Retrieved}{Total\ Retrieved}$$

2. **Recall:** measures the percentage of relevant items retrieved from entire corpus

$$Recall = \frac{Relevant\ Retrieved}{Total\ Relevant}$$

3. **F-Measure:** a measure of accuracy that takes into account both precision and recall

$$F - Measure = \frac{2RP}{R + P}$$

Initial Evaluation:

Initially, when we were first developing our cosine similarity algorithm we only used the summary of each item as our criteria. That means that we checked the similarity between the query and the summary of each item in order to retrieve the most relevant ones. During our evaluation of this method we found that the average recall was on the higher side, average precision on the lower, and average F-Measure on the lower side as well.

Breakdown:

1. **High Recall:** this means that our algorithm was able to correctly retrieve most, if not all of the relevant items related to the query.
2. **Low Precision:** this means that although we were able to retrieve close to all of the relevant items, there were also a lot of irrelevant items that were retrieved as well.
3. **Low F-Measure:** this means that our algorithm is not very accurate and needs to be improved. In this case the precision needs improvement.

	Similarity Criteria: Summary						
Query	Actual Relevant Items	Relevant Items Retrieved	Irrelevant Items Retrieved	Total Items Retrieved	Precision	Recall	F Measure
leather boots with zip	23	23	42	65	**0.354**	**1**	**0.523**
fur coat	6	6	18	26	**0.231**	**1**	**0.375**
long blue pants	19	13	4	17	**0.765**	**0.684**	**0.722**
short hooded jacket with pockets	30	30	57	87	**0.345**	**1**	**0.513**
bottoms	75	1	0	1	**1**	**0.013**	**0.026**

Avg Precision: 0.539 -- Avg Recall: 0.739 -- Avg F-Measure: 0.432

After Initial Evaluation:

After our initial evaluation we made changes in our similarity criteria. The second time around we included the name, brand, and type of each item to our similarity criteria in addition to the summary. In addition, we also added a cut off on the items that were shown. Any item with a cosine similarity below the cut off score would not be displayed. We discuss our thought process while making these changes later on in the report. During our evaluation this time around we found that the average recall, precision, and F-Measure were all higher than before.

Breakdown:

1. **High Recall:** this means that our algorithm was able to correctly retrieve most, if not all of the relevant items related to the query.
2. **High Precision:** this means that the number of irrelevant items retrieved was lower than before.
3. **High F-Measure:** this means that our algorithm is much more accurate in retrieving items relevant to the query.

	Similarity Criteria: Summary, Name, Type, Brand						
Query	**Actual Relevant Items**	**Relevant Items Retrieved**	**Irrelevant Items Retrieved**	**Total Items Retrieved**	**Precision**	**Recall**	**F Measure**
leather boots with zip	23	22	4	26	**0.846**	**0.957**	**0.898**
fur coat	6	6	14	20	**0.3**	**1**	**0.462**
long blue pants	19	8	1	9	**0.889**	**0.421**	**0.571**
short hooded jacket with pockets	30	19	13	32	**0.594**	**0.633**	**0.613**
bottoms	75	75	0	75	**1**	**1**	**1.000**

Avg Precision: 0.726 -- Avg Recall: 0.802 -- Avg F-Measure: 0.709

Discussion:

During our initial evaluation we realized that our similarity criteria of summary alone was flawed for a few different reasons. One of the reasons it was flawed is because the summaries of the items weren't very detailed, they were all very vague descriptions of the items. Not only that, the structure of the descriptions varied from brand to brand. Since our cosine similarity algorithm was based off of the Exact Match approach, this meant that if the summary of a relevant item didn't contain at least one of the words in the query then it would not be retrieved. In order to address this issue we added the name, brand, and type of each item to our criteria as well. We changed our algorithm so that it would compute the cosine similarity between the query and each of the parameters in the criteria, and then use the average to create the ranking list of relevant items. This method also addressed another issue that we saw. The issue was that there were times when an irrelevant item was ranked significantly higher than a relevant item because of the number of words from the query it contained in its summary.

Although we were able to improve the accuracy of the ranking list, we found that the number of irrelevant items being retrieved was about the same as the first time around. In order to address this issue we first checked the cosine similarity of all the items in the ranking list. By doing so, we found that the cosine similarity of most, if not all, of the irrelevant items fell below a certain number, 0.250. Using this information we decided to only display the items whose cosine similarity was higher than 0.250.

Although the average precision, recall, and F-Measure during the second evaluation was higher than the first, individually that wasn't the case every time. In terms of the precision for all of the queries, they were all higher than the ones from the previous evaluation. This means that the algorithm was able to retrieve mostly relevant items. However, the same cannot be said for recall. Individually, the recall for the second evaluation went down in some cases. This means that even though the algorithm was retrieving mostly relevant items, it wasn't able to retrieve all of them. This was most likely because of the cut off that we placed for the cosine similarity. Although this is a slight setback, the higher precision makes this a good trade off.

CONCLUSIONS

The Smart Shopping Website was designed to provide a website that would make searching, viewing and the selection of products easier. The goal of the website was to make a person's shopping experience more convenient by pooling products from different brands all on one site. While we were able to achieve that goal, there was more that we could've done to make it even better. Keeping that in mind, the goal for this project was to make the already convenient experience, even more convenient by expanding on the search function of the website. Overall, we believe that we were able to achieve that goal for this project. In the end, there were some things that did work and some things we think could've worked better.

What Worked

Overall, we were able to get the search bar function to work in that the search bar was able to return a list of items. To take it a step further, our algorithm was able to return a list of items mostly relevant to the query. Another thing that worked was getting the HTML pages to interact and communicate with the JSP pages. The JSP page is where we had the cosine similarity algorithm, that is where most of the work happens so it was very important to have that page be able to send and receive information from other pages.

What Didn't Worked

All in all we were able to achieve all of the goals that we set during the beginning of this project. While there wasn't anything that didn't work, there are some things we would like to improve or include in the future.

Future Work

One of the limitations of our project was the approach that we chose. Since we chose an Exact-Match approach that meant that the algorithm would only return items if its parameters contained at least one of the words in the query. This also meant that items of a similar nature to the query would not be returned. For example if a user searched "shirt" then "t-shirt" results would not show. For the future we would like to address this issue by introducing Collaborative-Based Filtering or by using a combination of Exact-Match and Concept-Based Approach. Another one of our limitations was the dataset. In the future we would like to create our own web crawler to extract more items and from more brands.

Team Contributions

1. **Aemun:** created the backend search algorithms using JSP and displayed results using PHP
2. **Fernanda:** created the proposal presentation, final presentation, and final report, and worked on the JavaScript for the search bar
3. **Neha:** created the entire front end of the website using HTML and CSS and set up the database