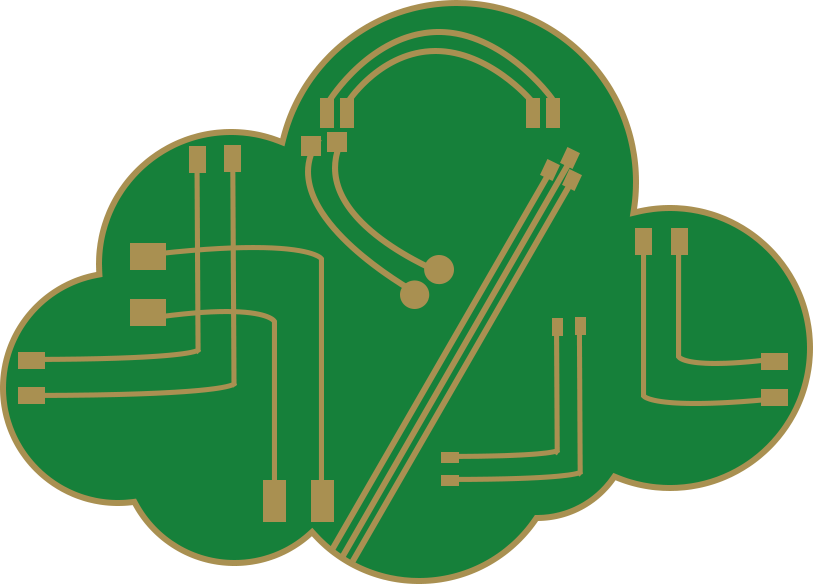
Cloud 9

CS 499 - Spring 2020

3/6/2020

## Team Members:

-Benjamin Ellis-

-Alex Reel-

-Camron Savage-

-Alex Tanner-

-Patrick Yoder-

-Yankai Zhao-

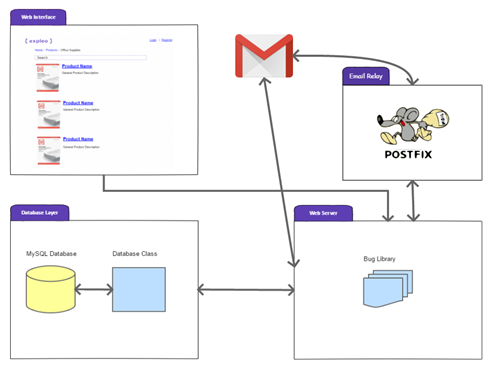
# Test Automation

## Customer:

Ben Fox - Trissential

Edited Using Google Docs

# High-Level Design (Architecture)



This is the architecture left behind by the previous group who built the base of the project. We plan to leave all of the same basic architecture in place, and therefore it is unnecessary to alter the diagram. Right now, we need to figure out how the email relay worked as that we not documented by the previous group. The changes we are making will be to components of the web interface, web server, and database, adding functionalities to them. But the basic architecture will remain the same.

# Detailed Design

The methods described below are divided into two types: new methods and legacy methods. As we are expanding on an existing project, we will first describe new methods we are adding to the project. The second section with give a brief description of the methods that existed in the code base that we inherited. To understand the full architecture of the project as it will stand at the end of production, it is important to have a clear picture of the entire project, but we put emphasis on the new methods that will be added over the course of the semester.

## 2.1 New Methods

### 2.1.1 Wishlists

A new button will be added to each product and product page, allowing the user to store the item in a wishlist. This will be a list of items that the user has saved but has not placed into their cart. The user will be able to have multiple wishlists that these products can be added to. From a wishlist, the user will be able to add items to their cart. The wishlist can be made public or private, and it will be possible for other users to see public wishlists.

### 2.1.2 Re-Order

It will be possible for a user to go into their list of past orders and press a button to add the same items in the same quantities to their cart. From there, the user can adjust the quantities and remove or add other items as normal.

### 2.1.3 Multiple Addresses

Users will be able to go to their account page and save multiple addresses to their profile. A user will have the ability to choose any of their saved addresses when placing an order.

2.1.4 Repeating Orders

A new page will be created which allows a user to set up a repeating order. This is an order that the users wants to receive multiple times, chosen on some specific time interval.

### 2.1.4 Store Administrator

A new type of user, a store administrator, will be created. This type of user will not be able to load bugs for other users like a full administrator, but will be able to interact with the store, including adding and removing products, changing prices, etc.

#### 2.1.4.1 Promotional Codes

Store administrators will be able to create promotional codes that a user can implement to receive discounts on their orders.

#### 2.1.4.2 Store-wide Discounts

Store administrators will be able to set store-wide discounts on specific items or on all merchandise in the store.

## 2.2 Legacy Methods

### 2.2.1 Login

Users will submit their username and password, which is checked for validity.

### 2.2.2 Registration

Users submit username, password, and other ecommerce information (such as name, address, and so on). The user is then sent an email to verify the new account.

### 2.2.3 Search

The user can search for any amount of a product name, which is queried against the database.

### 2.2.4 Filter

Filtering options exist as buttons and sliders. When filters are selected, the product list display is filtered by querying the database against the filtered options that were chosen.

### 2.2.5 Product List Display

Existing as its own class so as to be reusable, this constructor method builds the html code for displaying the current database results.

### 2.2.6 Database

The database class builds a connection to a MySQL database, which includes all data for the site (user information as well as product information).

### 2.2.7 Order History

Queries the database for any orders made by the querying user.

### 2.2.8 Cart

The cart queries the database for any orders that have not been completed.

### 2.2.9 Checkout

Marks the items in the user’s current cart as a completed order in the database.

### 2.2.10 Admin Functionality

This allows an admin to look at a list of users and bugs, and assign bugs to certain users. This forces those bugs to become active for that user’s profile.

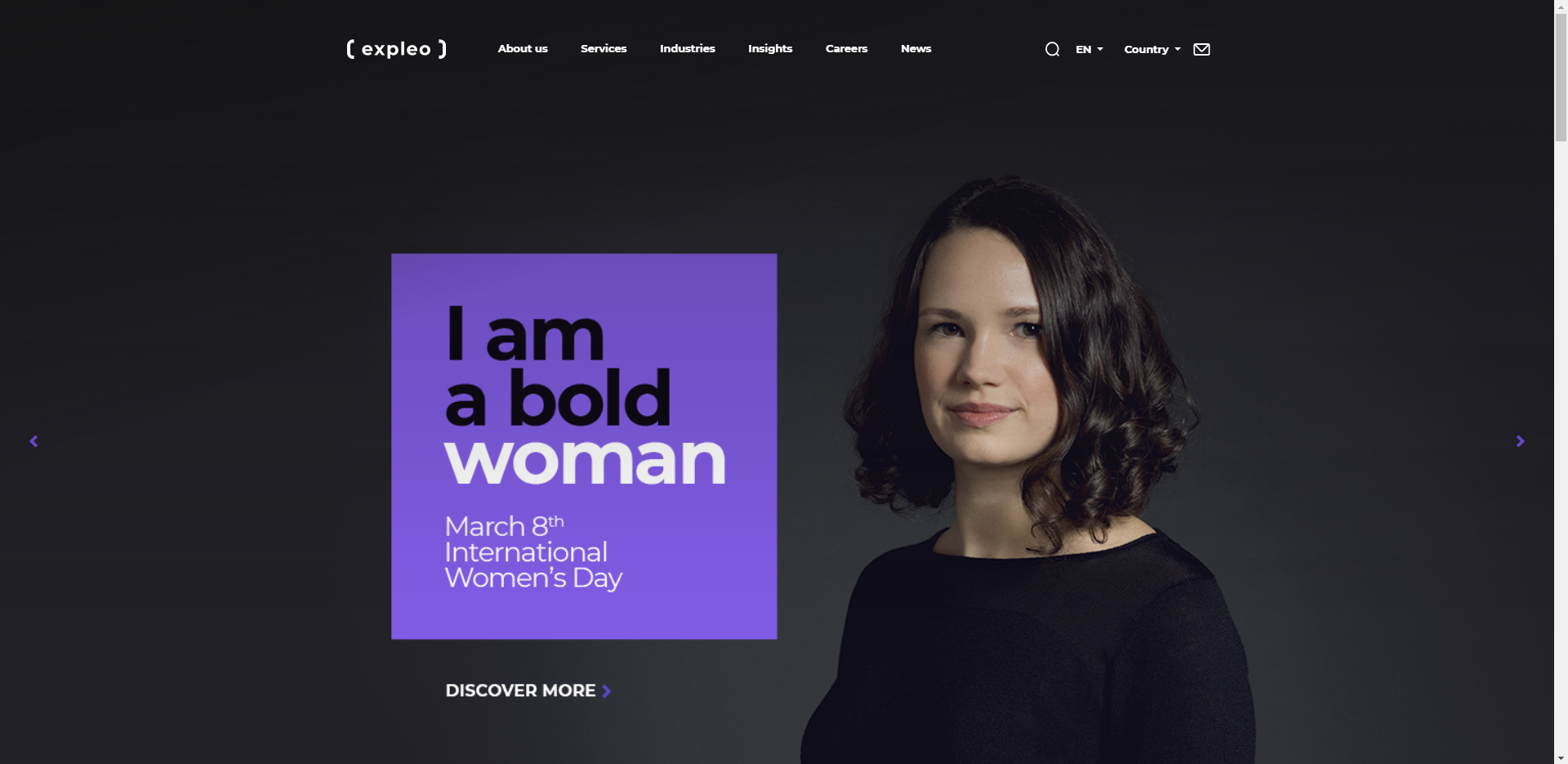
### 2.2.11 Bug Framework

The bugs are loaded onto user pages as separate php objects that can be toggled on or off by an administrator.

## 2.3 UI Design

The application we are creating relies on the existence of several pages, all of which have relatively unique UI requirements. This section will step through the general UI design of the majority of the significant pages. Since we are expanding upon an existing project, some of the web pages have already been designed. As such, the UI for new web pages will be discussed first with legacy web pages discussed afterwards. The new web pages include the wishlist, past orders, and store administrator pages. Beyond this selection the UI of existing web pages will be described with a discussion of any intended changes to them. Existing UI functions as needed, but requires the overall design be reworked to better match Trissential’s parent company's website, expleogroup.com, as per the request of the customer. These changes will likely be aesthetic in nature and will not change the overall function of the UI.

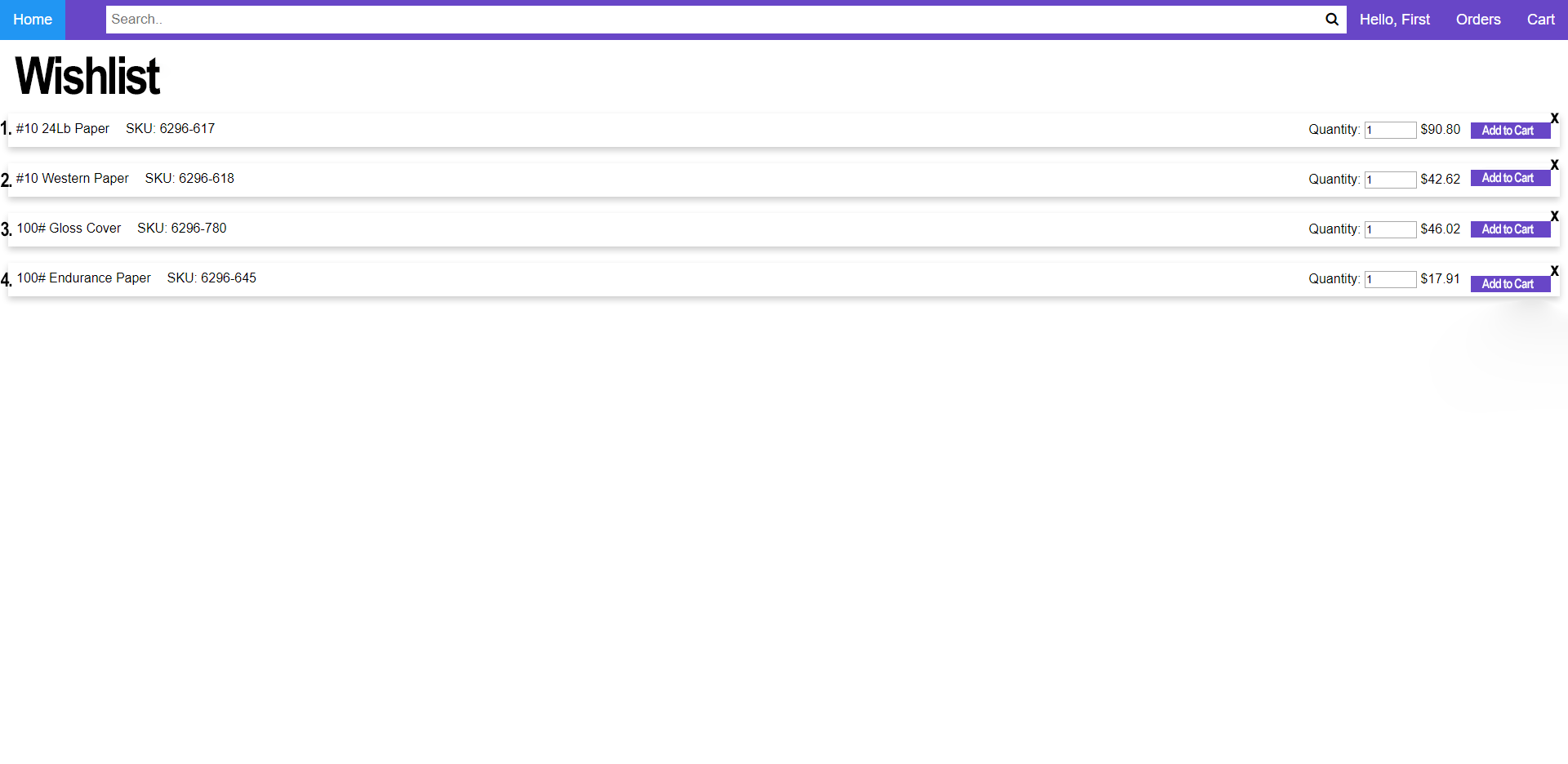
**Figure 1.2 ExpleoGroup.com UI**



### 2.3.1 Wishlist

A wishlist page will be required to compliment the introduction of a wishlist feature to the storefront. Such a page will need to maintain the header of the website for navigational purposes, as well as support wishlist features. In doing so, it will be necessary that the items which the user have added to the wishlist will be displayed here in a numbered list format. There should be a button for each item that exists which will allow the user to add that item to their cart from their wishlist page. Additionally, the user should be able to remove and reorder wishlist items. Users should be able to easily navigate from their wishlist to the product page of an item on their wishlist by clicking on the product.

**Figure 1.3 Wishlist UI**

****

### 2.3.2 Past Orders

For the past orders web page, it is important to show users what their previous orders have been in a clear and concise manner. We can achieve this by listing out previous orders with identifying order numbers with collapsable lists of their contents. This can be done with a simple arrow button as is typical of such a feature. Additionally, it is critical that the price of the entire order is displayed alongside each order. In addition to informing users of their past orders, a button can be placed on each order which would enable users to quickly make an identical order again. Implementing all of these features will likely have a similar look to the wishlist and cart pages, as both additionally have a goal to inform users of relevant products at the time of viewing.

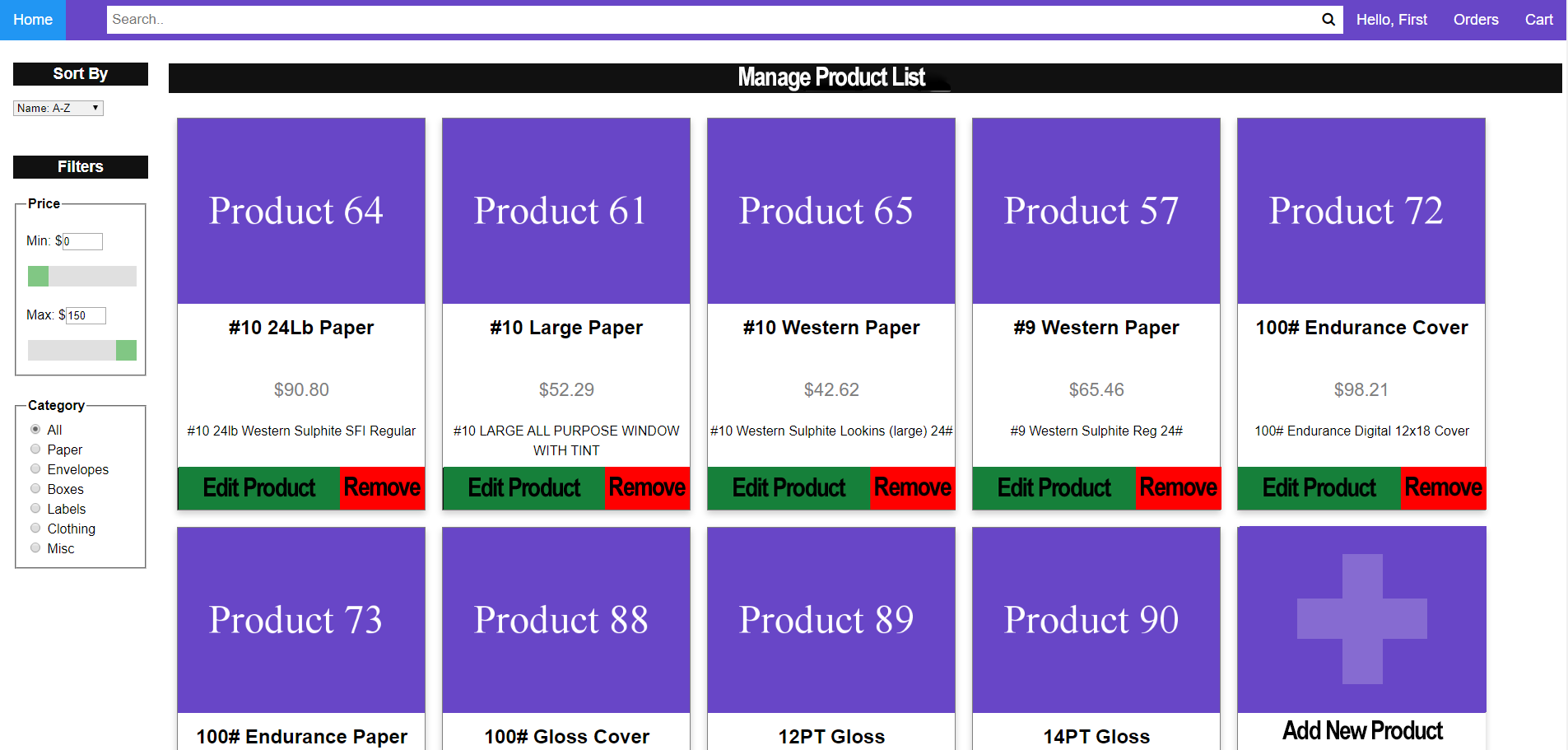
**Figure 1.4 Past Orders UI**

****

### 2.3.3 Store Administrator

With the addition of the store administrator role, comes the necessity of a store admin web page which enables such users to enact their abilities. Such a page would require the user to be able to add, remove, and edit store products. This can be implemented with a UI that shows store admins all products in the store in a manner very similar to the product list page. Altering the product list page by creating “new product” and “remove product” buttons would enable the store admin to function in their role. Additionally, an edit product option can be implemented in this page to allow for the modification of existing product pages.

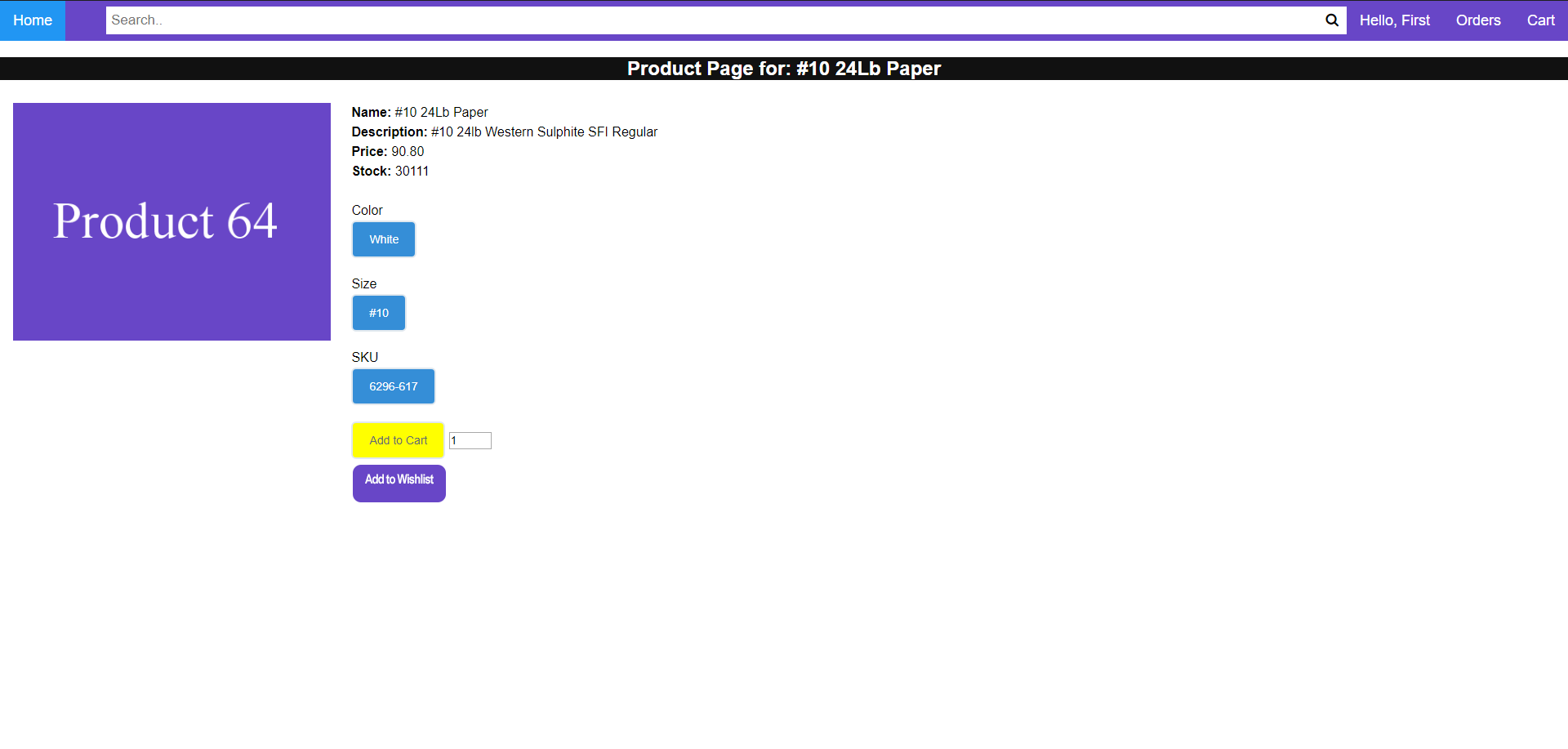
**Figure 1.5 Store Admin UI**

****

### 2.3.4 Product Page

The purpose of the product page is to show the user all information relating to a specific product. As such, it is necessary that the product page UI includes an space for images of the given product in addition to a description of the product. If the product has any options the user must choose, such as color or size, it is necessary that appropriate buttons exist on this page. Additionally, the price of the product is necessary information the user must be able to see on the page. In order to enable the purchase of products, an add to cart button is critical. Implementing the wishlist feature would require the addition of an “add to wishlist” button near the add to cart option.

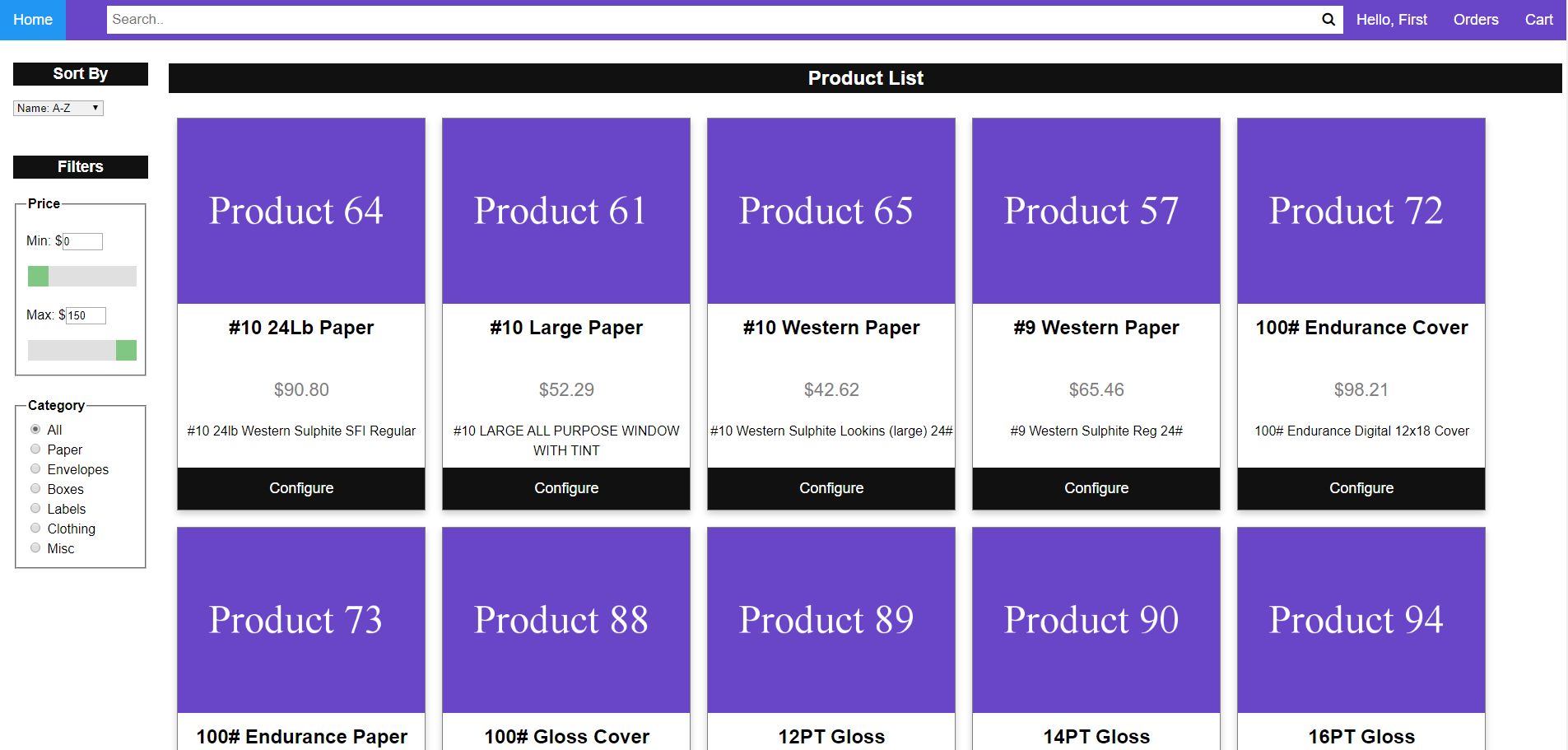
**Figure 1.6 Product Page UI**

****

### 2.3.5 Product List Page

The purpose of the product list is to allow users to browse the storefront and view all products which fit a selected criteria. As such, a portion of the UI must allow users to add filters to the displayed content. These filters include price limitations and categories for the products displayed. The largest area of the web page will be where products are displayed in tiles with basic product information on them. Each tile will feature a product name, image, price, and a button to view the corresponding product page. Tiles will be laid out in a grid which users can scroll through to find the product they want. In addition to filter options and the main display, the header and search bar are also crucial to this page as they allow users to better navigate the site and find products they want.

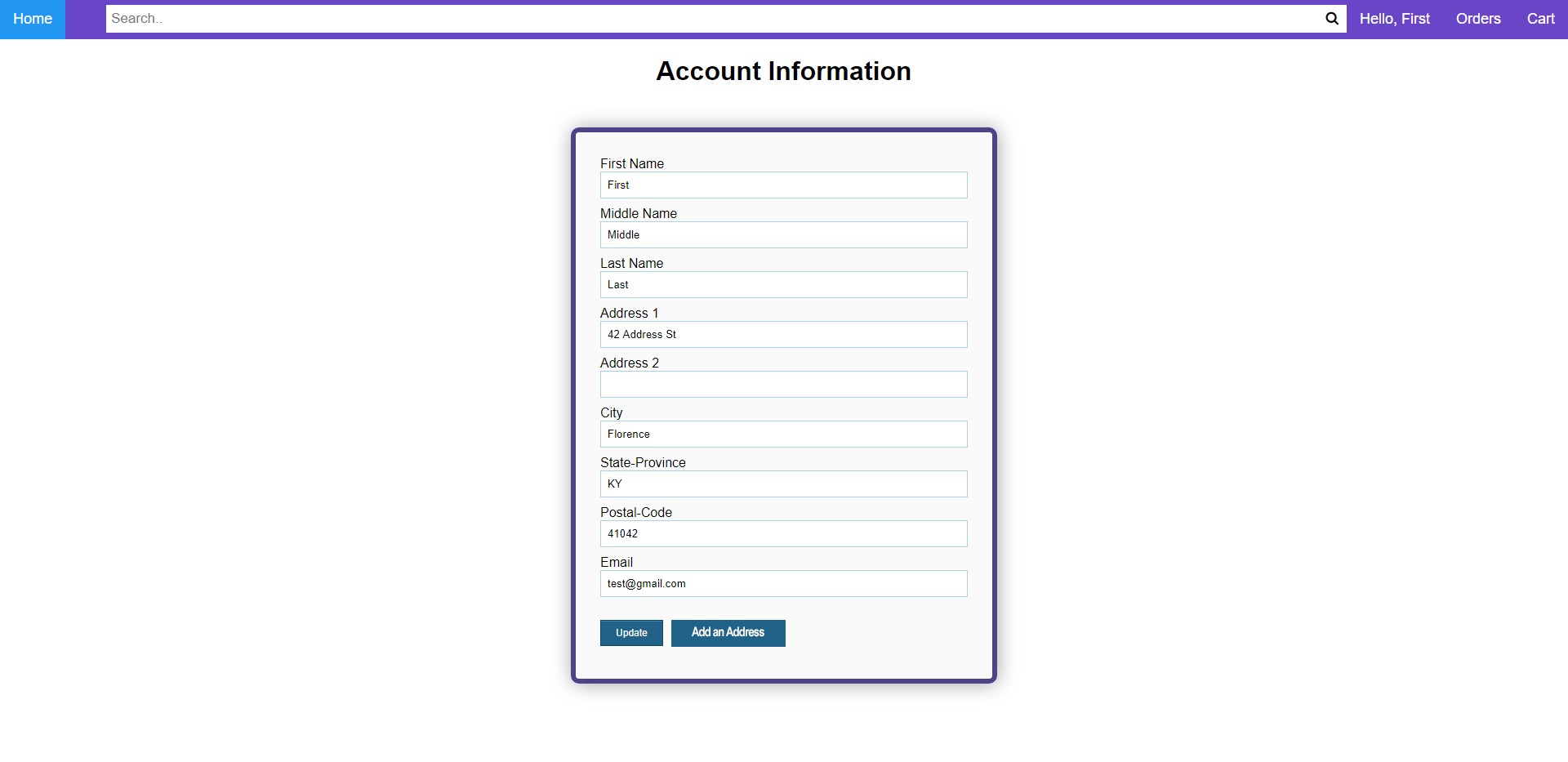
**Figure 1.7 Product List UI**

****

### 2.3.6 User Account Page

The goal of the user account page is to allow users to view and edit their account information. This information includes their email, name, and address. The account page will display the currently stored information for the user in text boxes which the user can change. Changes can be confirmed with an “update” button. In order to accommodate our new feature which enables users to have multiple addresses, an additional “add an address” button is necessary on this page. Such a button would allow users to add an additional address to their account which would allow for easier checkout.

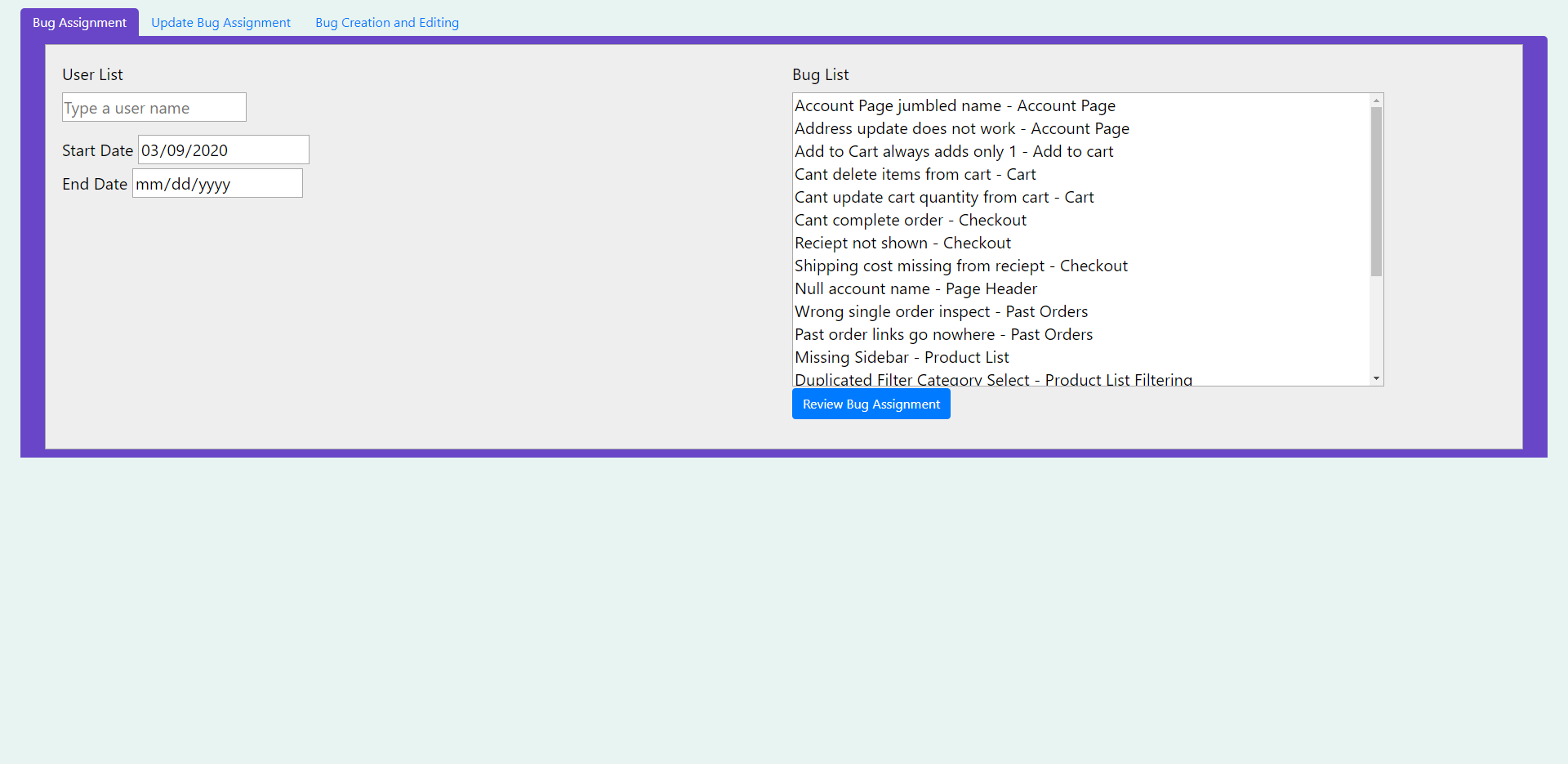
**Figure 1.8 Account Info UI**

****

### 2.3.7 Administrator Page

The administrator page is much different from the rest of the website. The goal of this page is to allow administrators to create and edit bugs, as well as assign bugs to specific users. As such, three tabs are necessary on this page, one for assigning bugs, one for updating bug assignments, and one for bug creation and editing. Admins will be able to switch between these tabs quickly by clicking on the tab they want. There is no need for the typical website header to be on the administrator page, so the search bar and header buttons will not be available on this page. The tab for bug assignment will feature a text box for admins to enter a user name, a box to establish a start date for the assignment, a box to establish an end date for the assignment, and a final box which identifies which bugs to assign to the given user’s account.

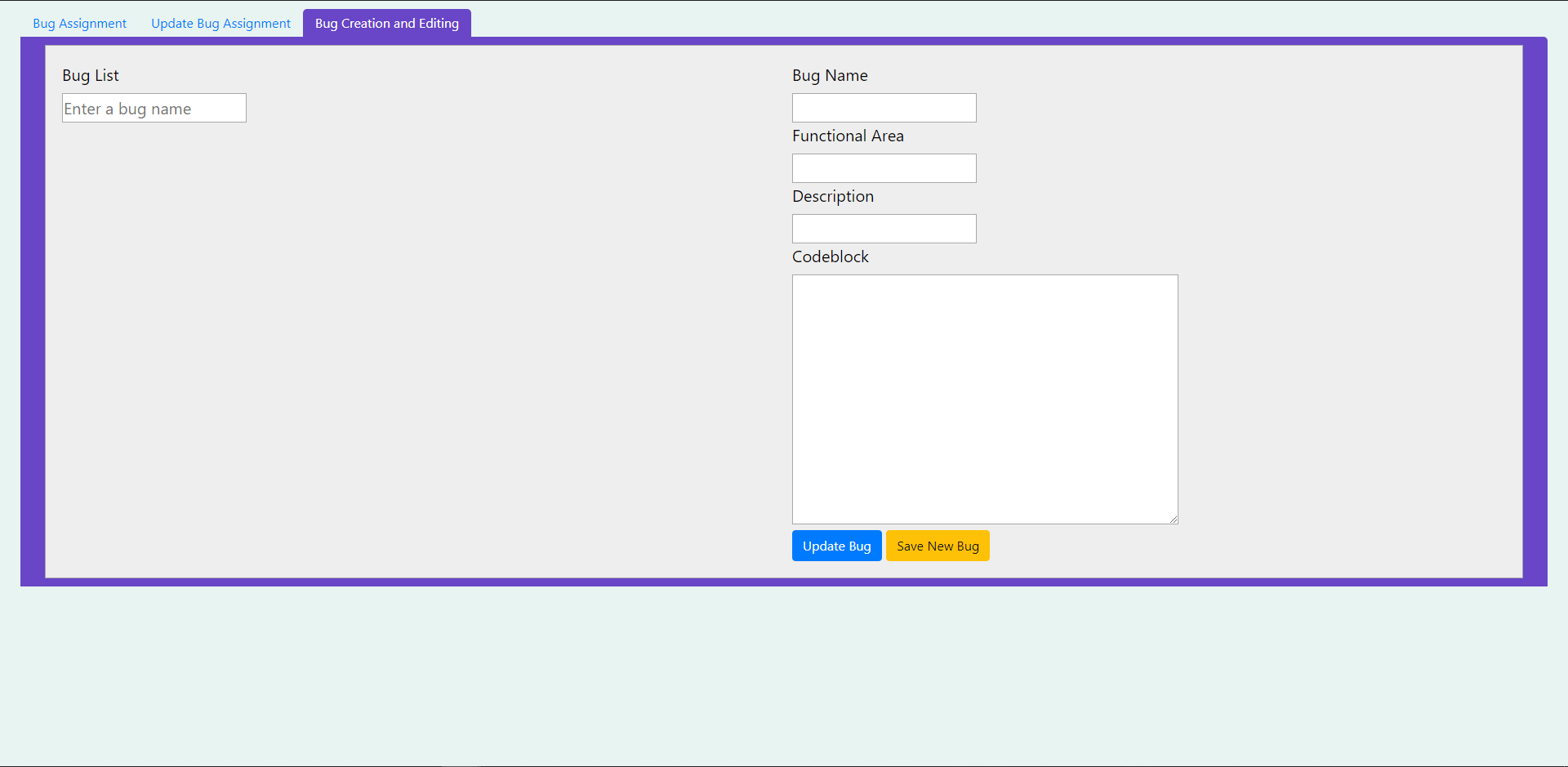
**Figure 1.9 Bug Assignment UI**

****

The second tab of the admin page will allow administrators to view and update bug assignments for any specific user. It is necessary for this tab to allow admins to identify the user they want and view/alter their bug assignments. As a result, its UI is fairly similar to the bug assignment UI featured in Figure 1.9.

The third, and final, tab of the admin page will allow for the creation and modification of bugs. Admins will be able to use a text box to enter a bug name to find a bug to edit. Additional text boxes will allow the admin to change the bug name, function area, description, and associated codeblock. Since bugs can be both edited and created on this page, it is necessary to have both an “update bug” button and a “save new bug” button as shown below in Figure 1.10.

**Figure 1.10 Bug Creation and Editing UI**

****

## 2.4 Design Pattern

Since the focus of this project is the expansion of an existing web application, the design pattern we will use will be the same as the design pattern used in the application’s original development. This pattern is the client-server model. The client-server model is a good fit for this application, as the server can be used to maintain the web stores database while the client handles how the user can interact with the database through the storefront. Such a design pattern allows for the easy implementation of storefront features and bug assignment abilities.

# Testing

Our testing will consist of a combination of automated testing and manual developer testing. During the semester we plan on learning and utilizing Selenium on Chrome for our automated testing, given its utility for web-based applications. Our current list of test cases can be seen below. This list will likely expand over the coming weeks as defects arise during the coding phase. Additionally, there does not appear to be any existing test scripts for the application. We are currently in conversation with our customer about whether writing tests for existing functionality is a desired use of our development time.

|  |  |  |
| --- | --- | --- |
| **Test Area** | **Test Description** | **Expected Result** |
| Past orders functionality | Complete an order. Go to past orders page. | Past orders page exists and correctly displays completed orders for the user. |
| User account page loading | As a user, click on the account page link. | Application redirects to user account page and correctly displays the user’s information. |
| User account updating | In the user account page, change data in the form and click on the update information button. | The page refreshes and displays the new information. The user’s data in the database is also updated. |
| Enforced character limits | In application form fields, attempt to enter a large number of characters. | An on-screen error is shown indicating that the field only accepts up to x number of characters (x will vary based on the field). |
| Order complete notification | Complete an order, check the email account of the relevant user. | An email was received notifying the user that their order was completed and what products were purchased. |
| Add to wishlist | Search for a product, click on the add to wishlist button. | The correct product has been added to the users wishlist. |
| Wishlist view | Click on the user’s wishlist | The page loads with the correct products. Wishlist products also appear in the database. |
| Wishlist privacy | On the user’s wishlist, set the privacy option to private or public. | The attribute is updated in the database to reflect the change. |
| Wishlist search | Search for and click on a wishlist. | The search shows items relevant to the search. Only wishlists set to public are shown. Clicking on the list directs the user to the correct page. |
| Repeating order | At checkout, set the order to repeat with some timeframe. | The order is processed again at the appropriate time. The user is sent an email to notify them of the order. |
| Promotion codes | Enter a promotion code at checkout. Finish the order. | If valid, the promotion code applies to the order and gives the user a discount. Otherwise, notify the user they entered an invalid code. |
| Add to cart from search page | While searching for products, click on the add to cart button. | The selected product has successfully been added to the user’s cart. |
| Compact list view | While searching for items, click on the compact view button, then the regular view. | The page view changes to have thumbnails or not depending on which button was pressed. |
| Free shipping | Add enough products to the cart so the price exceeds the free shipping price point. | The shipping cost for the order becomes zero and is reflected in the final price. |
| Store admin user | Create a user as a store admin, login as the new user. | The user is able to view and enter new pages to add and update products, set the free shipping price point, and set product discounts. |
| Add product | As a store admin, enter the add product page. Enter information into the form and click save. | A new product is added to the database with the appropriate information. The product is also visible by searching for it in the application. |
| Update product | As a store admin, select a product and click the update button. Change its information in the form field and click save. | The product’s information is changed in the database as appropriate. These changes are also reflected within the application. |
| Reorder bug | With the reorder bug assigned, resubmit an existing order. | Items that were not part of the original order are added to the cart. This is reflected in both the database and front-end. |
| Dead link bug | With the dead link bug assigned, attempt to click on the links marked to be broken. | The user is not directed to the requested page. Nothing happens |
| Cart size limit bug | With the bug limiting cart size assigned, add some number of items to the cart beyond the limit. | After the limit is hit, no more items are added to the cart. This is reflected in both the database and front-end. |
| Case sensitive search bug | With the case sensitive search bug assigned, search for items using a string that mixes capital and lowercase letters. | The search returns no items unless the cases are precisely the same as the product name. |
| Large cart bug | With the large value cart bug assigned, add enough items to the cart to put the value over $1,000,000. | A special effect is shown on the screen. |

# Quality Review

Most of the work we’ve done so far has been organizational and figuring out the structure of the previous class’s project and getting it working on our systems. We have noticed an issue that the previous structure doesn’t include any documentation on a method that is critical for the program, so we will need to figure it out from the raw code. This method would validate an email address when a user signed up for a new account for the ecommerce store. We have yet to get this method working so far.

The previous group also talked about writing automated test scripts in their documentation, but never completed that task, so it’s something we will need to begin from scratch.

# Metrics

## 5.1 Estimated Story Points / Number of User Stories

|  |  |  |
| --- | --- | --- |
| **Number** | **Story** | **Story Points (1-10)** |
| 1 | Improve website aesthetics. | 6 |
| 2 | Confirm/fix the current condition of the “past orders” functionality. | 5 |
| 3 | Add a page for user account information. | 4 |
| 4 | Enforce character limits on form fields where appropriate. | 2 |
| 5 | Ensure that all form fields have unique IDs for easy development in the future. | 2 |
| 6 | Add email support to notify users of completed orders. | 4 |
| 7 | Add wishlist functionality that users can save and use to add items to their cart. | 8 |
| 8 | Allow for automatically recurring orders. | 5 |
| 9 | Add promotional codes and store-wide discounts. | 5 |
| 10 | Display a dynamic image on the store’s front page. | 1 |
| 11 | Allow for adding to cart on the search page. | 2 |
| 12 | Allow changing between thumbnail and a compacted list view on the search page. | 2 |
| 13 | Allow for the setting of a free shipping price point. | 1 |
| 14 | Allow assigned bugs to be set as intermittent, so an admin can set how often the bug can occur. | 3 |
| 15 | Add multiple levels of users. | 8 |
| 16 | Add new bugs. | 7 |

## 5.2 Product Effort

We have been meeting each week for about two hours at a time. We will begin coding soon, which will require additional work hours outside of these weekly meetings. We estimate approximately 63 hours of work so far, given differences in when people could be available for each meeting so far.

# 6 Developer Notebooks

We are keeping our developer notebooks in the Wiki section of our Github repo:

<https://github.com/GreenAlex96/StorefrontTestingApp/wiki>

## Authors and Word Counts:

**Alex Reel (830 words)-** Testing

**Alex Tanner (1187 words)-** Structure of document, high-level design, new methods and legacy methods, quality review, metrics

**Benjamin Ellis (1357 words)-** UI Design and Design Pattern