Shopify Project

Will Cook, Lenin Lewis, James Marsh, Alyssa Newhart, Riley Slater, Daniel Vinogradov

Gianforte School of Computing, Montana State University

ESOF 423: Software Engineering Applications

Mr. Daniel Defrance

Spring 2022

1. Program

Source Code: https://github.com/423s22/G6

Over this past semester, our group worked with a client to develop an app for a client that owns a small business and runs their online store using the Shopify eCommerce platform. Shopify is an eCommerce platform for online stores and creates a point of sales system for many businesses. Shopify allows for the installation of third applications, so the client requested that we create an application to allow them to have custom pricing for product customization (such as size, color, material, and engravings) with an automatically generated price.

2. Teamwork

Team Member 1: Throughout this project, I have focused on the CI(Continuous Integration) testing in Github and the overall database integration. I build the CRUD(Create, Read, Update, and Delete) functions for the database using MySQL and Node.js. This was integrated into the group's Heroku app for the application using ClearDB. For me, this project has been a steep learning curve as I have never used Javascript, React, or the Rest API. Although I have struggled with some tasks, my experience with databases has helped to combat errors I developed while working. We chose these languages for a couple of reasons, some were required to work with the Shopify platform and MySQL was chosen for the memory, storage usage, and speed used when accessing the database. During the database creation, I was required to reformat the style used for storing or accessing the database. This was because I misunderstood what was needed and how databases were executed in Javascript. The level of logic required for the database was reduced to fit a usable but attainable scope within the amount of time left available for the

project. My group members have been patient with the bugs in the database and assisted in the final product for a working product. I also assisted in the later versions of the UML class and sequence diagram which include the database connections. Contributions can be viewed on Github, but as of May 6th, I have tracked 113 hours and completed 75 work-hour units. My assigned tasks can be found in the group Jira backlog and bi-weekly sprints.

Team Member 2: Throughout the project, this team member worked on numerous tasks. This team member's main role over the semester was to be the scrum master, they were deadline-oriented to ensure that the group stayed on the same page. The role of scrum master became more valuable as the team grew from three people to six people. This team member also spent time developing the frontend web development and UX/UI designs along with completing user testing. The main tasks completed over the semester are listed below.

- Homepage
    - Created the initial design and page links
- Creating the Frequently Asked Questions (FAQ) page
    - Create the questions JSON file
    - Used mapping to have a functional search bar
    - Designing the UI elements
- UML documentation (link)
    - Class Diagram (link)
    - Object Diagram (link)
    - Sequence Diagram (link)
    - Decorator Pattern ended up not being implemented in code (link)

- User Testing
    - Interviewed two people and created full profiles on each
    - Created tests for the users to complete
    - Complied the list of bugs and delegated tasks

Team member 2 has 125 hours of work logged and 83 hours of work units completed.

Team member 3: Team member 3 created the base front end/back end of our application using React, Next.js, and Node.js. They then embedded the app into our Shopify development store and created the Heroku app for our application. Team member 3 created the wireframe for the UI design of the application and developed the front end of the Add Options page, which is where the user can view, add, and remove custom engraving and dropdown menu options for a selected Shopify product. The components that make up the Add Options page are EngravingForm, DropdownForm, SelectOptions, ProductCard, AddOptions, ShowOptions, and their CSS stylings. These components were created using React and Shopify Polaris components. We chose to use Shopify Polaris components within our application to provide a consistent and familiar UI experience for the user. Team member 3 also worked with team member 1 to connect the EngravingForm, DropdownForm, and ShowOptions components to the database. Team member 3 wrote the initial developer documentation for the setup of our application within Shopify and continued to update the user documentation upon the completion of features within the application. They also added the CreateProduct, DeleteProduct, and BuildOptions helpers for our application. Team member 3 worked 124 hours total and completed 83 work unit hours.

Team Member 4: Team Member 4(TM4) worked on app front-end tests using Jest snapshots, theme app extension front-end tests using Cypress, and the theme app extension code with Team

Member 5(TM5). There were some configuration issues with babel and our nextJS but once I got them resolved it was smooth sailing. Babel is used by Jest to transpile anything that is not common JS, making it almost necessary to use with ES6 modules. Snapshot testing is not as rigorous as integration testing or end-to-end testing because all it does is mount the React component. Then, snapshot tests take a snapshot of the HTML that would be rendered from the mounted component and compare it against a previous snapshot to look for inconsistencies. Snapshot testing works well for the scope of this project and provides us with the feedback we need to diagnose errors during development. When trying to use Cypress to test our app front-end there were some issues with Shopify flagging the tests in its bot detection system. This is why I decided on using snapshot tests for the app front-end. Cypress was implemented to test our theme app extension front-end. The theme extension code that TM5 and TM4 have been working on has been pretty tricky to navigate. Finally, we were able to get it working after Team Member 2 and Team Member 3 refactored our database because of unexpected information needed to send items to the cart with Shopify's cart API. In total, I have 106 hours of work logged.

Team Member 5: Team Member 5 (TM5) completed the theme app extension alongside TM4 and wrote the public server routing / API. While developing the theme app extension, the Shopify environment itself was a large hurdle and forced the team to restrict the design space of the application in order to accommodate Shopify's constraints. The way Shopify stores its product information & pricing and the way applications are deployed to customers both made initial plans to meet the client's design requests unfeasible. The server and database were designed with the idea in mind that the team would be able to develop some private front-end functionality or validation for price checking, but TM5 quickly found that any code pushed to the

store page is publicly visible, and thus unable to send private requests to the server or database for pricing changes. So, TM5 created a separate set of public-facing routes for access by customers to request pricing and product info from within the Shopify store page. In total, TM5 has 103 hours of work logged.

Team Member 6: Our original group G7 merged with G6, which meant we needed to delegate each other new tasks. The work Daniel, and I, was delegated was strictly front end. I developed a mock store, filled it with images, and descriptions, created products, and collections, and incorporated Team Member 2's menus to design the store. The theme of our store is an outdoor apparel and equipment store known as Eagle Apparel. I also was assigned to create a brand, logo, and story for our theme. The store uses the theme Venture, the free version, available from the Shopify Theme Store. This theme is great because the theme has constant updates, meaning if we want to change our store to a different theme, we are not restricted to sticking to older versions of themes. Over the course of the project, TM6 logged 56 hours.

3. Design pattern

The Singleton pattern ensures a class has only one instance and provides a global point of access to it. We implemented the Singleton Pattern in our server architecture, specifically our Database Handler, to maintain the invariant that we have only one database connection at a time. Having limited database connections was necessary because of the limitation of our free account and we did not need multiple connections for the scope of this project. Multiple connections would be most useful for multithreading, which we cannot do because we only have 1 set of data to be written. To do this, we implemented the connect() and _checkConnect() functions, which

can be found on **lines 4-26** of our handlerDB.js module. We can call the connect() method to obtain a connection, which will either return the existing active connection or establish a new one (if none exists). This ensures that we are not creating more than one database connection at any given time. In the standard Singleton Pattern, one might return the singular object, however, we chose to adapt this pattern and return a True/False indicator instead, using the variable "con" as our static global point of access instead. This is because we wanted to call upon our database from within the handlerDB module, but we wanted to check for connections from our server. Creating a generic database connection manager was beyond the scope of this project, and we decided the additional division of functionality would have a negative impact on the readability of our code for new developers coming in. Since our handler and connection were so specific to our database connection and server design, we decided upon this simplification as the most effective compromise to meet our design goals.

4. Technical writing

Developer Documentation: https://github.com/423s22/G6#readme

User Documentation:

Project's goal

The goal is to create an embedded app for Shopify. It can be used in conjunction with other apps to create the user experience that the shop holder wishes for. The main functionality of our app is that the user can see the overall sum of the product

when additional changes to the product are made. i.e., a necklace that has a leaf pendant can have the gemstones and metal type changed and with each change, the overall price will be updated before the user moves it to the cart.

Installation

Please contact us for an installation link

- After clicking on the link, navigate to your Shopify admin.
- In your Shopify admin, authorize the use of the app by clicking install app

Using Rival

1. Once you have successfully installed the *Rival* app, you can find it within your *Apps* page in your Shopify Admin.
2. After you have clicked on *Apps* you will be directed towards your *Apps* page for your store, which will display all the Apps you have currently installed on your store.
3. You can open *Rival* by clicking on the app's name.

Homepage

On the *Rival* homepage, you will find buttons that navigate to the various pages of the app such as the *Add Options* and *FAQ* pages. You can also click on a page's name within the top navigation bar to navigate directly to it. The current page is indicated on the navigation tabs by a green line underneath it.

Add Options

1. First, navigate to the *Add Options* page using the navigation tab.
2. Once you have successfully directed to the *Add Options* page, click on the *Select options* button to open the product selector.
3. In the product selector, select the product you would like to add options to.
4. Once you are ready to proceed, click *Add*.
5. After clicking *Add* from within the product selector, you will be directed to *Option Selection.*
6. On this page, you are able to see the product you have selected.

If you would like to cancel and return to product selection, you click the *X* on the product card.

Add Engraving Option

1. Open the *Option Types* dropdown menu to select an option.
2. Select *Engraving* from the dropdown menu and click *Add Option*
3. On the *Engraving form*, select your desired number of lines, enter a description for your option, and enter any additional price associated with the engraving option (optional)
    a. Option type selection. Select Engraving from the dropdown menu
4. 4. When you are finished, click *Submit* to add your engraving option to your product.
    a. On successful submission, you will be redirected back to the *Add Options* page
    b. If you would like to exit without submitting click the *X* at the top of the card to return back to to the *Add Options* page

Add Dropdown Option

1. 1. Open the *Option Types* dropdown menu to select an option.
2. 2. Select *Dropdown menu* from the dropdown menu and click *Add Option*
3. 3. On the *Dropdown form*, enter a title for your dropdown menu (required), and then enter the options you would like to appear in your menu
4. Once you are finished entering your options and menu title, click on *Apply Options* to continue on to price selection
5. In price selection, you can select an option from the menu and add a price to it using the price box
6. Click *Add* to apply the price to the option
7. When you are finished, click `Submit` to apply your dropdown menu to your product.

On successful submit, you will be redirected back to the *Add Options* page

If you would like to exit without submitting click the *X* at the top of the card to return back to to the *Add Options* page

Viewing Applied Options

1. Navigate to the *Add Options* page
2. Click the *Applied Options* button to open up the applied options card

Note: this menu will be empty if you haven't applied any options

Note: if you would like to refresh the applied options card, click the `Refresh` button

Note: you can hide/open the applied options card by pressing the `Applied Options` button

Removing Applied Options

1. Open up the applied options card
2. Find the option you would like to remove
3. Click the *trashcan icon* on the option card to remove it

Note: the option should be removed instantly. If not, you may need to click the `Refresh` button.

Questions? Navigate to our FAQ page within the app for additional resources.

Found a bug?

Reporting a bug: fill out a form that follows these steps

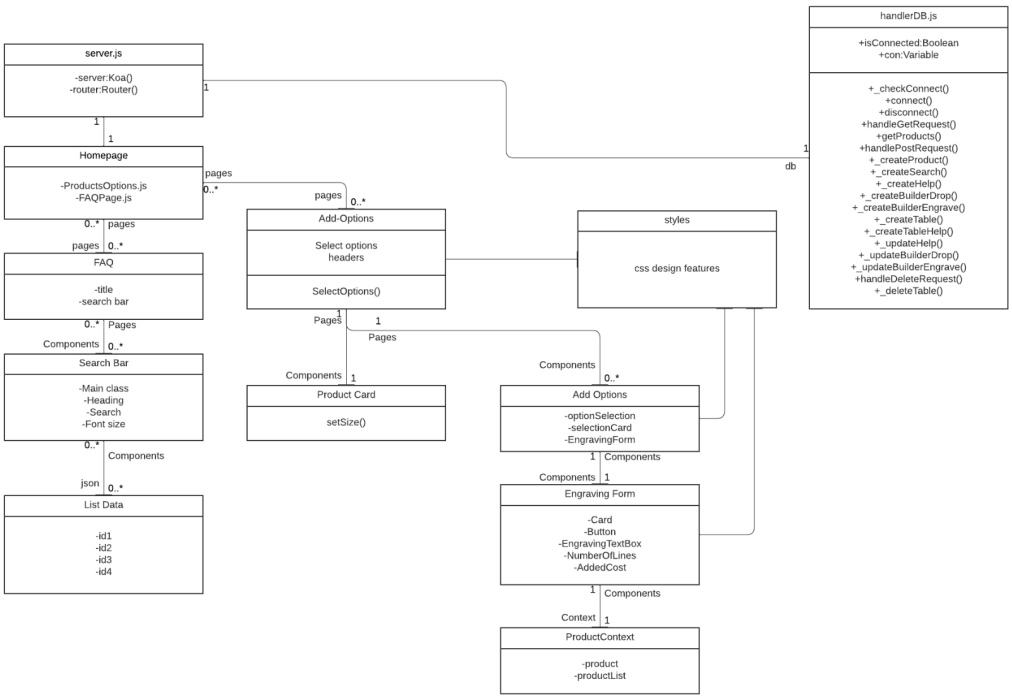
- When a problem is found a unique number will be assigned to the incident
- Give the bug a descriptive title after the number is assigned
- Set the priority to the bug issues so it can be fixed in an appropriate amount of time
  Inform us what operating system you are working on
- Give us a description that focuses on these points
  - Try to recall the exact steps that were taken to find the bug in the system
  - Specify the exact issues that occurred on your machine
  - Try to create a verbal visualization for the issue at hand
  - Expected and actual result
  - A screenshot of your error

Submit the form to us!

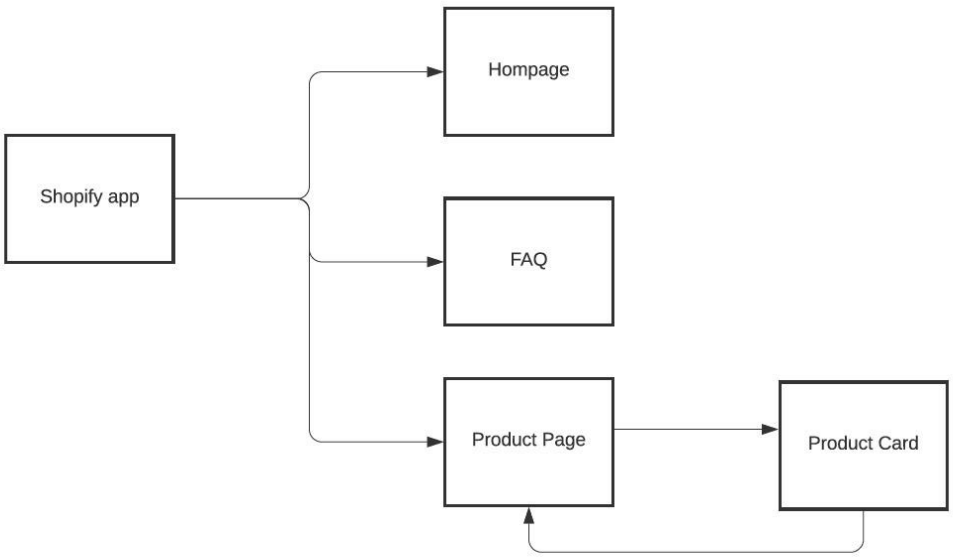
For more information on how to interact with our app regarding Shopify interaction follow this link

5. UML

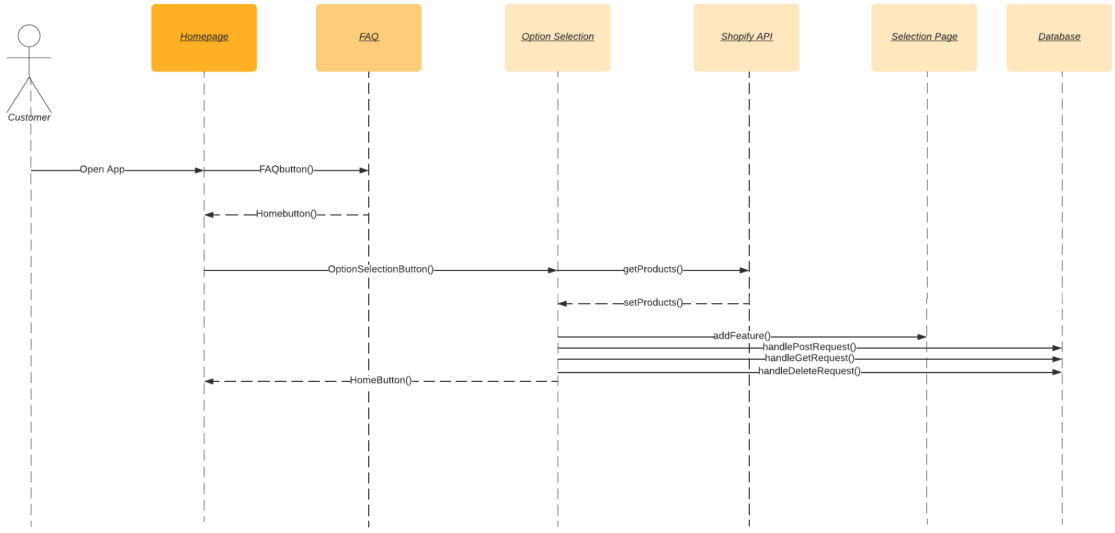
Class Diagram



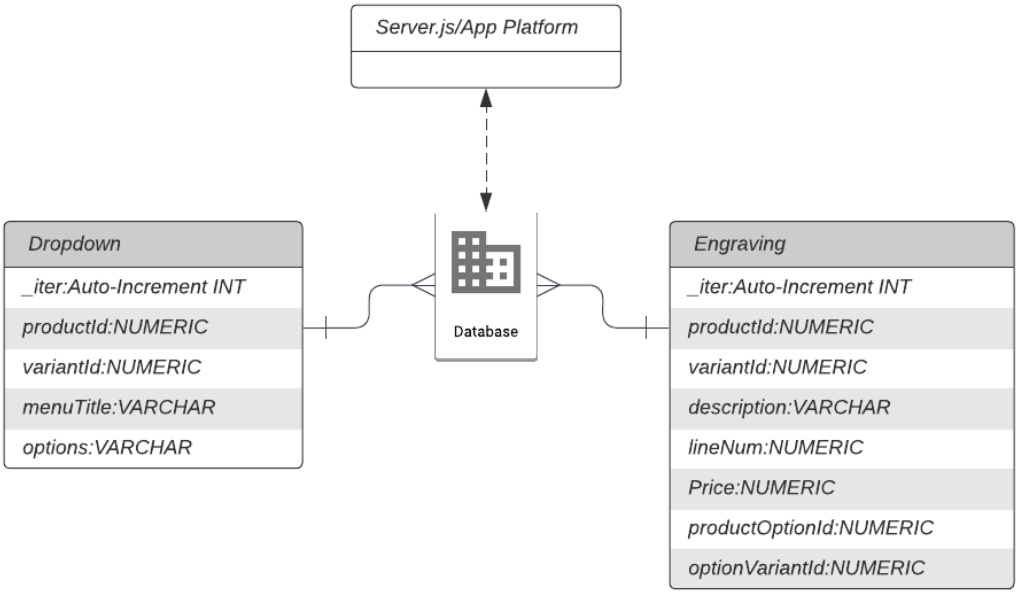
Object Diagram



Sequence Diagram



Database Architecture Diagram:



6. Design trade-offs

Trade-off 1: When creating the search bar element of the Frequenly Asked Questions (FAQ) the original design choice was to use the Polaris documentation through Shopify to create the search bar and use the search bar to compare to the data in a JSON file. However as the project progressed it became effident that search bar feature in the Polaris documaention did not search the JSON file in a useful manner for the scope of the feature. Using this challenge as a pivot in creating the FAQ feature, creating a map comparion was used to compare the user input in the search bar to the information in the JSON file. This allowed for the user to type keywords into the search bar and find the correlating documentation.

Trade-off 2: While writing the code for the theme app extension we decided to use the programming language built by Shopify, Liquid, only when necessary. Shopify requires this code to be inside of a ".liquid" file. Effectively, the contents of the ".liquid" file are injected into the storefront of the merchants' store which connects our database to the merchants' storefront. Once we understood this piece of technology we were able to make it our own and deviate from the native Liquid syntax. Liquid was made by Shopify to really only interact with the contents that are readily on the page(in the DOM) but we needed to query our database and then dynamically generate the correct UI for the storefront. Liquid could not provide this type of functionality. We used JavaScript to accomplish this and built a string of HTML to be injected into the merchants' storefront based on the results of the database query.

Trade-off 3: When implementing Shopify product creation/deletion for our custom option price storage, we decided to use the Shopify REST Admin API over the Shopify GraphQL Admin API. Originally, we had planned to use the Shopify GraphQL Admin API since

it is Shopify's recommended Admin API. However, integrating GraphQL into our application proved to be a challenge. Some issues arose with our application's current React package version and Apollo Client, which is Shopify's recommended GraphQL client. Updating our React package version had the potential to cause issues elsewhere in the application due to Shopify's version dependencies. Additionally, it was determined that there would not be a significant difference performance-wise between the two APIs for product creation/deletion. As a result, we decided to pivot and use the Shopify REST Admin API for product creation/deletion, which was straightforward to implement given our current application setup.

7. Software Development Life Cycle Model

Over the semester we used Scrum development in order to stay on track for the project. Scrum development is broken down into two major elements: the product backlog, and the Sprint. The product backlog is a list of all possible features, bugs, and tasks required to reach the team's goal with the project. The Sprint is a one-to-four-week period of development aimed at completing a specific subset of tasks chosen by the team. At the start of the print, the team sits down in a planning meeting to set the goal for the Sprint, and choose tasks from the Product Backlog that will help achieve that goal. All of the tasks chosen are added to the Sprint Backlog, and then team members volunteer for specific ones which they will start working on for the Sprint. Then they work on them, and if they have extra time they choose extra tasks from the Product Backlog to complete as well. Each day the team gets together for a short meeting to keep each other up to date on what they are working on, how it's going, and what obstacles they are facing. At the end of the week, the team sits down in a Sprint Retrospective to analyze how the Sprint went, identify additional tasks to add to the Product Backlog and identify ways to improve

for the next Sprint. Any uncompleted tasks are added back to the Product Backlog, ready to be continued in a future sprint.

Using Scrum was very helpful to us, because of the high focus on identifying the most important features of the product to be working on at any given time. We did not complete ALL possible features for the application - there is plenty of additional development that could go on - but Sprint helped us make a robust, working application that successfully contains a smaller set of features. We started with a “zero feature” application and iterated upon that to ensure that no matter how many features we had, our software always worked. Our Sprint goal was usually focused on completing a new feature in our app, or extending it with some new functionality. Once our main release was complete, our group focused on honing the features that were in place and making them better. The Team found the Sprint Retrospective to be a useful tool for rapidly improving our processes each Sprint, by fostering healthy discussion about obstacles encountered during development. Overall, Scrum development worked well for our group and allowed for us to make a working product in a timely manner.