BestBuy: A Versatile Platform for Digital Commerce

Spencer Butler
University of Idaho
Moscow, Idaho, USA
butl2691@vandals.uidaho.edu

ABSTRACT

This project is the implementation of an e-commerce system with binding contracts and three different ways for customers to be matched with products. This project uses the Laravel framework for back-end, and the Bootstrap and JQuery frameworks for front-end. Matching methods are implemented as ways of searching. Contract entry is implemented as offers which must be mutually accepted before either party can see details about the other party. The database is populated with data generated by custom data generation functions which use the PHP Faker library for generating certain values.

CCS CONCEPTS

• **Information systems** → **Database views**; *Database query processing*; *Relational database model*; **Online shopping**.

KEYWORDS

Query processing; database views; priority axis searches; database design; online shopping

ACM Reference Format:
Spencer Butler. 2022. BestBuy: A Versatile Platform for Digital Commerce. In *Proceedings of WSDM '23: The 16th ACM International Web Search and Data Mining Conference (WSDM '23).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The purpose of this project was the creation of an e-commerce system according to provided specifications. The specifications included mutual anonymity prior to entering binding contracts and three different ways for customers to find products. In the course of implementing these specifications, making the system product-agnostic rather than focused on any particular type of product was chosen as a priority.

The system created supports registration of users as either customers or vendors. After registration, vendors can list products they sell and customers can list products they own. When listing products for sale, vendors can specify their desired price. Customers can search for products being sold, using three different search methods each corresponding to a specification-designated method. Customers can take any product in those search results, and offer a specific payment for that product. Vendors can accept or reject offers.

Additionally, the system is populated with data automatically generated across several specific product categories.

2 TOOLS USED

On the back-end, the system uses the Laravel PHP framework [3]. Laravel was chosen in large part because it was featured in the lab assignments for the class. Laravel also provides easy support for inheritance-based templates for webpages, which was used to give all pages on the site a common structure. Additional features used include direct control over routing and replying to incoming HTTP requests, query-building tools, and facades for common problems like user authentication.

On the front-end, the system uses the Bootstrap framework for CSS and Javascript [4], and the JQuery framework for Javascript [1]. Bootstrap provides convenient and effective ways to achieve a clean and usable interface with minimal effort, which was desirable so that the primary focus of the project could be creating the functional components of the system rather than the visual components. It also provides easy access to things like popup displays via modals. JQuery provides simple ways to manipulate the contents of pages client-side, using Javascript. It also greatly simplifies making AJAX requests and placing the contents returned by those requests into the page.

The database management system used was MySQL, but all queries were written in PHP via Laravel's query building tools. These queries have different syntax than SQL, but support a similar range of semantic meanings. Additionally, the system could be swapped to use a different underlying database management system via changing Laravel's environment configuration without having to change any of the queries.

```
$results = $results->whereExists(function ($query) use ($name, $val, $table) {
    $query->select(DB::raw(1))
          ->from($table)
          ->whereColumn($table . '.product_id', 'sells.product_id')
          ->where($table . '.name', $name)
          ->where($table . '.value', $val);
});
```

Figure 1: Query from the exact search implementation

The system uses an Apache server solution. Both Apache and MySQL were bundled as part of the XAMPP package [2]. This package also provided the PHP MyAdmin interface for working with the MySQL database.

3 DATABASE STRUCTURE

The database contains multiple tables, primarily used to represent users, products, and the relationships between them. All tables have a numeric, auto-incrementing ID used as a primary key, and

WSDM '23, February 27–March 3, 2023, Singapore

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

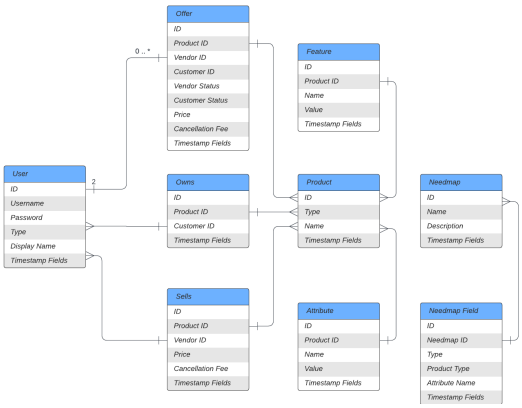

Figure 2: Diagram representing the tables of the database

timestamp fields for the time of creation, last update, and soft-deletion of a record.

3.1 Users

The Users table keeps track of the basic information about each registered user of the system. In addition to the standard columns, it contains columns for each user's username, password, type, and display name. Usernames must be unique as they are used to login to the system. Type can be either customer or vendor, which controls several aspects of how the user can interact with the system. The display name can be any text, and need not be unique. It is shown upon entering a contract with another user.

3.2 Products

The Products table stores the basic identifying information about each product. There are two columns used: name and type. The product's type is a general category, such as "Smartphones" or "Rental Apartments," while the name is an identifier specific to this product. This table does not store any further details about products.

3.3 Attributes and Features

Attributes and Features are the two tables used to store details about products. These tables are highly similar, differing only in the type of data stored. Each has three main columns: product_id, name, and value. Each record represents a single salient trait of a product. The product id is the product which the trait applies to. The name is a descriptor of the trait, such as "Weight" of a phone, "Bedrooms" of a house, or the "Author" of a book. The value is the actual value of that trait for this product. That is, the actual weight of that phone, the number of bedrooms that house has, or the name of the book's author. Attributes and Features differ in the type of the Value column. Features have numeric values, like Bedrooms, while Attributes have text values, like Author. Different operations are possible on Attributes than Features.

Due to the similarity between these two tables, an alternate approach would have been merging them into a single Details table. This table would have an additional column, type, which would be constrained to values such as "numeric" or "descriptive." The different logical treatment of Details of different types would be done by selecting only those records with the desired type, rather than by selecting records from Attributes or from Features. This approach would also have allowed the later addition of more types of product details, such as booleans.

These two tables are the largest in the database, owing to the fact that most products have multiple numeric traits and multiple descriptive traits.

3.4 Sells and Owns

Sells and Owns are the two tables that associate a product to a user. Sells associates a product to a vendor, while Owns associates a product to a customer. Each record in either table has an id of a user and an id of a product. Sells has two other columns, one to specify the vendor's desired price for the product, and one to specify the penalty for either party breaching a contract involving the purchase of that product from that vendor.

Sells and Owns both constitute many-to-many relations between Users and Products. A user may sell or own any number of products. Additionally, every product must be sold or owned by at least one user, but may be both sold and owned, and may be sold or owned by multiple users.

3.5 Offers

Offers is the table representing an open offer to purchase a product. It has columns for a customer, a vendor, the offer's associated payment and fee for breaching contract. Additionally, it has two columns for the status of the customer and vendor with regard to this offer, facilitating the acceptance or rejection of the contract, and the removal of the offer from the database once it has been fully resolved.

3.6 Needmaps

Needmaps is the table storing top-level information about individual types of need-based searches. Its columns contain a name and brief description for the search type. This information identifies the search to customers, while the information stored in corresponding Needmap Fields is used to resolve the search once the customer has selected it.

3.7 Needmap Fields

Needmap Fields is the table storing information used to resolve all need-based searches. It has columns for the needmap associated with each field, the type of product this field comes from, the name of the attribute this field covers, and whether this field is part of the left or right hand side of the needmap. A needmap field with product type "Houses" attribute "Square Feet" and type "lhs" will result in its needmap summing up the square feet attribute of all houses owned by the customer executing the search. It will then try to match this against products corresponding to values in "rhs" fields for the same needmap.

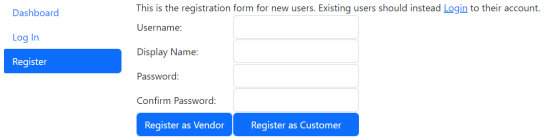

Figure 3: User registration interface

4 MAJOR PAGES AND FUNCTIONALITY

4.1 Login and Registration

For a visitor who is not logged in, only a few pages are accessible. The homepage directs them to either create a new account or login to an existing account. Both login and registration use the same form for vendors as for customers. No different details are needed at registration for the different types of accounts, and the type of the new account is determined by which of the two registration buttons was used to submit the request.

4.2 Dashboard and Listings

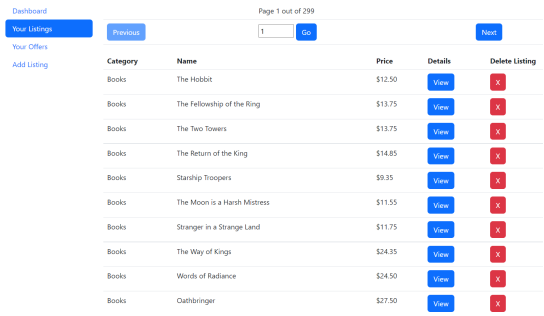

Figure 4: List of products sold by a vendor

When a visitor is logged in as a registered user, they are shown a dashboard with their display name, username, and type of account. They can then view their listings, which are the products sold by a vendor or owned by a customer. When a vendor views their sold products, they see a price value, while this is not visible for customers viewing their owned products. Either type of user can view the details of any product in their listings. They can also remove a product from their listings, and if it isn't in any other listings or in a currently open transaction, it will be removed from the database, along with all of its detail fields.

4.3 Add Listing

Both types of user can use a form to add a product to their listings. They first specify the category and name for the product. Vendors must additionally specify the desired price for the product. If any existing products match the specified category and name, they will be shown, and the user can view the details of those products and add them to their listings. Users can also choose to fully specify the details and create a new product. Every detail field must be directly specified as either a numeric or a descriptive detail. Descriptive detail values can be left blank. Blank detail values will be entered into the database with a value of Yes.

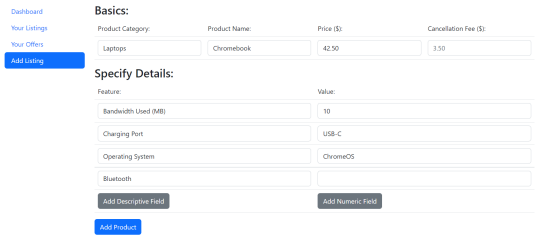

Figure 5: Interface for a vendor listing a new product for sale

4.4 Exact Search

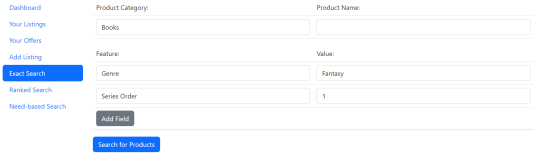

Figure 6: Interface for specifying an exact-match search

Customers can search products being sold. The first method for doing this is an exact match. The customer must specify the general category of the product. They can add additional restrictions, specifying any number of details the product must have. As they select more restrictions, the values suggested for further details are narrowed down, to only suggest values compatible with the already suggested values. However, it does not narrow down the names suggested for additional details, as all detail name inputs in a single form access the same list of suggestions. Once the customer is done building their search query, they can execute the search, and will be shown a list of all products which match all of the restrictions imposed. This list will be sorted by price.

4.5 Ranked Search

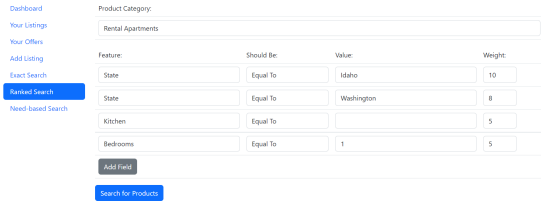

Figure 7: Interface for specifying a ranked-priority-match search

The second type of search is a ranked priority match. Customers must specify the category of product. The search will return all products within that category, sorted first by how well they match the user's specified criteria, and secondarily by price. The user can specify any number of criteria. Criteria specifications are the name of an attribute or feature, a comparison operator, a desired value, and a weight. All products which have a value for that detail are

checked against the desired value, according to the specified comparison operator. If the comparison is true, the specified weight for that criterion is added to a search-matching score for that product. This search-matching score is used to sort the products in the list returned by the search.

4.6 Need-based Search

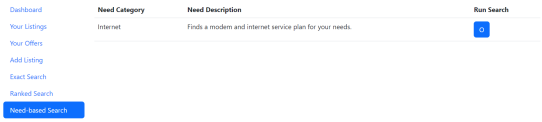

Figure 8: Interface for executing a need-match search

The third and final type of search operates on the needs of the customer, rather than requirements the user sets. It displays a list of all needmaps defined in the needmaps table, showing their name, description, and an option to execute each one. Executing a needmap will take all of the needmap fields corresponding to the needmap, and sum up all values of attributes from products the customer owns where the attribute is specified as a left-hand side field for the needmap. It will then take all attributes specified as right-hand side fields for needmap. For each right-hand side field, the search will return a list of products where their value for the attribute is greater than the sum of the customer's left-hand side values. These lists will be sorted by price. If the customer already owns a product of the type corresponding to a right-hand side field, with a value for the attribute greater than the sum of their left-hand side fields, no list of products will be returned for that right-hand side field. This is to allow customers to identify only products they are missing, when two or more types of products are used to fill a single need, such as internet service requiring a modem and a subscription.

4.7 Search Results

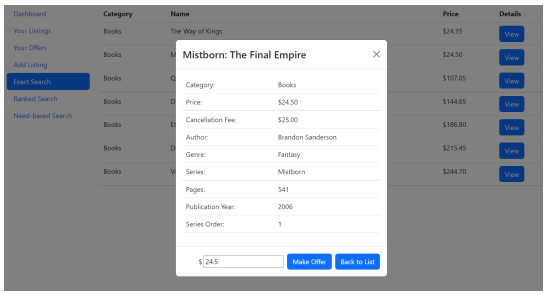

Figure 9: Details of a product being sold, with option to make an offer

Search results are given in the form of a list of products, very similar to the listings of products owned by a customer or sold by a vendor. Since search results only retrieve products which are being sold, all products have a price specified. The customer can view the full details of products. They have the additional option to offer a contract to the vendor selling a product. This offer will, by default, use the price asked for by the vendor, but the customer can specify a different amount of payment for the offer.

4.8 Offers

Once a customer has made an offer on a product being sold, the offer will be added to the list of offers for that customer, and for the vendor selling the product. The customer will not know who the vendor is, nor the customer the vendor. When a user views their list of offers, they can see the current status of each offer. This status is either "Waiting" if the offer is waiting for the other partner to take action, or "Action Needed" if the offer is ready for the current user to take action. The list of offers also shows the name of the product and the payment associated with each offer. By clicking a button to view details for an offer, the user can see the details of the product, and take any necessary action.

If one side rejects an offer, it will be removed from their list. It will remain in the other person's list marked as action needed, until they remove it.

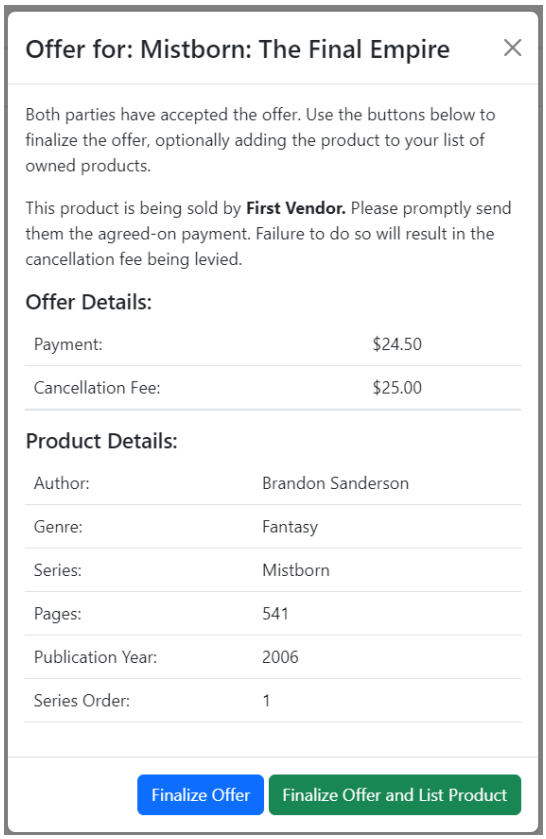

Figure 10: Details of an offer that has been accepted by the vendor

If both sides accept, the offer will still be visible in both people's lists. It will now display the details of the other user involved in the contract, facilitating the fulfillment of the contract. Currently, the only detail given is the partner's Display Name. A more robust contract fulfillment system would likely also show their shipping

address and link to a payment form. In addition to the extra details, the user will now be given a chance to dismiss the contract, removing it from their list of offers. The customer in a contract will also be given the option to immediately add the product to their list of owned products when they dismiss the contract.

Once both sides of an offer have removed it from their list, whether by dismissing a mutually-accepted offer or by acknowledging the partner's rejection of an offer, the offer will be removed from the database.

5 DATA GENERATION

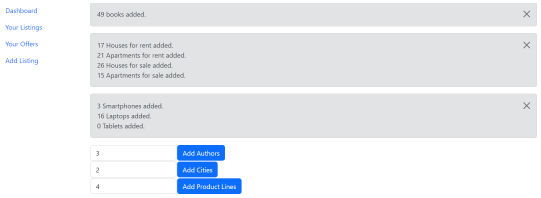

Figure 11: The admin-exclusive datagen page, with feedback from execution

Data generation was implemented through custom functions using PHP Faker. Three functions were made, each interfaced with through an unlinked page on the site, which can only be accessed by an authenticated user with the username admin. Each data generation function operates in batches, generating multiple related products at a time. For convenience, the data generation page allows specifying a number of batches, and that request will generate that many batches.

The first data generation function generates books by a specific random author. Each author has several genres they can write, and writes multiple things, where each thing is either a book or multiple books belonging to the same series. In this function, Faker is used to generate the names of authors, and lorem ipsum text for the titles of books and series.

The second data generation function generates housing in a single city. Each housing unit is either for sale or for rent, and either an apartment or a house. Depending on this, the unit has random traits, including size, bedrooms, and bathrooms. These traits influence the final price of the unit. In this function, Faker is primarily used to generate addresses for the names of individual listings, as well as cities and states for the detail fields of each batch of listings.

The third data generation function generates a product line of smartphones, laptops, or tablets. Various traits depend on the product type, and can change with progression from one item in the product line to the next. Year of release and price increase from one item in the line to the next. In this function, Faker is used to generate the names of companies to specify as a manufacturer, as an additional option beyond randomly using one of the manufacturers already listed in the database. It's also used to easily provide a floating point value for adjusting some of the product traits from one item in a product line to the next.

The focus with the custom data generation functions was to provide usefully coherent data. Generating books in batches of an author prevents each book from having its own, unique randomly generated author, and allows querying by author. Generating housing by city and technological devices as part of lines provides similar grouping to that data, allowing meaningful queries.

6 AREAS FOR FURTHER DEVELOPMENT

As a whole, the interface for the system is very minimalist and unrefined. Work could be done to design a more visually appealing and engaging interface. Additionally, more textual instructions and explanations on how the system works and how to interact with it would be a valuable improvement to the overall user experience of the site. Aside from this, there are multiple specific areas which could be worked on to improve the system. With a focus on technical aspects and capabilities, some of these areas are:

6.1 Technical Debt

Throughout the project's development, there were a lot of decisions made which do not scale well to the size of the project, and a lot of decisions which should be made for a project of this size that were not made. This poses problems for long-term development and maintenance of the project, and should be resolved if it were to become a long-term project.

Chief among the decisions that should be made is the creation of a set of standards for variable names, functions, and URLs. These standards should cover HTML form names, PHP, and Javascript. Following a proper set of standards would allow much quicker recognition of the purpose of a thing by its name, and prevent confusion when referring to different variables holding the same data. It would also facilitate interoperability of the system, where solutions used for one aspect could be applied to solve the same problem elsewhere, without having to change names and references between the two areas.

A decision which ought to be reconsidered is the division of routing functions across different Laravel controller classes. Early on in development, three controllers were created, one to respond to requests from unauthenticated users, one to respond to authenticated requests for complete webpages, and one to respond to Ajax requests. This decision was sensible for the scale of the system when the decision was made, but the number and length of functions in these controllers has increased dramatically. To facilitate ongoing development, the Ajax controller in particular should be split into multiple controllers, each corresponding to a distinct type of request.

The Features and Attributes table should also be merged into a single table, with an additional column to differentiate between types of details. Doing so would improve the ease of development of future system features, especially any addition of more types of product details.

6.2 Laravel Database Integration

In addition to general correction of technical debt, there is a lot of room for improvement in the integration of the database with the Laravel framework. Laravel provides the Eloquent Object-Relational Mapper to interact with relational databases in an object-oriented manner. Properly setting up the Eloquent models, with specifications for relationships between models, would allow future queries

to be written in more straightforward and maintainable ways. Additionally, Laravel Migrations should be set up, to facilitate importing the database schema with a single command, rather than having to manually import it via a tool like PHP MyAdmin.

6.3 Query Caching

Pagination of search results is supported by storing the most recent search query. Changing to a different page of results repeats the query, with an additional page parameter set to control what page is returned. This is a straightforward and simple way to implement pagination, but requires repeating the query calculation for every page swap. As the size of the database grows, queries will increase in calculation time, and the delay in changing pages is likely to become a noticeable hindrance to user experience. To reduce the page change time, recent queries could have their results cached, and the page change requests could return results from that cache, rather than recomputing the entire query calculation.

6.4 Client/Server Division of Responsibility

Currently, there is a lot of unneeded communication between the client and server. As an example, when forms are recursively expanded to add an additional field entry, the client requests the contents of the additional field from the server. Instead, the server could send a hidden prototype of all input types with the initial web page, and the Javascript front-end could copy that prototype and initialize its values rather than retrieving an initialized input field from the server.

Additionally, increasing the number of features supported by client-side Javascript offers another way to optimize pagination, instead of or in tandem with server-side query caching. Rather than the client loading a single page of results, the client could load 5 pages, centered on the current page, and hide 4 of those pages. When the user moves from page 10 to page 11, the local Javascript would hide page 10, show page 11, stop storing page 8, and request page 13 from the server. This would reduce the amount of time needed to move between adjacent pages and create a more responsive paging interface, since the rendered page could be changed without waiting for data from the server.

6.5 Exact Search Improvements

Currently, the exact search method only supports checking for equality between stored values and desired values. It does not support other comparisons, such as not equals, greater than, or less than. Reworking the exact search entry to use comparisons similar to the ranked search would allow for more flexibility in retrieving specific products. Additionally, another useful operator would be a substring or contains operator for descriptive traits. Adding this would allow more effective filtering based on fields such as product name, by specifying more than one acceptable value. For example, instead of retrieving all products named "IPhone 8" being sold for less than 150 dollars, a customer could retrieve all products whose names contain "IPhone" and are being sold for less than 150 dollars.

Additionally, the feature to progressively refine suggestions as the user fills out the form is limited. It only updates suggestions for detail values, and not detail names, as all detail names use the same list of suggestions. This could be improved by changing the detail names to each use an individual list, and initially populate those lists with values from the original shared list.

Suggestions also only update when the user changes a detail's name, and not when the user changes the value of a detail. It also only updates the suggestions for the value input corresponding to that particular detail name. This means that it is possible to get suggestions out of sync with what they should be by filling in and changing inputs in specific orders. This could be prevented by adding a function call to the onchange for every single input in the form, and updating every single list of suggestions in that function. That is, however, a brute force approach.

6.6 Need-based Search Improvements

The needmap system could be made more versatile to support a wider range of need types. As an example, consider purchasing carpet for all owned homes. The current system could match the square footage of houses and apartments to the square footage of rolls of carpet sold by vendors. However, it requires individual products matching the right-hand side of needmaps to have sufficiently high values, and does not allow summing up multiple right-hand side products. This is sensible for many subscription-type services, such as internet plans, since you cannot simply pay for two low-tier plans and receive twice as much coverage. However, for products such as rolls of carpet or floor tiles, multiple purchases can be easily combined. Two purchases of three hundred square feet each will cover just as much floor as one purchase of six hundred square feet. To support combinable products like this, needmaps could be given an additional field, which specifies the mode of interaction between left-hand and right-hand side fields. Modes could include sum-to-one, like internet service, sum-to-sum, like carpet, or one-to-one, where individual items from the left-hand side are matched with individual items from the right-hand side, like separate insurance policies for different vehicles.

Additionally, there are types of needs which currently require multiple needmaps to accurately emulate, and thus require the customer to perform multiple searches to properly fill. An example is internet service, when you consider wired and wireless internet as distinct from each other. A device which connects to the internet via Ethernet does not need Wifi, and does not increase bandwidth demands on routers. By specifying product type of left-hand side columns to only be mobile devices, such as tablets, smartphones, and laptops, a search can be made to find routers using only devices that need mobile connectivity. However, if this same search is used to find a subscription, it will fail to include enough bandwidth for any wired devices like desktop computers. These can be separated into two different needmaps, with one being for internet as a whole, and one being for only wireless internet. This is not a user-friendly solution, however, as it requires the user to be aware of the distinction and actively make both types of searches. Instead, another layer of abstraction could be used, to allow one button displayed on the need-based search page to run searches corresponding to multiple needmaps.

6.7 Vendor-driven Offers

Right now, offers can only come from customers, and then wait for vendors to accept or reject them. However, the offer system itself is ambivalent, and could support offers made by either party, if there were a front-end system for vendors to make offers. In order to implement this, an additional table would need to be added to hold standing product requests from customers.

Standing product requests would be issued by customers from product listing pages when they make a search and do not find any of the resulting products to be acceptable. This new table would contain a customer id, and a human-legible request description parsed from the search query which the customer is listing as a standing product request. Vendors would be able to browse requests in this table, and submit an offer matching one of their products to a specific request. Additionally, a nullable string column would need to be added to Offers, so customers could see which of their standing requests a vendor-issued offer would be aimed at satisfying.

6.8 Product List Display

Currently, product lists show only the category, name, and price of the product, and all details must be viewed by clicking the button to open the modal for an individual product. This is slow and inefficient when seeking to compare details of multiple products. To provide a better user experience, and facilitate faster shopping, the product list should be able to take any arbitrary product detail, and show it as a column in the list. This change would require modifying the list interface to allow the user to select additional columns, and reloading the list to add those columns. It would also require displaying something for products which contain no value for a desired column.

Additionally, allowing users to sort exact-match search results, or their own product listings, by the value of a certain detail would serve multiple use-cases. For example, viewing all books in a series listed in their proper order, or sorting rental housing by the number of bedrooms. Sorting by price, from low to high, is used as a general stand-in with search results, but is not always the most useful column to sort by.

Pagination could also be reworked to allow the user to specify a different number of products per page, rather than always using the same value.

6.9 Listing Interactions

Users' interactions with their own listed products could be expanded on and improved. Vendors selling large numbers of items currently have no effective way to find a specific item. Providing a tool to enable users to search through their own listings would facilitate that.

Additionally, there is currently no way to edit a product once it has been listed. Implementing one would allow users to have a greater level of control over their products. However, a properly implemented edit option for this system would not just change the values of the details corresponding to the existing product. Doing so would change the values of the product everywhere the product exists, which can include not only the listings of the product from the person editing it, but also other people's listings and ongoing contract offers. A robust edit functionality would need to make a deep copy of the product in such cases, allowing the user to edit the values for their own listing without affecting anything else.

Another valuable option would be a 'Hidden' field in the Sells table, allowing vendors to hide their products from showing up in customer's searches without permanently deleting the product. Additionally, a button to automatically hide a product would be added to the finalize-contract screen for vendors, similar to the customer's button to automatically add the product to their list of owned products. This product hiding functionality would facilitate the management of products with limited availability, such as housing.

6.10 Account Management

The system would also benefit from more robust tools for account management. Common features which are not implemented in this system are an email field in the users table and password resets using that email. Additionally, a table for customer shipping addresses would also be useful to enhance contract fulfillment. With that, when a customer makes an offer, they would be able to choose a shipping address associated with them in that table. Vendors would be able to see the ZIP code of a shipping address before accepting offers, to allow them to reject offers which their shipping solution cannot fulfill. The full shipping address would only become visible after the offer is accepted to allow the vendor to deliver the product.

6.11 Data Generation

Right now, the system has 3 different data generation functions which can produce a total of 8 different product types. Adding additional data generation functions corresponding to different types of products would allow for testing the applicability of the system model to a greater variety of product types.

Data generation support could also be added for fake users, to allow testing the system with many different listings in owns, sells, and offers.

6.12 Admin Interface

Currently, the system has two different unlinked pages, one used to add needmaps and the other used for data generation controls. Both pages are only accessible to a user with the username admin. For ease of access of these pages, and any potential additional admin-only interactions, another menu could be made to link to these pages if the current user is an admin.

7 PRIMARY DIFFICULTIES IN DEVELOPMENT

7.1 Timeline

The primary logistical difficulty involved in developing this project was establishing a timeline, and distributing work appropriately across the semester. At the beginning of the semester, we lacked the knowledge to make meaningful plans about the project's development, as we had very little idea of how to implement any of the project's requirements. By the end of the semester, these requirements were possible to implement, but effective work on the project could have begun far earlier. If there had been more clear expectations and guidance about how much of the project

should be done by different points in the semester, there would have been more opportunities for getting feedback on the project's development. It also would've been easier to schedule working on the project to avoid leaving a large amount of work for the end of the semester.

7.2 Ranked Search Optimization

A major technical difficulty was encountered with the performance of the ranked priority search method. The initial approach was functional for small datasets, but after implementation and usage of data generation to populate the database, became very slow, taking a minute to rank a category of a thousand products.

This was because of the logical technique and organization of queries within loops. To sort the products by how well they matched the search, each product within the category was given a numeric score, starting at 0 and increasing by the appropriate weight of each criterion the product matched. The score was initialized to 0 using a loop iterating over every product in the category. Within this same loop, an inner loop iterated over every criterion in the search specification. For every criterion, it queried either the attributes or features table, depending on the criterion's specifics. This query was used to identify whether the current product matched the current criterion, and increase its search matching score if it did. Once there were a thousand products in the database, these queries were looking through detail tables of roughly three thousand entries, and a thousand queries were made to resolve a single criterion. This was very slow.

To resolve this, the score initializing loop was separated, becoming a single-purpose loop. The criteria resolving loop was restructured. Rather than checking if one specific product matched each criterion, it now returns a list of ids for all products matching that criterion. It uses that to select products from the current category, and increases their search matching score. This technique is still far slower than the exact search, but it has scaled to effectively resolve a search across a category of three thousand products and detail tables with forty thousand entries, taking only several seconds to give results.

7.3 Product Details

The firm distinction between attributes and features, between numeric and descriptive details about a product, created issues throughout the project's duration, up to and including making this report's terminology clear. Separating them into two tables made earlier development stages easier, as it simplified basic logic such as identifying the type for inputs in the detail specification form for listing new products. However, as the project continued, there were more scenarios where the two types of details should be treated the same.

One example is making the exact search form use a generic Add Field button and showing all types of product details as suggestions, rather than only numeric or only descriptive details. To implement this, three lists were used. Lists for numeric and descriptive detail name suggestions were already being created. A third list was made, containing all values from either list. This third list was used to provide suggestions for form input. When a detail name is entered, it is checked against the two previously existing lists to determine what type of detail it is and set the value input type appropriately. If the details had all been a single table, with a third column indicating the type of the detail, the combined suggestions list would have been easier to populate and checking the type of a given detail name would be very simple.

The distinction also frustrates efforts to add additional types of details. Done properly, any new type of detail should be a new table to match the existing pattern. During project implementation, booleans were recognized as valuable, in order to support product features which are either present or absent. Rather than creating a third table and reworking all existing interactions with product details to support this, the descriptive details table was overloaded. Any form input which can take a descriptive detail value can also be left empty. This is then converted, server-side, to a value of Yes, to indicate that the feature is present. However, this led to unexpected behavior in certain cases, such as accidentally leaving a descriptive value input empty. The interface allowed it. The server treated those values as Yes, and search results came back empty, as existing products all had actual values for those details and not just the boolean indicator Yes.

8 PROJECT SOURCE

The project is available on github at https://github.com/shreckneps/360project The database schema can be imported using the DATABASE.sql file in the root of the repository. To make use of the admin-only pages, used for registering needmaps and generating data to populate the database, create an account with the username admin. At least one vendor account should exist before running any datagen functions. An alternative database import, including both data and schema, can be imported using the DATABASE_FILLED.sql file. All users in this file have password equal to their username.

9 CONCLUSION

The purpose of this project was to implement an e-commerce platform, with specific support for three different ways of matching customers to products, and binding contract entry between customers and vendors.

This system implements all the core specified functionality, in largely versatile and broadly-applicable ways. There is still a lot of room for refinement of many aspects of the interface, and many desirable features which the system could be expanded to include.

REFERENCES

[1] OpenJS Foundation. 2022. jQuery. https://jquery.com/.
[2] Apache Friends. 2022. XAMPP Installers and Downloaders for Apache Friends. https://apachefriends.org/.
[3] Laravel LLC. 2022. Laravel - The PHP Framework for Web Artisans. https://laravel.com/.
[4] Bootstrap Team. 2022. Bootstrap - The Most Popular HTML, CSS, and JS library in the world. https://getbootstrap.com/.