Senior Team Capstone Portfolio
Shopify Application
ESOF 423: Spring 2022
Montana State University

Team Members:

John Jubenville

Ryan Krauss

Nathan Sheffels

Section 1: Program.
See attached source.zip file for entire application.

Below is a sample from the theme-app-extension directory:

theme-app-extension/assets/app-block.js:

```
function removeItemFromCart(id) {
  const val = id.toString();
    var newVariantIdArray = new Array(val);
    var product_data = newVariantIdArray.map(variantId => {
       return {quantity: 0, id: variantId}
     })

     jQuery.post('/cart/update.js', {updates: {val: 0}})
     .done(function() {
       console.log("success");
       window.location.reload();
     })
     .fail(function() {
       console.log("error");
       document.write("welcome to Javatpoint");
     })
     .always(function() {
       console.log("complete");
       document.write("welcome to Javatpoint");
     });
}

function addItemToCart(id) {

    const val = id.toString();

    var newVariantIdArray = new Array(val);

    var product_data = newVariantIdArray.map(variantId => {
       return {quantity: 1, id: variantId}
     }
     var data = {
       items: product_data
     }
```

```
    fetch('/cart/add.js', {
      body: JSON.stringify(data),
      credentials: 'same-origin',
      headers: {
        'Content-Type': 'application/json',
        'X-Requested-With':'xmlhttprequest'
      },
      method: 'POST'
    }).then((response) => {
      return response.json();
    }).then((json) => {
      /* yay! our products were added - do something here to indicate to the user */
      console.log('products', json)
      window.location.reload();
    }).catch((err) => {
      /* uh oh, we have error. */
      console.error(err)
    });

 }


function myfunction()
 {
document.write("welcome to Javatpoint");
        }
```

theme-app-extension/assets/image-gallery.css:

```
image-gallery figure {
  overflow: hidden;
}

#removeLink{
  color: black;
  background-color: whitesmoke;
  border: 1px solid black;
  border-radius: 2px;
  text-decoration: none;
  font:inherit;
  font-size: 14px;
  padding-left: 5px;
  padding-right: 5px;}
```

theme-app-extension/blocks/app-block.liquid:

```
<image-gallery data-id="{{ block.id }}">
<script defer src={{ 'app-block.js' | asset_url }}></script>

  {%- if block.settings.heading -%}
    <h2>{{ cart.total_price | money }}</h2>
  {%- endif -%}

  {%- if block.settings.description -%}
      <p>{{block.settings.description}}</p>
  {%- endif -%}



{% for collection in product.collections %}
  {% for product in collection.products %}

  {{ product.title }}
  <br>
  {% for option in product.options %}
    <label>
      {% for variant in product.variants %}

        {% assign isInCart = false %}
        {%for item in cart.items %}
          {% if item.id == variant.id %}
            {% assign isInCart = true %}
          {% endif %}
        {% endfor %}


        {% if isInCart %}

          <a id="removeLink" href="/cart/change?line={{ forloop.index }}&amp;quantity=0">Remove
item</a>

          {% comment %} <button id="add-{{ collections['hat'].id }}"
onclick="removeItemFromCart({{variant.id}}); return false;">Remove from cart</button> {%
endcomment %}
        {% else %}
          <button id="add-{{ collections['hat'].id }}" onclick="addItemToCart({{variant.id}}); return
false;">Add to cart</button>
```

```
        {% endif %}



      {{ variant.title }} - {{ variant.price | money }}
        <br>
      {% endfor %}


    <br>
    </label>
    {% endfor %}


  {% endfor %}
  {% endfor %}
<br>

</image-gallery>

{% schema %}
  {
    "name": "Image Gallery",
    "target": "section",
    "stylesheet": "image-gallery.css",
    "javascript": "app-block.js",
    "templates": ["product", "index"],
    "settings": [
      {
        "type": "text",
        "id": "heading",
        "label": "Heading",
        "default": "PRODUCT_PRICE"
      },
      {
        "type": "text",
        "id": "description",
        "label": "Description",
        "default": "This will display product options"
      }
    ]
  }
{% endschema %}
```

Section 2: Teamwork.

Our team has worked very well with each other throughout the semester. The project we chose required an extensive amount of research and logistical hurdles to overcome before we could collaborate on code together. At the midpoint in the semester we really hit the ground running and established a workflow to accomplish the task at hand.

Team member one contributed greatly to the group's documentation and testing requirements, along with UML and design pattern choices. This member also provided some styling on the front end theme app, Team member one contributed roughly 25% of the overall time committed to the project.

Team member two was the primary developer for the store front embedded app and merchant side app. They created the code and logic that connected products by collection, and displayed variants and their prices on the front end app. They also wrote the javascript functions that add remove products to the cart based on their product ids. Team member two was also responsible for the deployment of the merchant side app on heroku, the deployment of the theme app extension on the shopify store server, and creating the collaborative shopify development environment that all team members could contribute on. Overall team member two contributed roughly 50% of the overall time committed to the project.

Team member three contributed a small amount to every aspect of the project such as creating UML, generating ideas for project solutions, and attempting to code functionality into our store. With many roadblocks related to Shopify API and other technical issues, their progress was stifled. After team member two worked with them to overcome these issues, they were finally able to work more on the project itself. Overall team member three contributed roughly 25% of the overall time committed to this project.

Section 3: Design pattern.

An important design pattern used in the Shopify Embedded Theme App Extension was the State Pattern. The State Pattern is when an object changes its behavior based on a certain state property. A common example of the State Pattern is a vending machine: if there are enough coins in the machine it will dispense a drink, if there are not enough coins it won't dispense a drink. Whether it has enough coins or not is the state, and dispensing or not dispensing is the resulting behavior.

This pattern was applied to the product objects in the Shopify Embedded Theme App Extension. Each product displayed on the page had a functional button displayed next to it that either added or removed the product from the cart. If the product was already in the cart, then the button should remove it. If the product wasn't in the cart then the button should add it. We needed a way to determine which button would be placed next to each product when the product page loaded.

Without using the State Pattern, we would have had to have a separate dataobject to determine what functionality should be used on each product. Given Shopify's limited environment and static template format, this would have likely required an external database.

Instead, the State Pattern allowed us to determine the behavior of the object by inspecting its state. Whether or not the product was in the cart already is the state, and what button gets assigned to the product is the behavior. The state was stored in a liquid boolean variable called isInCart. Everytime the page is reloaded the isInCart variable gets refreshed by scanning the cart for the matching product. Then the isInCart variable is used to determine which button is placed next to the product.

The exact implementation of this pattern can be seen in 'theme-app-extension/blocks/app-block.liquid. A sample of the code is copied below for reference:

```
{% assign isInCart = false %}
{%for item in cart.items %}
        {% if item.id == variant.id %}
            {% assign isInCart = true %}
        {% endif %}
{% endfor %}

{% if isInCart %}

      <a id="removeLink" href="/cart/change?line={{ forloop.index
}}&amp;quantity=0">Remove item</a>

{% else %}

          <button id="add-{{ collections['hat'].id }}"
onclick="addItemToCart({{variant.id}}); return false;">Add to cart</button>

{% endif %}
```

Section 4: Technical writing.

Documentation For Users:

By adding our app to your shopify storefront, you will be able to accurately display cart totals beyond the 100 product combinations that shopify inherently allows.

Our app is still being developed, but a demo of our app interacting with a sample storefront can be found here. Password: admin

How to install and use our App on your shop:

Distributed downloads not currently available

In order to install our App, email esof423g5@gmail.com with the URL of your Shopify store, and we will send an install link to you as soon as we can. Using the link you receive, install the app and apply it to your store in your store's dashboard under the "Apps" subsection.
***

For additional information regarding installing Shopify apps on your storefront, follow this link.

A store front embedded app can be found in the online store builder. Locate the embedded block app under the "add block" menu, and place the block in the desired location on your storefront. We recommend directly underneath the product display. Once the embedded app is placed your customers will be able to view what options are available for the product, and the price.

Functionality: Our app requires merchants to upload their products and any desired variations as individual products. Items that a merchant wants to be included on the same product page should be grouped under the same "collection", which is included within Shopify's default functionality. Having variations listed as individual products allows us to dynamically calculate, and update the cart total.

If you are a merchant, and you would like to add a new product, or a variant of a product, to your store, from your Shopify admin homepage navigate to Products. From the Products page, select Add product. Input all the relevant details for the product you are adding. If you would like this new product to be offered as a variation or option on an existing product, make sure to include them in the same collection. Click save, and your product will be added to your storefront.

Reporting bugs: Please report any bugs to our development team at esof423g5@gmail.com, along with any error codes that you receive.

Documentation For Devs:

Store Installation/Deployment:

The app is set up to be run on a docker image deployed to Heroku. To set up the image the .env file must have the same api key, secret code, app url, and redirect links as what is listed in your shopify partner app dashboard. To deploy the app: push the root directory to an empty heroku app on your heroku account, build the image, and release the app using heroku's cli. Make sure the heroku app is set up to expose the correct port on run.

Once the app is deployed it should be possible to set up continuous integration with a github repository.

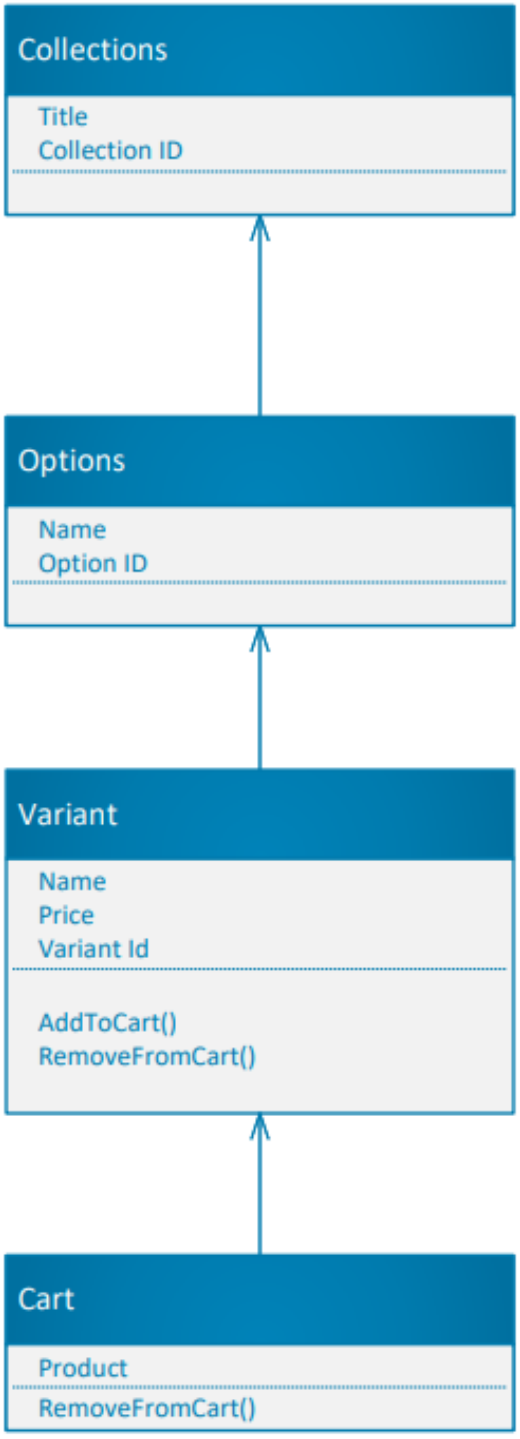
To contribute to this project: visit our github for the latest version of our software.

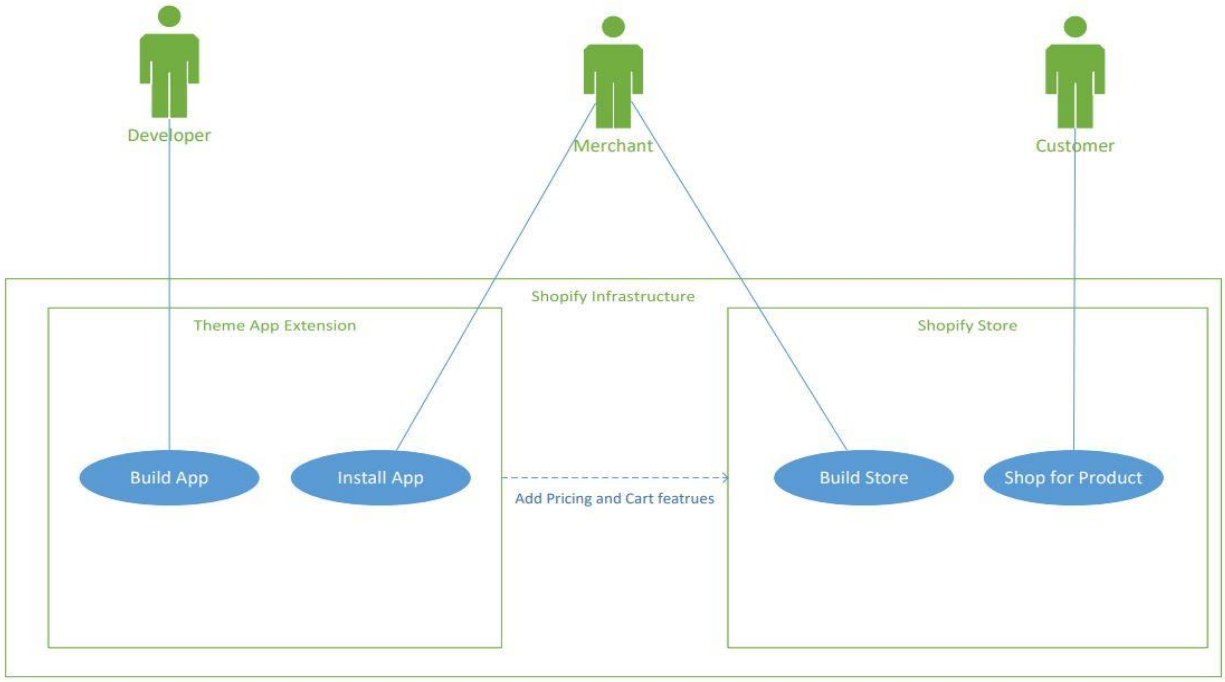
Tests: Every push is sent to our heroku server, and heroku ensures that the build is stable before deploying. Front end tests have been built using Selenium, and that test suite can be found on our github under the 'FrontEndTest' directory.

Reporting bugs: Please report any bugs to our development team at esof423g5@gmail.com, along with any error codes that you receive.

Known Bugs: See issues tab of github. Bugs verified through our developer email will also be shown here.

Section 5: UML.





Section 6: Design trade-offs

Our main design trade off was the decision to use a state pattern, as well as Shopify's own framework, to set up our store. We were able to use Shopify's collection functionionality to group objects into options for different products. This design choice allowed us to work entirely from within the existing framework without having to create a database or robust backend. In terms of work flow, this allowed our group to more easily work on our app by only focusing on creating the store itself instead of developing a database. For our potential customers, exclusively using Shopify functionality is beneficial to users already familiar with how it works. Someone who already understands collections, variants, and products will be able to understand and use our app effectively.

Section 7: Software development life cycle model.

<u>Methodology:</u>

The software development process that we used was Agile Management, specifically Scrum. We started the project with a product backlog which encompassed all of the requirements for our project. The semester was broken into seven Sprints, enumerated 0-6. Each Sprint was 2 weeks and was broken down as follows:

Start of Sprint: Leading into a new sprint, we would have a Scrum meeting with the group and parse the product backlog items that we would tackle for the coming Sprint. This was called our Sprint Backlog. We would also establish a goal for each Sprint in addition to the backlog. Each member would volunteer to accomplish tasks specified on the Sprint Backlog.

Middle of Sprint: Each day we would hold a daily Scrum meeting and each member would discuss progress that they have made. We would also discuss blockers that we ran into and collaborate ways to navigate around these blockers.

End of Sprint: All the Sprints were finished with a Sprint Retrospective meeting. During retrospectives the team would discuss what worked during the previous Sprint. We would also discuss what could be improved upon during the following Sprints and where we should start on subsequent Sprints. At the end of Sprints group members would submit individual group evaluations and update the Sprint Artifacts with their respective contributions to date.

Results:

This format worked well for our team. It promoted a flexible framework for development that allowed us to gracefully navigate blockers that emerged throughout the development process. There were no shortages of blockers while working with Shopify's API. The Scrum development cycle also provided us with a platform for consistent communication between group members which kept us on the same page. This is crucial to reduce redundancies throughout the development process.