

Superlative Sales CS360 Project



Seth Lunders

lund4272@vandals.uidaho.edu

Donald J. Hammer

hamm6866@vandals.uidaho.edu

ABSTRACT

The *Superlative Sales* platform is a commercial website designed to facilitate transactions between customers and vendors. It provides a marketplace for vendors to display their products and services while giving customers powerful tools to find them. The site works from a database of products and services provided by vendors from which customers may provide queries. It has an interface for vendors to add their listings.

1 INTRODUCTION

The design requirements for the project include a web based interface for user and vendor interaction and three options for product/service searches. The requirement to host the site on the University of Idaho servers was omitted.

Tools and Software

Our team used git, Github, Github Desktop, and Discord to compile files and communicate changes to the project. We used Django for server back-end and Bootstrap 5 for styling the HTML on the front-end. We used SQLite as our database, as it is easy to set up and get working with Django. We used Visual Studio Code for writing and managing our code. We used Overleaf and LaTeX as our text editors for reports.

Here we will outline some of the helpful features of each of the tools we used.

git:

Git was extremely useful for our product. It helped us keep track of changes, as well as share the code between us.

Github:

Github gave us a place to host our git repository. This ensured our code was always safely backed up, in case something happened to our devices.

Discord:

Discord was useful for collaborating from a distance. It has a nice screen sharing feature so we could show each other our code and ideas we had for the project.

Django:

Django was the most important tool for getting this project done. Once we got past the learning curve (though there is still a lot to learn), it was fairly straightforward to add the features we wanted. As we mention elsewhere in this paper, there are multiple features we would've like to implement but did not. This wasn't due to a lack of understanding of Django, just a lack of time to implement everything.

The main workflow went like this:

1. Design the database models
2. Create the base template for our site, with the
3. Pick a page to create
4. Create the path for the page in `urls.py`
5. Create the view function or class for the page in `views.py`
6. Create the template for the page, extending the base template.
7. Use the context data from the view function to populate the template with objects (ie, products)

Bootstrap 5: Bootstrap was very useful in quickly creating a site that looks good. Given more time, it would have been nice to write custom CSS, but for our purposes Bootstrap worked great, and gave us time to focus our work on the back-end side of our site.

SQLite

SQLite is the database we used for our project. We picked this because of the ease of setting it up and sharing it over git.

Visual Studio Code

All of the programming for the site was done in Visual Studio Code. We took advantage of the code formatting, linting, and styling features. Another super helpful feature was all the extensions available, they're not strictly necessary but they do help the coding process go smoother and faster.

django-extensions and graphviz

As the size of our models.py file increased, the time to update our ER Diagram increased as well. These tools allowed us to automatically generate an ER Diagram. This also allowed us to see Django's models that we didn't code ourselves, such as the User model.

Overleaf

Overleaf is an online LaTeX editor and viewer. This is what we used to format this report. One very helpful feature here is the ability to share the document so we could both work on the document at the same time.

SQLite

SQLite is the database we used for our project. We picked this because of the ease of setting it up and sharing it over git.

Firefox & Chrome

We tested our site in both Firefox and Chrome. The responsive design tools in both made it easy to ensure the site looks good at any screen size.

Linux, WSL, and Anaconda

We used both Linux and Windows machines to work on the project. On Linux, it's fairly straightforward to get the Django server running. We used miniconda to create a virtual environment to install all the python libraries needed. On Windows, we used the Windows Subsystem for Linux, as well as Anaconda, to accomplish the same thing.

ionicons

For icons, we used the open-source icon library called ionicons. The site is located at <https://ionic.io/ionicons>.

Lorem Picsum

We used this to get filler images for our site. It is an API that allows you to get random images, located at <https://picsum.photos>

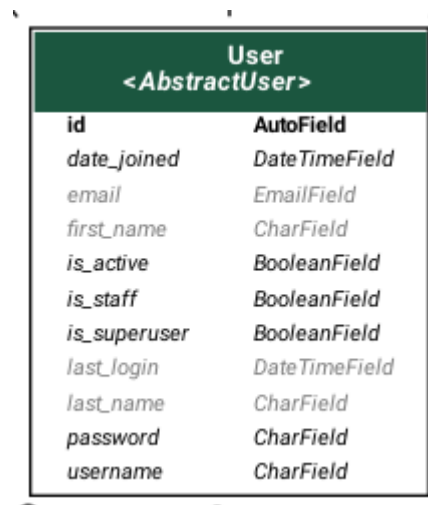
2 DESIGN AND ARCHITECTURE

Database Models

User:

The user model, used for keeping track of a Customer's or Vendor's data, is the default one provided with Django. It stores their unique user id, the date and time they joined, their email, name, username, and password hash.

The user model is linked to shipping and billing addresses with a 1-to-many relationship. While not implemented on our site, this would allow one user to store multiple addresses.



User <AbstractUser>	
id	AutoField
date_joined	DateTimeField
email	EmailField
first_name	CharField
is_active	BooleanField
is_staff	BooleanField
is_superuser	BooleanField
last_login	DateTimeField
last_name	CharField
password	CharField
username	CharField

Vendor:

The Vendor model is what people use to list products and services on our site. It has a 0 or 1 relationship with the User model, so a user can have at most 1 vendor associated with their account. A customer can then search for their products and services, or they can look at specifically that vendor's information on the Vendor Detail page.

Vendor	
id	BigAutoField
owner	OneToOneField (id)
address1	CharField
address2	CharField
city	CharField
country	CharField
date_created	DateTimeField
description	TextField
emergency_phone_number	CharField
logoURL	URLField
name	CharField
phone_number	CharField
state_or_province	CharField
zip_code	CharField

ProductListing	
id	UUIDField
vendor	ForeignKey (id)
active	BooleanField
description	TextField
imageURL	URLField
name	CharField
price	DecimalField
quantity_available	IntegerField
restockDate	DateField

Invoice:

The Invoice model stores its own unique ID, the foreign key to the Purchaser, date placed, and the total payment amount. It serves as an intermediary between the Users and the Products and Services purchased. By splitting the orders up this way, one order may have many InvoiceProducts and InvoiceServices.

InvoiceProduct:

The InvoiceProduct is a relationship between a unique invoice and a product. It stores the product purchased, quantity purchased, who sold the product, the quantity ordered, the status of that product order, and the price per unit paid at the time of purchase.

InvoiceService:

This is similar to the InvoiceProduct, but stores a link to the service, and price paid instead of product and quantity ordered.

ProductTag/Service Tag:

These have a many-to-many relationship with ProductListings or ServiceListings. They make it easy for customers to look at categories of items. They only store two things, the tag name and the view count. The view count is important for visibility on the site, as the sidebar has the most popular Product and Service tags listed on every page.

Product/ServiceListing:

This, along with the ServiceListing, make up the most important components of the site. They store the product or service's image url, name, vendor, description, tags, price, and availability. To be able to be found on the by customers, vendors should include relevant keywords in the description area, as well as relevant tags.

Cart:

The cart model serves as a link between the user and a list of items in their cart. It has a unique id, a one-to-one relationship to a user, and a subtotal. The subtotal is calculated whenever a CartProduct or CartService is saved with a link to the cart.

CartProduct/Service:

This is very similar to the InvoiceProduct. It is a link between a ProductListing and the user's cart. It stores the cart it is connected to, the product it represents, and the quantity the user has in their cart. The CartService is almost the same; it has an id, cart, and service.

Other Models:

There are other models in the ERD, but they all are created by Django, handling things like sessions, groups, and logging.

3 UX/UI

Nav Bar:

The nav bar has a simple, easy to use design. We used icons instead of text to give it a more polished feel. On the left side, the first thing in the nav bar is our site name, which links back to the index page. Then we have Vendors, Products, Services, and the Cart pages.

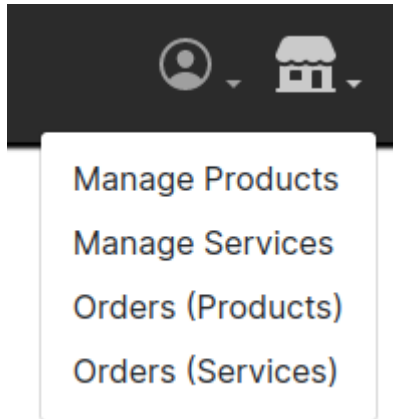


User Menu:

On the right side of the screen, if the user is logged in we can see the User icon. This is a dropdown menu with links to 'View Orders' and to log out.

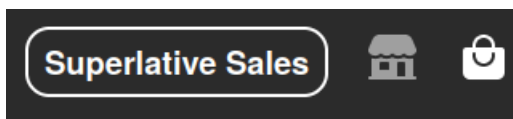
Vendor Menu:

Next to the User Menu, if the user has an active Vendor associated with their account, a Vendor dropdown menu is displayed. This menu has links for vendors to manage their products, services, and current and past orders.



Sidebar:

When the user is not signed in, they will have the following options available: Popular Product Tags, Service Tags, and Cart. These tags are listed by popularity; that is, they are sorted from most to least unique views. The top 7 most popular tags are shown on the sidebar of the site. If the customer has products in their cart, a cart card will appear, with product names, quantities, and price. It also has a checkout button with the subtotal listed above.



Welcome, Seth

Cart
Camera 1 \$700
Golf Balls 24 \$24
\$736.00
Check out

Popular Product Tags

[Photography](#)
[Kitchen](#)
[Outdoors](#)
[Sports](#)
[Golf](#)
[Electronics](#)
[Books](#)

Popular Service Tags:

[Painting](#)
[Sports](#)
[Technology](#)
[Food](#)
[Computer Repair](#)
[Transportation](#)
[Outdoors](#)

Login Page:

It prompts the user to enter their username and password. This accommodates both customers and vendors.

Password Reset Pages:

If the user forgets their password, they can click the Forgot Password link. This will prompt them to enter their email, where a password reset link will be sent. Our demo product is not able to send emails, so the links are instead printed in the console.

Customer Pages:

Check Out:

Currently, the 'Check Out' page is a placeholder, just showing the cart. This is where users would be prompted to enter in payment and shipping information.

Cart Detail:

The Cart Detail page allows customers to view all the products and services in their cart. They can update quantities, remove services, and check out from here. Currently, clicking the 'Check Out' button creates an Invoice of all the Products and Services in the user's cart, then empties that cart. Customers can then view their invoice and order status from the 'My Orders' page.

Orders List:

The 'My Orders' page lists all a customer's orders. It sorts them by date and time placed, and shows the total amount paid. Clicking the 'Details' button sends the user to the 'Order Details' page of that order.

Orders Detail:

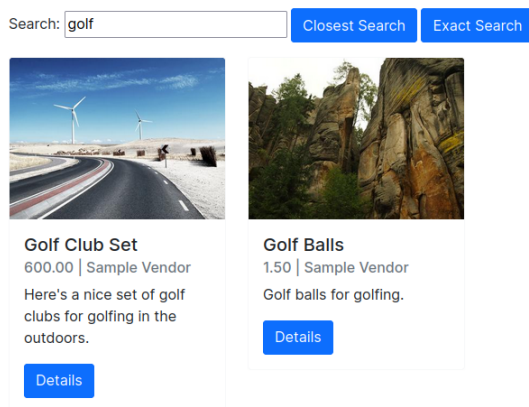
The Order Detail page lists all the Products and Services that were in that specific order. It shows the price that was paid, quantity ordered, and the status of each item. It also shows Date and Time the order was placed, as well as its unique order number.

Product Search:

The product search page is probably the most important page of our site. There are two types of searches available to the user. The first, and default, is the Closest Search. This takes in a string, tokenizes it into key words, then takes a union of a query of each of those words. The second type is the exact search. Only products that contain all the keywords searched for will show up in these search results.

The products are displayed to the user in a list of cards, as shown below. (Note: The products link to a 'https://picsum.photos/' url, which returns a random photo on each query.)

Products



Index Page:

The index page is the landing of our site. It has our logo, our names, and a short description of the site and why it was created. We included some stats that query the database to check how many Vendors there are, how many Products and Services are active on the site, and how many total sales have been placed. We also provide helpful links to

their respective pages.

Vendor Pages:

Manage Product/Service Orders:

This page is where Vendors can view orders that users have placed. They can mark the order as 'received', 'canceled', or 'shipped'. The Customer is able to see this status in their 'My Orders' page.

Manage Products/Services

Vendors can go here to create new listings, activate/deactive listings, or edit listings they already have.

Other Pages

- Product Listing Detail
- Product Listings (List)
- Service Listing Detail
- Service Listings (List)
- Vendor Detail
- Vendors (List)

Responsive Design

When designing our site, we wanted to make sure it works on any device. Bootstrap makes this easy to set up. Here are a few areas that we used responsive web design.

Nav Bar:

The nav bar is the most obvious responsive element. When the screen goes below a certain width, the buttons disappear into a hamburger menu.

Popular Tags:

The nav bar is the most obvious responsive element. When the screen goes below a certain width, the buttons disappear into a hamburger menu.

Desktop:

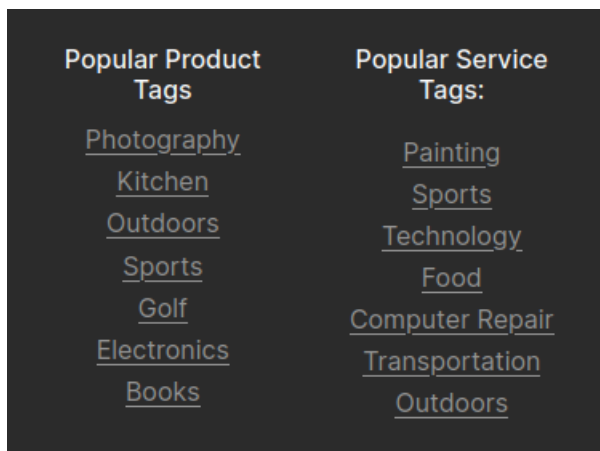
Popular Product Tags

[Photography](#)
[Kitchen](#)
[Outdoors](#)
[Sports](#)
[Golf](#)
[Electronics](#)
[Books](#)

Popular Service Tags:

[Painting](#)
[Sports](#)
[Technology](#)
[Food](#)
[Computer Repair](#)
[Transportation](#)
[Outdoors](#)

Mobile:



Nav Bar:

The nav bar is the most obvious responsive element. When the screen goes below a certain width, the buttons disappear into a hamburger menu.

4 SQL Queries

In this section we outline some of the SQL queries used on our site, as well as their Django equivalents.

Listing All Objects

Often, we just want to get all the objects in a list.

Here is the basic Django function for that:

```
ProductListing.objects.all()
```

The raw SQLite query for this looks like:

```
SELECT "storeapp_productlisting"."id",
       "storeapp_productlisting"."active",
       "storeapp_productlisting"."name",
       "storeapp_productlisting"."imageurl",
       "storeapp_productlisting"."price",
       "storeapp_productlisting"."description",
       "storeapp_productlisting"."quantity_available",
       "storeapp_productlisting"."restockdate",
       "storeapp_productlisting"."vendor_id"
FROM   "storeapp_productlisting"
```

Here's what our Exact Search looks like, in Django and SQLite:

```
ProductListing.objects.filter(Q(name__contains = q)||Q(description__contains = q))
```

```
SELECT "storeapp_productlisting"."id",
       "storeapp_productlisting"."active",
       "storeapp_productlisting"."name",
       "storeapp_productlisting"."imageurl",
       "storeapp_productlisting"."price",
       "storeapp_productlisting"."description",
       "storeapp_productlisting"."quantity_available",
       "storeapp_productlisting"."restockdate",
       "storeapp_productlisting"."vendor_id"
FROM   "storeapp_productlisting"
WHERE (
       "storeapp_productlisting"."name"
LIKE
       %test% ESCAPE \'
OR
       "storeapp_productlisting"."description"
LIKE
       %test% ESCAPE \'
)
```

5 Conclusion and Future Developments

Given the limitations of time and experience, we were not able to include every function that we might want. Currently all accounts have to be

made with the help of an administrator. This includes user accounts and vendor accounts, as well as features like tags.

Another area for improvement is that services are essentially treated the same as products; that is, there is not a way to bill per unit of time. Sellers can only offer a flat rate. The model can allow for recurring (e.g. per hour, per day) services, however.

Currently, we do not have a checkout view. Checking out your cart creates invoice for your order, turning all the CartProducts and CartServices into InvoiceProducts and InvoiceServices. This means there is no page for payment, getting shipping information, etc.

The current filtering process is limited to keyword based queries. This puts more onus on the vendors to properly describe their products or services. Furthermore, this could also be exploited by putting irrelevant keywords onto an unrelated product to get views.

We would also like to have options to sort by things like price, shipping time, availability, etc. For Products and Services, sorting by popularity

(like the Tag) would be nice.

One downside we found with SQLite is the search functionality. Currently, our exact and closest searches make one database query per keyword. With a different database system, this could be reduced to one query per search, improving performance.

As was mentioned earlier, we used Bootstrap for the styling of our site. This helped take some of the work off styling, so we could work on the more important back-end side of things. However, this also means we weren't able to really make the site unique. Given more time, it would've been nice to develop an entire custom style for our site.

Conclusion

There's still a lot of room for improvement on our site, but that's how it goes when you're learning something new. We're happy with how the final product turned out; this has been one of our favorite projects so far. It's been very interesting to see what all goes into making a dynamic site, and we're looking forward to using our new skills on projects in the future.

References

- [1] 2022. Django <https://www.djangoproject.com/> : May 9, 2022
- [2] 2021. Generating ERD for Django Applications <https://wadewilliams.com/software/generating-erd-for-django-applications/> : May 9, 2022
- [3] 2022. ionicons <https://ionic.io/ionicons> : May 9, 2022
- [4] 2022. MDN Local Library (Django) https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Tutorial_local_library_website : May 9, 2022