Building a Best Buy Store Interactive Webpage Using Django

Morgan Brockman
University of Idaho
Moscow, Idaho, USA
broc7795@vandals.uidaho.edu

Ian Fleming
University of Idaho
Moscow, Idaho, USA
flem8015@vandals.uidaho.edu

ABSTRACT

We set out to apply Database Structure concepts in the practical setting of building an interactive website. To do this, we used Django, the popular python-based web framework. This involved using the SQLite3 database, Django's default included database, and a good option for portability, though most database interaction was done through Django-provided functions. While Django simplified the process of interaction with the database directly, freeing up focus for website polish and features, a majority of our time was spent learning to work with Django and its functionality.

CCS CONCEPTS

• **Networks** → *Naming and addressing*; *Network management*; • **Software and its engineering** → *Object oriented languages*; **Extensible languages**; *Classes and objects*; *Inheritance*; **Modules / packages**; *Recursion*; **Application specific development environments**; *Object oriented frameworks*; • **Information systems** → **Relational database model**; *Data access methods*; **Database views**; Integrity checking; *E-commerce infrastructure*; **Resource Description Framework (RDF)**; **Question answering**; *Graph-based database models*; *Web Ontology Language (OWL)*; • **Computing methodologies** → **Natural language processing**; **Knowledge representation and reasoning**.

KEYWORDS

Web Development, Web Framework, Django, Python, SQLite3, HTML5, CSS, Visual Studio Code, GitHub, Database Systems, Discord

ACM Reference Format:
Morgan Brockman and Ian Fleming. 2018. Building a Best Buy Store Interactive Webpage Using Django. In *CIKM '22: 31st ACM International Conference on Information and Knowledge Management.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 DEVELOPMENT TOOLS

The following are technical descriptions of the various tools that were utilized in the development of this project

1.1 VSCode

We both made heavy use of VSCode, a popular, fully featured, heavily-extensible text editor designed for a wide range of programming situations. The most obvious feature utilized was VSCode's syntax highlighting and auto-completion. The program scrapes your project's files, imported packages, and relevant classes, and provides context-aware auto completion suggests, or highlights relevant variables. The program also features many productivity-increasing text-editing features, such as split view windows, auto and bulk indentation, multi-line editing, and much more. Another huge source of functionality is VSCode's extensibility. Through first party and community extensions, nearly endless functionality can be added to VSCode's application, such as in-app git control, LaTeX editing, viewing, and building, and more to be covered in the following subsections.

1.2 git Version Control

The version control software git allows developers to track changes made to their software verbosely, robustly, and indefinitely. Every time me made a change or added a feature to our app, we recorded the changes as well as a message describing the change using git. In doing so, git allowed us keep close track of the progress of our development, and exactly what changes where made when. It also allowed very simple reversion to previous changes, when the situation arose.

1.3 Github Collaboration Service

Not entirely separate from git, but a service all on it's own, is Github. Github's git repository functionality allowed us to combine git's version control features with a remote repository, allowing us to develop the project on two separate machines, at two separate locations. Whenever a change is tracked with git, it must be stored in the Github repository to truly be tracked. Once this is done, the repository acts as an intermediary between us, managing our changes to the code and the times they were made, keeping track in order to merge our changes and share them between us.

1.4 Discord Communication Service

Almost all of our communication was done over the Discord communication app. Having an easily accessible go-to method for communication is a necessity for any collaborative work. Discord's ability to communicate via text, speak over voice calls, share files, and even share real-time code manipulation was all essential to our ciollaborative development.

1.5 VSCode Database Viewer

As mentioned previously, VSCode offers a plethora of extensions to enhance its featureset. One such extension is any of several database viewer extensions. These extensions allowed easy, simple, real-time viewing of our application's database, as well as the database's structure and datatypes of every field.

1.6 dbdiagram.io Database Diagram Builder

1.7 Django Development Server

In the use of the Django Web Framework, our app and website can be locally hosted from our own machines. This enables us live feedback for any changes we make on multiple axes. This is only possible if we first host a Django virtual environment. The purpose of a virtual environment is quite tactful as it prevents permanent errors from blooming and the ability to manipulate the state machine in anyway without affecting one's machine or static environment. In addition, a benefit of this is the detailed error pages generated by Django for the back-end and its error-handling.

2 DEVELOPMENT LANGUAGES

The following are technical descriptions of the programming languages this project was built off of.

2.1 Python

The Django Web Framework utilizes the Python language for all of its back-end development and implementation. We used Python version 3.8.10. Python proved to be straightforward to learn and to utilize as the syntax is very light and the libraries are easy to use. The only downside in using Python is its performance, but our project didn't seem to experience such issues.

2.2 SQL

Our project utilized the SQLite3 framework for the data base implementation. This framework proved the most effective and reliable for our project for it is lightweight and functions quite well in Python and Unix environments.

2.3 HTML

HTML5 was utilized for the front-end development of our project. As a styling/design language, it was relatively straightforward to learn and program. Our project utilizes multiple pages in HTML that create the overall structure as it may appear on the Internet. Furthermore, the use of HTML enabled the possibility of back-end functionality such as user input and feedback.

2.4 CSS

The latest version of CSS is responsible for graphical styling and the creation of visual hierarchy amongst all of our web pages in our project. We have a primary style sheet that essentially resets the styling for the main elements prior to anything. This is a good coding practice. The rest of the pages utilize their own style sheet for organization sake and the prevention of conflict. All of our styling in CSS was coded from the ground-up and was pure.

3 DEVELOPMENT

The development process was very simply structured into three rather distinct parts. First, there was the design of the database that would be running under the hood of our website. Then, there was the development of the website, which would be separated pretty cleanly between the back-end development, working with Django's various functions and features, and the front-end development, working with the HTML, CSS, and Django's templating features. We both worked together on the design of the database. Morgan primarily focused with Django's functions, while Ian worked with Django's HTML templating system and the website's front-end.

3.1 Database Design

Our database design was relatively simple with how many tutorials there were online for similar websites. We followed Mozilla's tutorial for building a website for a library using Django, and it's database design mapped pretty well to ours.

All products are contained in the Product table. Each product has its own unique UUID, a 36 character uniquely-generated identifying string, with a special datatype built into Django. This serves as the table's primary key.

Within the Product table are several one-to-one foreign key relationships for the various features available. These are each stored in tables which simply contain an iterating id primary key and the name for the relevant feature. Doing this allowed us to make querying matching products very simple.

Aside from this there are several less important fields which are simply stored as their own relevant variables within the table itself. This allowed for more flexibility in storage, though it did make querying them a bit more involved.

We then used Mockaroo to generate data matching the tables and populate our database.

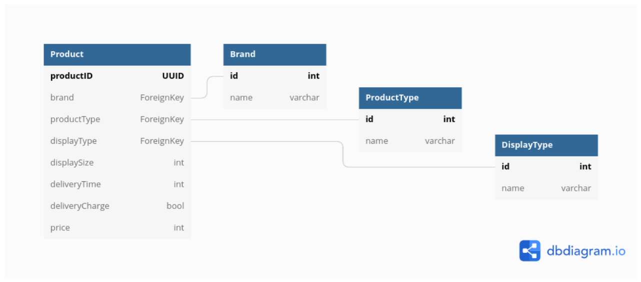


Figure 1: Relational database diagram for Best Buy website

3.2 Django

Much of our time spent on the project was learning how to work with Django's vast and extensible feature-set. For brevity, a simplified description of how building a website with Django works can break it down into the following three feature.

models.py

Rather than working with any particular database system directly, Django abstracts this work with its models feature. Using conventions defined by Django, the developer can design their database using more abstract database design concepts more directly. Tables are defined as classes, with variables belonging to that class being treated as forms within that table. These variables are extremely extensible

through Django's documentation, allowing the developer to do this such as: defining the primary key, linking another table as a foreign key, defining the forms data types (such as integers, bools, dates, or times), or even automatically generating standardized UUIDs (universally unique identifiers) salted through your MAC address, namespace hashes, or a plethora of other methods at the developer's choosing.
Once these models are implemented, Django handles building and modifying them on its own, regardless of the database one chooses. Switching databases is as simple as modifying a settings file and possibly installing the Django package for its support. Often, depending on the previous database implementation, data migration between databases is as simple as a Django command.

views.py
Paired with Django's models feature is the views feature. This allows the developer to define variables and functions, take GET and POST calls, manipulate both and pass the results to the generated HTML page. Along with all the functionality for creating dynamic web pages this allows by simply effectively extending HTML with Python and everything that offers, a particularly important feature is Django's object.filter() function. This effectively abstracts SQL queries from the previously defined models to more python friendly if statements and equality operators.
The process for dynamic web page deployment in essence becomes: deliver a web pages data with python, return any data through HTML POST, retrieve that data from models using python, and deliver the new web page with python.

HTML Templates
The HTML templates provided by Django were organized in such a fashion where the user is given a basic/generic page responsible for the general HTML/head data and styles that the web pages will be governed by. Any other web pages (including the index.html) created by the user will inherit or extend from this basic/generic page. Embedded commands provided from Django are used in each web page to produce content based on the back-end functionality.

3.3 Website Design

Our project aimed to delivery visual simplicity alongside effective functionality. Our website was designed to be easily navigable and interactive. Our color schema mimicked that of Best Buy's alongside other contrasting elements. We used the Open Sans font family because it is visually pleasing and concise with its various font weights. We aimed to have elements on all pages to be equally spaced and centered for a professional and forward appeal. Furthermore, we added simple animations to certain elements (i.e. buttons) to add flair and personality. The front-end styling behind our website is pure CSS and not assisted with Bootstrap or any other applet alike. In short, we wanted to craft a website that was personable and simplistic while delivering the services our users want effectively.

3.4 Deployment and Hosting

Django bundles with it its own entirely self-contained development server. While this is primarily intended for development on a local machine, and while the practice does not necessarily follow best security practices, the development server is entirely extensible through only a few settings changes. For out particular project, we are simply using a domain Morgan happened to already own. Using Google Site's interactive settings, we simply pointed the desired subdomain to a specific port, forwarded that port on the development machine's local network, and the website is now fully accessible whenever the development server is running.

3.5 HTML Templates

In our use of the Django web framework, we utilized the provided basic/generic web page. Here, we included all of our style sheets and other various "head" data and information. A stylistic and functional navigation bar was added to this page so that it would be present for all other pages that extend from it. Here on, we created the rest of the necessary pages our project would need and had them extend from our basic/generic. Each had their own style sheet loaded statically from the page as well. This structure worked most effectively and allowed the front-end development much more seamless and organized.

4 DATABASE

We used SQLite3 to implement the website's database, as it works well with Django out of the box, and its portable nature made collaboration using git much easier than many alternatives.

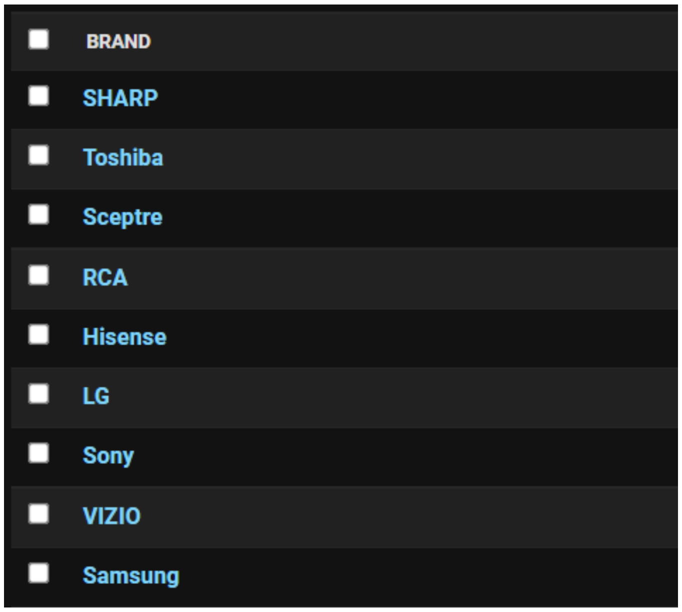
Our database is made of of four tables: a larger product table, and three short feature tables.

Pictures provided are from the website's admin interface, where table data can be graphically viewed or modified.

4.1 Brands

Our Brand table contains only an automatically iterating ID integer as its primary key and a variable character field as its name.

For each brand or make of product available, an entry is stored in this table, and then referenced by a one-to-one foreign key in the product table. This allows easy listing and searching of all the available product brands.



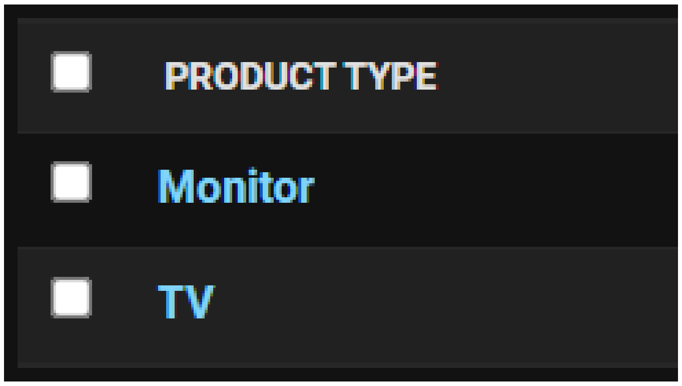
BRAND
SHARP
Toshiba
Sceptre
RCA
Hisense
LG
Sony
VIZIO
Samsung

Figure 2: Picture of the Brands table.

4.2 Product Types

Like the Brand table, the Product Types table contains only an automatic iterating ID integer as its primary key and a variable character field as its name.

This table contains an entry for every type of product available in the store. Like the brands, the product table may contain a one-to-one foreign key referencing a particular type of product in this table.



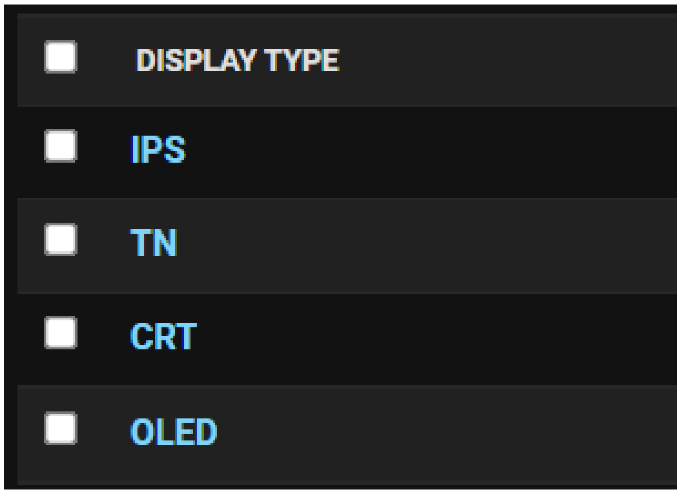
PRODUCT TYPE
Monitor
TV

Figure 3: Picture of the Product Type table.

4.3 Display Types

Like the previous two tables, the Display Types table contains only an automatic iterating ID integer as its primary key and a variable character field as its name.

Each entry in this table represents a different type of display technology available in the store. Like the other two feature tables, the product table may contain a one-to-one foreign key referencing a single type of display technology listed in this table.



DISPLAY TYPE
IPS
TN
CRT
OLED

Figure 4: Picture of the Display Type table.

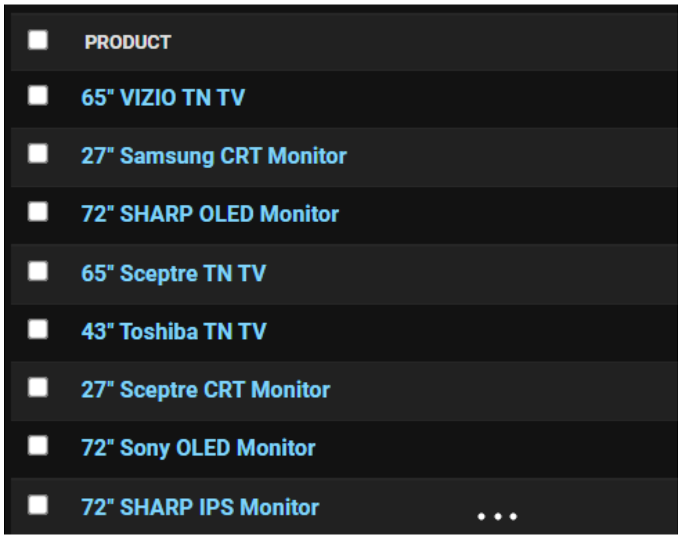
4.4 Products

Our products table is where the bulk of the information is stored. For each product available in the website, there is an entry on this table.

Each entry contains a unique UUID (Universally Unique Identifier) acting is the entry's primary key. The UUID is a special 36bit Django data field, automatically generated at instantiation whenever a product is added to the table.

Each entry also contains three foreign keys, one for each feature which is common to many products: Brand, Product Type, and Display Type. The available data for each of these features is stored in separate feature tables, as described previously. The foreign keys each mark one-to-one relationships with a single product in each of their feature tables.

The other fields of the product table contain more specific information on a given product. The displaySize field denotes in inches the size of the product's display, if it has one. This is simply stored as an integer. The deliveryTime field denotes in days how long the product's delivery time is estimated to take. This is also a simple integer field. The deliveryCharge field is a boolean denoting whether or not a given product charges paid delivery. The price field self-evidently stores the product's price. This is again a simple integer field denoting the price in USD.



PRODUCT
65" VIZIO TN TV
27" Samsung CRT Monitor
72" SHARP OLED Monitor
65" Sceptre TN TV
43" Toshiba TN TV
27" Sceptre CRT Monitor
72" Sony OLED Monitor
72" SHARP IPS Monitor
...

Figure 5: Picture of the Product table.

4.5 Mockaroo

Mockaroo is a website service that provides the user the ability to create mock data in various formats for a database. Our project utilizes and is dependent on mock data for the functionality of the database as a whole. We had it so multiple columns would be present to function as characteristics or features of a product and the values of each would vary and generate randomly.

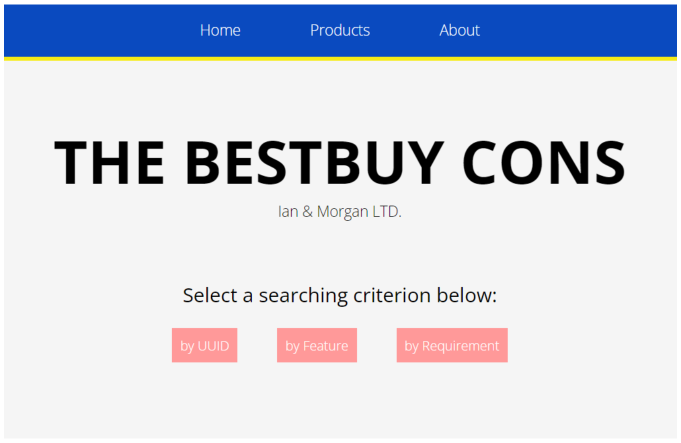
5 FUNCTIONALITY

The following sections will describe the front-end implementations and the back-end implementations of each page.

5.1 Home Page

Functionally, the front-end development of the home page is quite simple and straight-forward. A basic splash/welcome page is produced alongside three buttons. These three buttons, labeled: "by UUID", "by Feature", and "by Requirement" are responsible for redirecting the user to the page of the particular searching criterion they desire.

These buttons feature Django HTML Template function calls which automatically handle URL Mapping to the correct view function, taking the user to the given URL and generating for them a dynamic HTML page, depending on the selected search feature.



THE BESTBUY CONS

Ian & Morgan LTD.

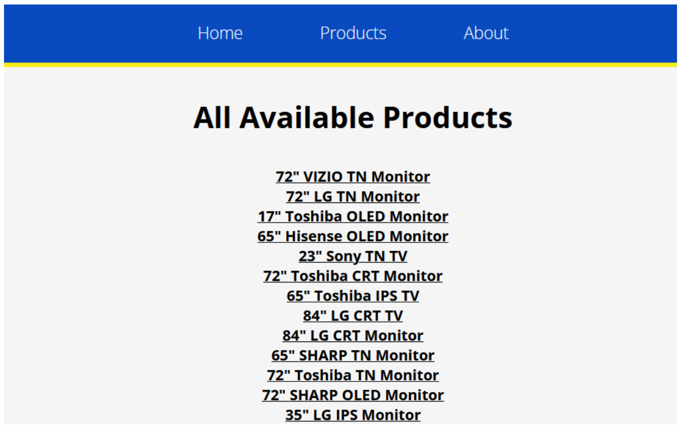
Select a searching criterion below:

Figure 6: Picture of the home page.

5.2 Products

The functionality behind this page simply loops through all the items present in the database and prints each as a list item into HTML. The sole purpose of this page is to allow the user to view all of the available products we sell.

A set of objects is defined within the corresponding python view function. The HTML then uses a provided HTML Template looping function to print each of the provided objects. By including all objects in the database, all objects are listed on this page.



All Available Products

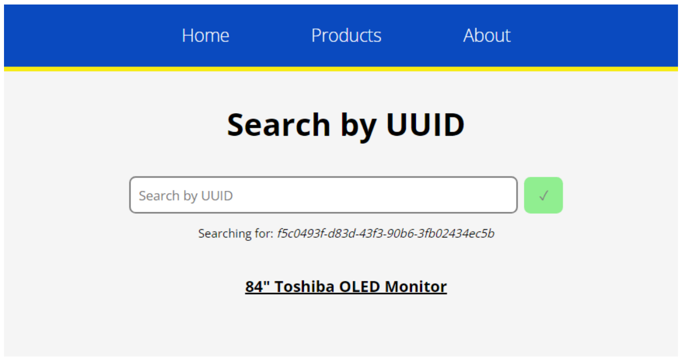
72" VIZIO TN Monitor
72" LG TN Monitor
17" Toshiba OLED Monitor
65" Hisense OLED Monitor
23" Sony TN TV
72" Toshiba CRT Monitor
65" Toshiba IPS TV
84" LG CRT TV
84" LG CRT Monitor
65" SHARP TN Monitor
72" Toshiba TN Monitor
72" SHARP OLED Monitor
35" LG IPS Monitor

Figure 7: Picture of the product page.

5.3 by UUID

The front-end functionality of this web page uses two key elements. An input element and a submit button. The input element is a type search, which enables the user to input anything they desire. In this case, the input is looking for the UUID of the user's desired product. The button is of type submit and is responsible for sending the user's input to the back-end. In order for a match to be found, the user must type in the exact UUID in order for results to show, else, the database will return empty.

The UUID is simply passed back to the view, this time through a POST call containing the desired UUID. Using python in this page's view function, the application performs a filter() query for any products with a matching UUID then sends the matching products to the HTML template to generate the new page.

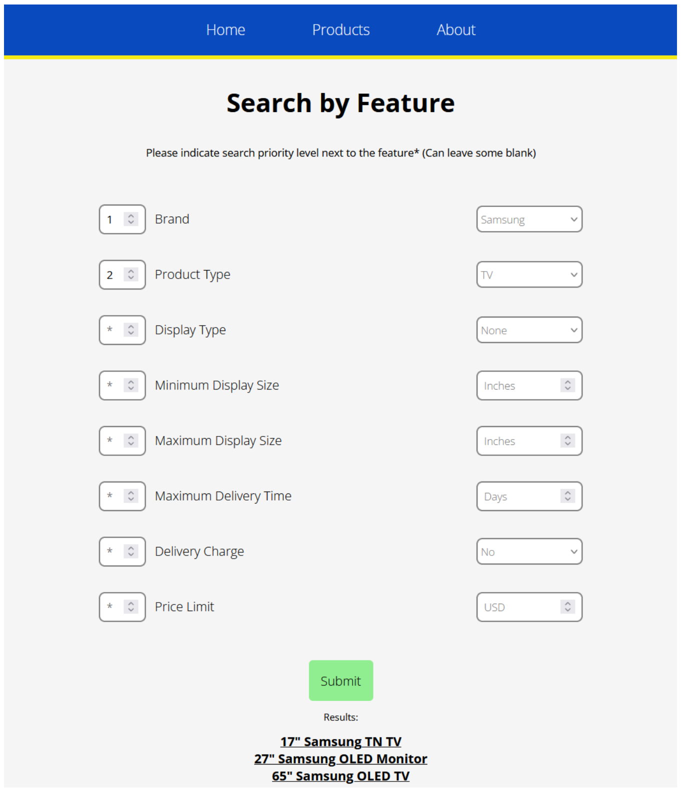


Search by UUID

Searching for: *f5c0493f-d83d-43f3-90b6-3fb02434ec5b*

84" Toshiba OLED Monitor

Figure 8: Picture of the UUID page.



Search by Feature

Please indicate search priority level next to the feature* (Can leave some blank)

1	Brand	Samsung
2	Product Type	TV
*	Display Type	None
*	Minimum Display Size	Inches
*	Maximum Display Size	Inches
*	Maximum Delivery Time	Days
*	Delivery Charge	No
*	Price Limit	USD

Results:

17" Samsung TN TV
27" Samsung OLED Monitor
65" Samsung OLED TV

Figure 9: Picture of the best match page.

5.4 Best Match

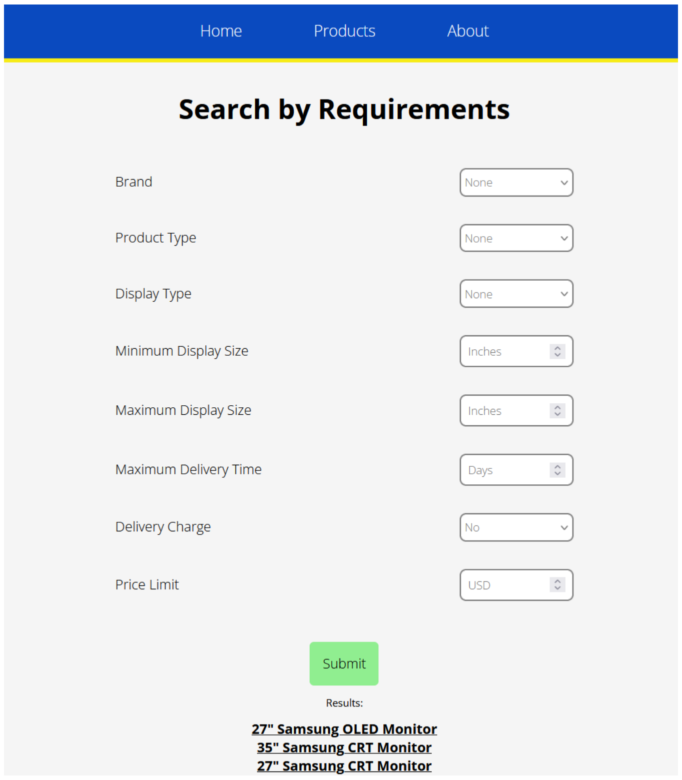
The front-end functionality of this web page is quite dense and rigidly organized. It can be characterized into eight sections or input fields. Four of which are drop-down menus and the rest are input text boxes. Each section is labeled by the feature or characteristic of the product that's to be searched. Furthermore, before each section label are small input boxes with a range of one to eight responsible for dictating the priority of each feature. There is a submit button to send this data to the back-end.

Due to the unique nature of this feature, the back-end functionality is quite complicated. Once receiving the data provided by the search submission, significant work is done parsing, reorganizing, and restructuring the data to be more workable. Once the different feature queries are all separated and linked up with their corresponding priority values, a chain of database queries are performed. First, the highest priority query is performed and stored. Then, the query of the next priority is performed, then a check is performed to determine if any results were found. If not, the search is cut short and the last set of results are returned. Otherwise, it continues down the priority list until all queries are performed or an empty query is returned. It then prints out the matching products, similarly to the previous page.

5.5 by Feature

The front-end functionality of this web page is very much similar to that of the 'Best Match' page but the back-end functionality is quite different. There are the eight sections alongside their feature labels and accompanying input types. There is no priority input in this criterion for the features. There is a submit button to send this data to the back-end.

On the back-end this feature works very similarly to the Best Match search. Depending on the criteria selected, database queries are chained one after the other, shrinking the pool of results for every criteria. No special data processing is needed, however, as there is no dynamic priority tied to each search criteria, and the queries can simply be chained one after another with no regard for order.



Brand	None
Product Type	None
Display Type	None
Minimum Display Size	Inches
Maximum Display Size	Inches
Maximum Delivery Time	Days
Delivery Charge	No
Price Limit	USD

Results:

27" Samsung OLED Monitor
35" Samsung CRT Monitor
27" Samsung CRT Monitor

Figure 10: Picture of the feature page.