Storefront Testing App

Architecture

Team 4 - Bugs are Features Too

University of Kentucky - CS 499 - Fall 2019

Customer: Ben Fox, Trissential

October 18, 2019

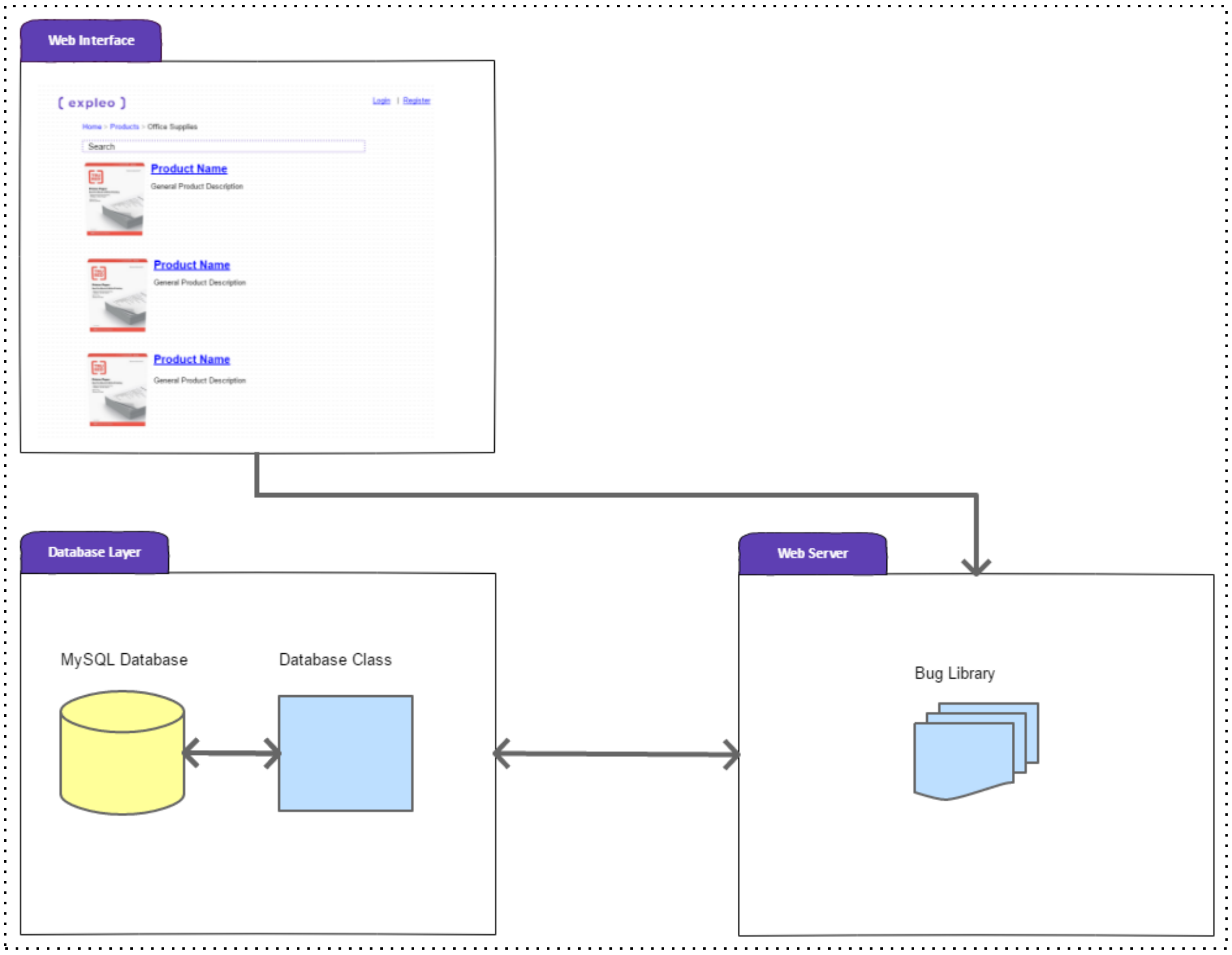
Authors:

Seanna Lea LoBue - High Level Design, Method Design, UI Design
Michael Murray - Testing, Metrics, Method Design, UI Design
Eric Prewitt - Method Design, UI Design
Nicholas Wade - Method Design
Kee West - Method Design, UI Design

1. High Level Design (Architecture)

Our web store bug testing tool is designed to provide a customizable storefront to Trissential employees. This will allow teams to improve their skills in building Selenium scripts to perform automated testing to locate website flaws via automated search, product and option selection, and order creation. In order to do this, the web-based tool will allow an administrator to login and access a list of bugs and users, assign bugs to a user for testing, and then edit and create additional bugs for future testing.

Figure 1.1 High Level Design for Bug Delivery Storefront



This web site tool uses the standard client-server model and involves three main components. The topmost component is the web interface. This interface provides both storefront functionality as well as administrative functionality. The users will be able to click on items, display content pulled from the database, and make changes to the database items. From the employee perspective, this would be through placing an order. The administrative function will allow the selection of a user and the assignment of bugs from a list. Additionally, they will be able to access a list of bugs, load the bug data to a window, and either update and save the bug or to create a new bug from either loaded or raw data. The web server component will read from the database and serve out web pages based on the bugs assigned to the logged in user. The database layer will build and maintain a store database and a bug database. It will serve out database content and allow for the updating and maintenance of bug information.

2. Detailed Design

2.1 Method Design

In the subsections below we will describe the methods we plan to implement written out in pseudocode. In order to provide a stronger understanding of the nature of the methods described, each method or methods will be grouped by a section header defining the section of the project that it belongs to. While additional insights will be discovered during the course of actually implementing these systems, we hope that by detailing the methods in this form we can establish a more uniform vision for how these methods will be designed, assisting in module integration.

2.1.1 Login

On load, this page will establish a connection to a database containing user info and display a form for the user to input their username and password into. Once the user submits their info, the page is reloaded with a POST request containing this information. When the POST data is seen, the page will then check the user’s input for format validity and displays an appropriate error message if it is invalid. Then, it will take the user’s information and send a query to the user information database to obtain the user’s information. If it does not, the user will be prompted that there is not an account with that username/email or that their password is incorrect. If they entered the correct information for an account, they will be logged into the website using a cookie to keep track of their login throughout the website. There is also a “I forgot my password” link, which will allow the user to send a link to their email that will allow them to regain access if they forget their password.

2.1.2 Registration

On load, this page will establish a connection to a database containing user info and display a form for the user to input their desired username/email and password into. When the user submits their info, the page is reloaded with a POST request containing this information. When the POST data is seen, the info is checked for validity and displays an appropriate error if the info is of an invalid format. It will then take the user’s information and send an insert query to the database containing the user’s information, plus a calculated user id. The user is then sent an email using the given email to validate their account.

2.1.3 Search

Search bar will consist of a textbox and a search button.

When either the enter button press event is detected when the textbox is the active element or the search button is clicked on, the text content of the search textbox is saved. A database query will be formed selecting from the product table, where item name and item tags will be compared against user search entry. Any elements of the table that meet the criteria will be extracted from the database and forwarded one at a time for display to the product list display method described below.

2.1.4 Filter

On the product list page, a number of relevant filtering options will exist in the form of radio buttons, option selections, dropdown menus and slider bars.

All filtering options will possess an identical class type so that any click of elements of this class will trigger filtering logic. The state of the filters will be read and compared to the initial or preceding selection state. If the state of the filters has been changed, cluster of selections will trigger a condition for the composite database query, where a running query string is concatenated with each additional condition. After all filter elements have been reviewed, the query string will be sent to the database and all relevant elements of the product table will be forwarded one at a time to the product list display method described below.

2.1.5 Product List Display

Existing as its own class to encourage reusability, this constructor method will build the html code for displaying the database results in a visually appealing manner.

Taking in one database query return row, the title, image, price, description and button link to product page will be displayed in a card format. Button tagging will be done dynamically based upon the name of the product, where whitespace will be eliminated to produce feasible tag values. Button linking and form submission data will be related to this same name id and while call the product page with the appropriate submission data for the individual card generated.

2.1.6 Database

The database back end is currently a single page, but is in the process of being converted into a class to encapsulate the functionality and make it available to all other classes within the tool.

The database class builds a connection to a mySQL database. The buildDatabase function then checks for the existence of the database. If it does not find that database in the schema, the function then creates both the database and each table within it via a number of database subfunctions. If the database table is empty, then the function reads from raw files to insert a standard set of products within the “storefront” section.

The update function will take a record identifier, update parameters and table identifier to build a single unified update function that will be available across all classes. The insert function will take a table identifier and website values in order to add a new entry in the identified table. A selection function will allow each page to request the appropriate information from the database ranging from product categories to order history.

2.1.7 Order history

On load, this page will establish a connection to the database and then check if the user is logged in. If the user is logged in, it will query the database for any past orders made by that user and then display them as a list.

2.1.8 Cart

The cart will, on load, query the database for the order that is marked as not being completed. If there is an incomplete order, it will display the items in the order along with their price, quantity and the running total for the order. There will also be a button next to each item to remove that item from the order along with a way to change the quantity of that item. The page will have buttons for continuing shopping and checkout which will commit changes to the cart before taking the user back to the storefront or the checkout page respectively.

2.1.9 Checkout

We intend to implement many different functionalities into the checkout. Each user that logs into the website will be designated a personal cart containing any items they have decided to purchase. When checking out, we plan on implementing the option to allow users to choose shipping speeds, payment methods and delivery locations with the ability to manipulate those as necessary. When the user decides they would like to proceed with the checkout they will be provided with a page that displays all of the order details and allows for user confirmation and cancellation. Extra features we would like to implement, if time permits, would be the option to determine if the product is a gift, enter promotional codes for discounts and the ability to save orders for a future date. All actions and purchases by the user in the checkout will be stored in the database for access at a later time if so desired. Each individual checkout will be accessible by the administrator and the pertaining user only.

2.1.10 Admin functionality

The administrative page will have two sections. The bug assignment section will allow the administrator to view a list of users, select a user. They will then be able to select bugs from a searchable and filterable list and assign one or more bugs to the selected user. That information will be saved to the database, where it will be served to the user upon their next log in.

The bug editing section will allow the administrator to locate an existing bug, display the bug code in an editable pane, and updating bug metadata or code. That bug data can also be saved as a new bug. If the administrator prefers to create their bugs from scratch, then they can instead use the bug data display framework to create a new bug and save it for the database. Those new or updated bugs will then be available as soon as the administrator goes back to the bug assignment section.

2.1.11 Bug framework

On the backend the bug framework will consist of a table for bugs and a table of which bugs are active for each user. The bug table will have columns for the page that the bug is going to be on, the section of the page where the bug goes and an id unique for that section of that page (these three things being a composite key). The final column of the bug table will be a pointer to a file that contains that section of code for the page with ID 0 being the file with clean code. The user table will have the user ID for its primary key followed by columns for each section of each page which will store the bug ID that is active or 0 if not active. When any page besides the admin page loads it will query the user table for the columns relevant to the page on the row of the current user, and then the page will section by section query the bug database for the code that matches that page, section, and id from the user table as a PHP include statement of the file that is returned by the query. Admins will be able to load these bug files in the admin page then edit and save them as the same bug or as a new bug which will add new entries to the bug table while new user creation will add to the user table with default of ID 0 for all sections.

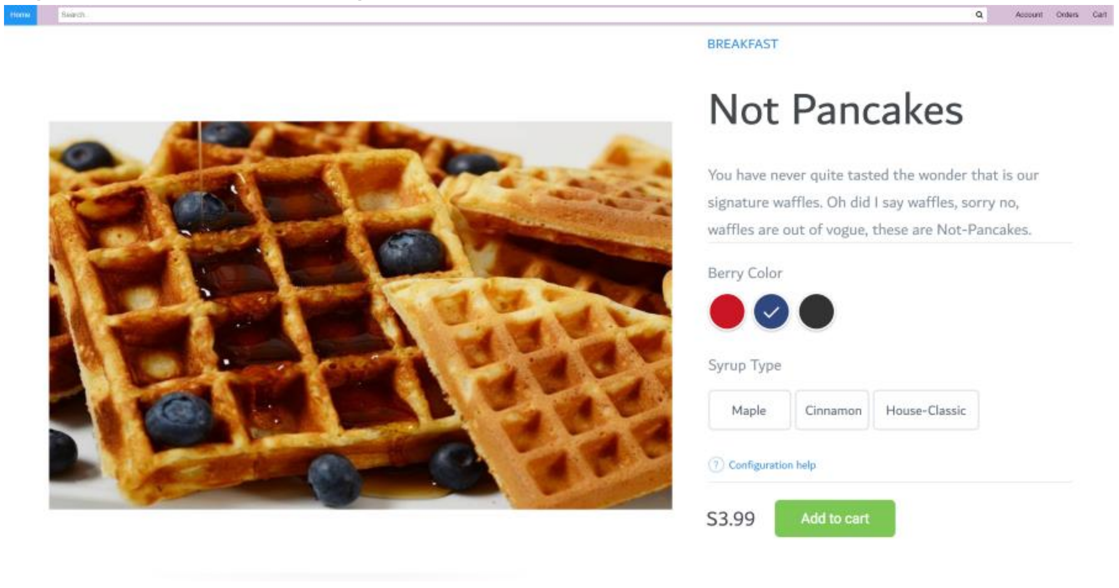
2.2 User Interface Design

The storefront we are designing is composed of a number of webpages connected together to provide the underlying e-commerce functionality. Below we will show our intended user interfaces for the core pages of interactions for both the testing and admin users. However, a number of additional transitory pages will exist in the end product which will be visually consistent but are not of sufficient importance to be listed below. While elements are subject to change in terms of position and layout, the elements demonstrated below represent a good faith approximation of our design intent and requirements for user interaction/experience.

2.2.1 Product Page

Although we have not yet implemented the individual product pages, we do have a base form from which we will model them after. While this website is being designed with the idea that it will be used for demonstration and training purposes, we still intend on making our pages as functional as possible. Each product in the list will have the ability to be clicked on for access to a unique product page individually detailed around the product. Core functionalities we intend to implement are the ability to add each individual item to the user checkout cart, as well as a separate button that will redirect the user to their personal cart. Each product page will display an image of the product, customization options as well as a brief description of it and the price. Each page will also present buttons that allow the user access to different areas of the website and a search bar to redirect to a searched product. If time permits, we intend on implementing a couple features that will offer a more professional aesthetic to the design including the ability to add products to a user designated wish list, display similarly related products on the page, change the quantity of the product to add to the cart and the option to choose the size of the product, if it pertains. All product pages will be stored in the database and accessed after user or administrative login to the website.

Figure 1.2 Product Page UI



BREAKFAST

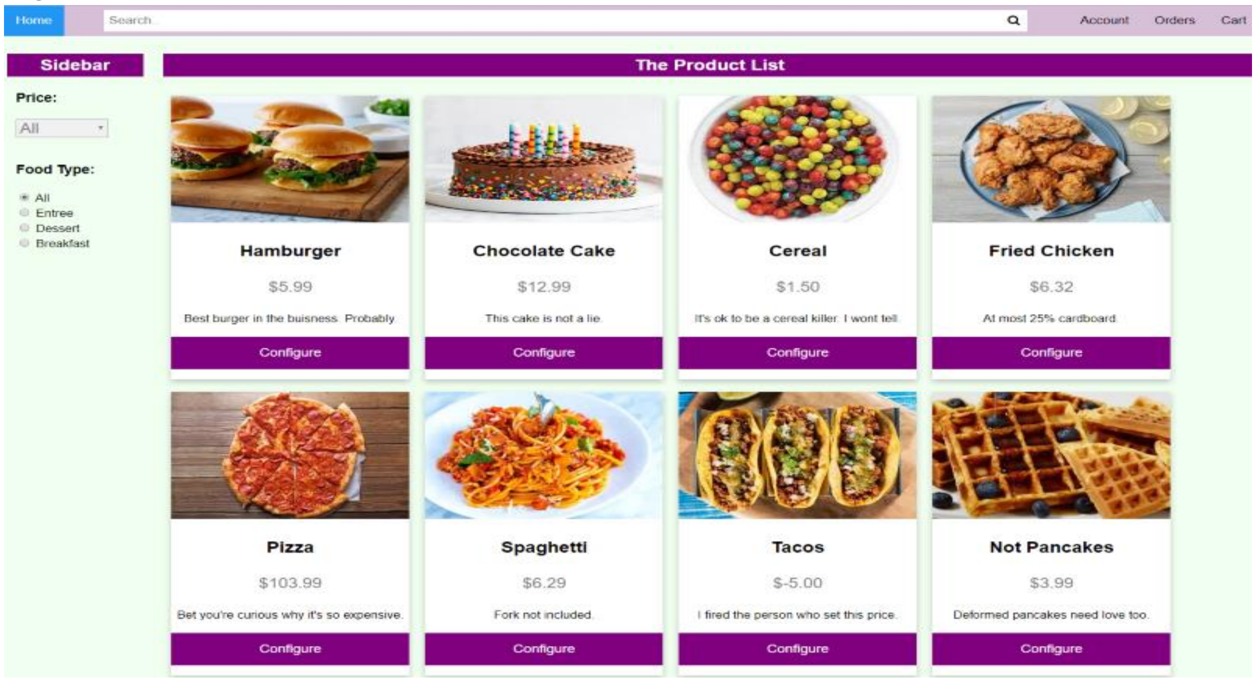
Not Pancakes

You have never quite tasted the wonder that is our signature waffles. Oh did I say waffles, sorry no, waffles are out of vogue, these are Not-Pancakes.

$3.99

2.2.2 Product list page

Figure 1.3 Product List UI

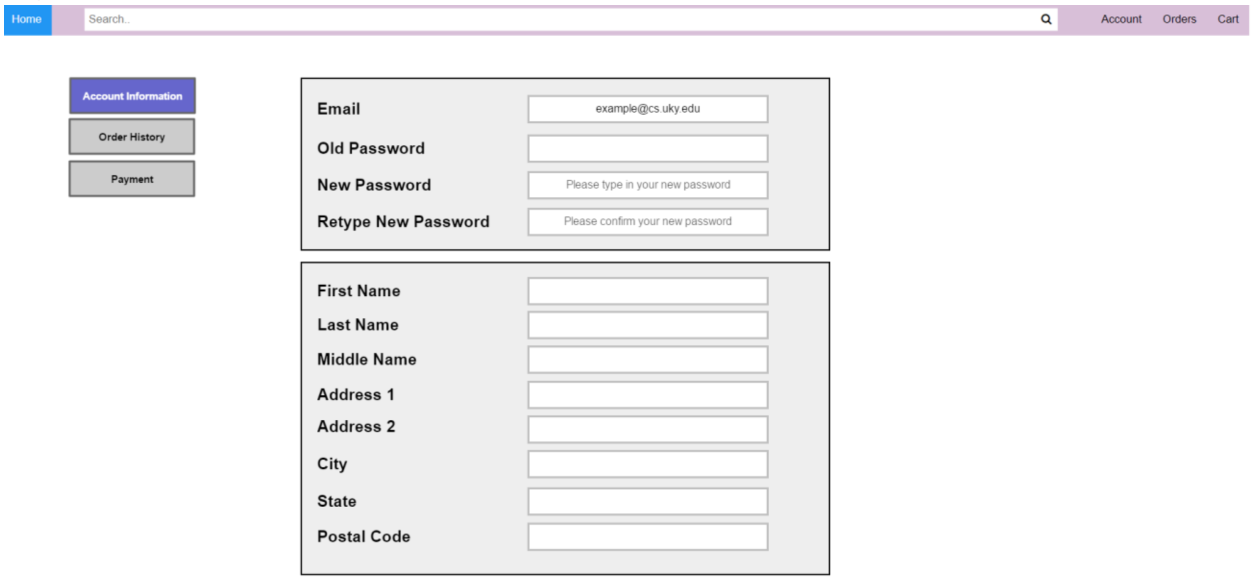


Product	Price	Description
Hamburger	$5.99	Best burger in the buisness. Probably
Chocolate Cake	$12.99	This cake is not a lie.
Cereal	$1.50	It's ok to be a cereal killer. I wont tell.
Fried Chicken	$6.32	At most 25% cardboard.
Pizza	$103.99	Bet you're curious why it's so expensive.
Spaghetti	$6.29	Fork not included.
Tacos	$-5.00	I fired the person who set this price.
Not Pancakes	$3.99	Deformed pancakes need love too.

There are three main regions to the product list page, the header, the sidebar and the product list display region. The header will exist on all standard user facing pages, after a successful login, and contains a button to navigate to the home page, a search bar, a button to navigate to the user's account page, a button to navigate to the user's order history and a button to navigate to the user's shopping cart. This header will adjust to dramatically thinner displays, such as mobile, where element ordering will adjust to be stacked in the display. The sidebar will contain all filtering options. Shown currently are a dropdown box and a radio button selection set, but more elements will be added including slider bars and multi-select checkboxes. The product list display region contains all of the product cards that meet either the search input or any additional filtering imposed by the filter options on the sidebar. Each card is composed of an image of the product, its name, its price, a description and a button to navigate to its product page to be used for making the final purchasing decisions. The size of a card is fixed, however the number of cards displayed on a row of cards dynamically adjusts with the width of the display. This is a screenshot of fully functional element we have developed; however, examples entries are not representative of the final product.

2.2.3 User Account page

Figure 1.4 User Account UI



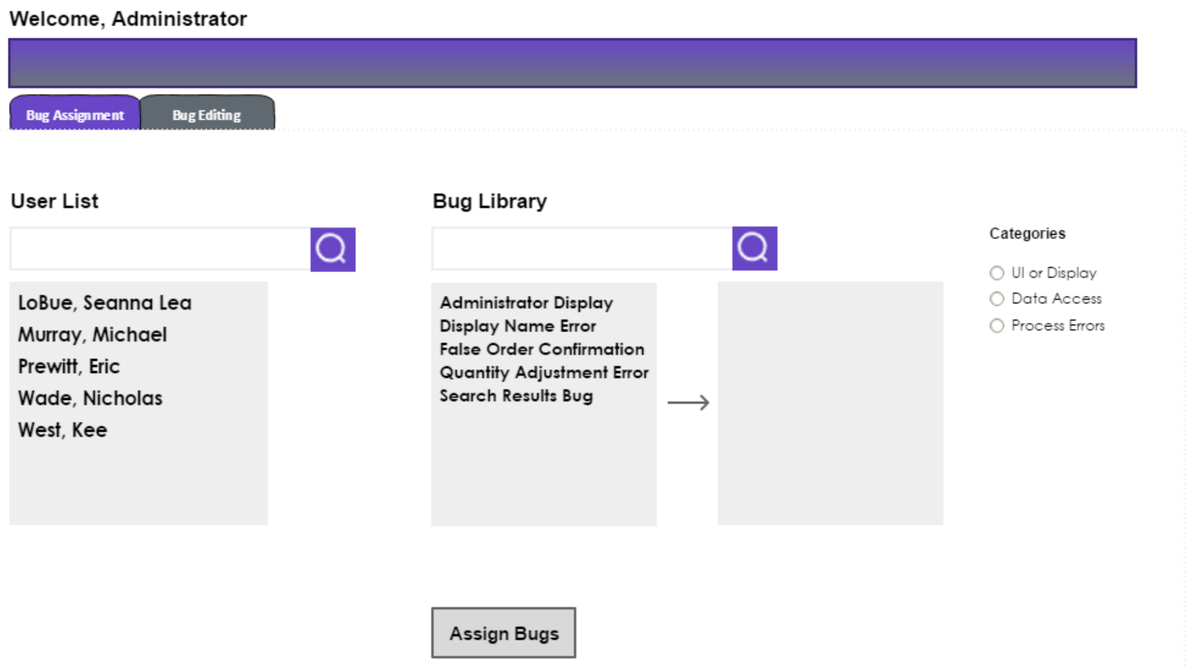
Email	example@cs.uky.edu
Old Password	
New Password	Please type in your new password
Retype New Password	Please confirm your new password

First Name	
Last Name	
Middle Name	
Address 1	
Address 2	
City	
State	
Postal Code	

The user account page shares the header element seen on other pages, allowing continuous access to the home, account, orders and cart button as well as the search box for navigating the user to the product list page and immediately filtering the products listed based on input. Inside of the account page are three different tabs, account information, order history and payment. The account information tab lets the user see the individual attributes that are stored in their account information as well as the ability to change their password. The order history tab will show a sorted list of all orders placed, showing the order id number, the items in the order as well as the total price of the order. The payment tab contains information on payment settings as well as listing any outstanding balances such as from the Net 30 payment method.

2.2.4 Admin page

Figure 1.5 Admin Page UI - Bug Assignment

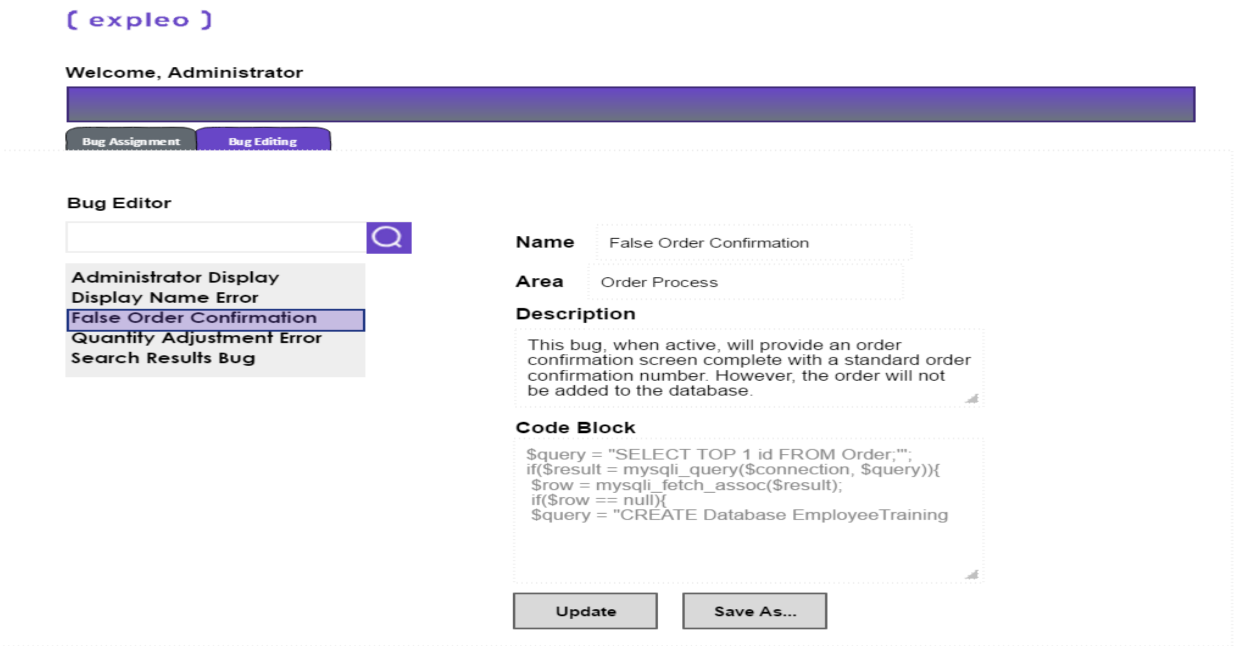


LoBue, Seanna Lea
Murray, Michael
Prewitt, Eric
Wade, Nicholas
West, Kee

Administrator Display
Display Name Error
False Order Confirmation
Quantity Adjustment Error
Search Results Bug

When an administrator logs in, they will be taken to an administrative page with two tabs. The first is for bug assignment, and it will allow the administrator to select a single user by either browsing through a list or by searching in a box. Once the user is selected, the administrator will then search or filter through the list, either double-click the bug or use the arrow to move them to the selection pane, and then assign them to the user.

Figure 1.6 Administrative UI - Bug Editing



Administrator Display
Display Name Error
False Order Confirmation
Quantity Adjustment Error
Search Results Bug

Name: False Order Confirmation

Area: Order Process

Description: This bug, when active, will provide an order confirmation screen complete with a standard order confirmation number. However, the order will not be added to the database.

Code Block:

```
$query = "SELECT TOP 1 id FROM Order;"";
if($result = mysqli_query($connection, $query)){
 $row = mysqli_fetch_assoc($result);
 if($row == null){
 $query = "CREATE Database EmployeeTraining
```

In the bug editing tab, the administrator will be presented with a list of bugs and a blank bug pane on the right. If the administrator prefers, they can create a bug directly in the right pane, or they can search or browse for an existing bug in the list on the left. Once they have clicked on a bug, the bug code and metadata will be displayed in the pane. There the administrator can make edits and either update the existing bug or save it as a new bug.

2.3 Design Pattern

Our project primarily relies on the client-server model as our application is an e-commerce website. The server is primarily in place to perform database queries based upon the inputs of the client. The client side provides a means of interacting with the content displayed and making requests to the server. In addition, for the implementation of bugs, the server controls variations of interaction frame, results and scripts served to the client based on the bug settings applied by an administrator. Thus, elevated clients can manipulate server interactions with other target clients.

3. Testing

Our unit testing will be divided into two main categories. The first exists to ensure that the core website functions as intended, whereas the second category tests whether the bug implementations product the expected results. After looking at each module separately, integration testing will be used to check whether our modules, i.e. pages of our website, interact as anticipated. Many of the functions of our website has to do with how pages interact; therefore, many of the testing concepts are categorized as integration tests, not unit tests. Due to the graphical and web-based nature of our project, our tests will be performed using Selenium on the Chrome browser. This is additional value as the use of our finished product by the client will match this, Selenium on Chrome. Therefore, by enforcing our tests to use these products, we will more naturally discover and correct unforeseen pitfalls for automated testing well before delivery to the customer.

3.1 Core Website Unit Testing

ID	Objective	Description	Expected Result
1	Ensure a new account options allows input.	From the home, click to make a new account.	Information input forms appear.
2	Ensure new account is created.	Input user information into forms and submit, then login.	Login successful and user name shown in the top bar.
3	Ensure previously created account is accessible.	Login with previously created account credentials.	Login successful and user name shown in the top bar
4	Ensure user info can be updated.	Click to edit a field in account information.	Submitted change persists after a page reload.
5	Ensure product filters function as intended.	Click a product filter on a known query. (Repeat for all single filters)	Only specific product should remain.
6	Ensure product search function as intended.	Search for an item. (Repeat for a couple items)	The item card should be displayed.
7	Ensure product configurations function as intended.	Change configuration (size, quantity etc.) on a specific product page.	Price displayed on add to cart should match expected value.
8	Ensure cart status is updated on adding item to cart.	Click to add an item to the cart.	Cart status indicator is incremented by the number of items added.
9	Ensure items can be removed from the cart.	Click to remove an item from the cart.	Item is no longer displayed in cart and the cart total is correct.
10	Ensure checkout page validates input.	Click to submit an order for a populated shopping cart without putting in payment.	Order not accepted. User requested to fill missing information.
11	Ensure all interactable elements have tags.	For each interactable element on the page, inspect for tag.	Each element will have a unique tag. (Manual test)

3.2 Bug Framework Unit Testing

ID	Objective	Description	Expected Result
1	Ensure setting a bug works as intended.	A bug is set to active. (Repeat for all defined bugs)	The bug has the intended impact.
2	Ensure Admin can set a	Admin sets a bug for a user.	User’s experience is impacted in line with bug’s intended

	bug for a user.	That user logs into their account.	impact.
3	Ensure set bug does not impact other users.	Admin sets bug for User 1. User 2 logs into account.	User 2 does not experience any bugs.
4	Ensure that new bugs can be added.	A bug is added to the database and the bug is set to active.	The new bug has the intended impact.
5	Ensure that a bug can be rated.	Set a rating for a bug. Reload the page.	The rating persists through page reload.
6	Ensure that bug history works as intended.	Set a bug to a user and click on the user's bug history.	The set bug is displayed in the user history.

3.3 Integration Testing

ID	Objective	Description	Expected Result
1	Ensure database is connected to landing page.	Enter login information and attempt to login.	Login accepted and transitions to the homepage.
2	Ensure homepage is always accessible.	Click the Home button. (Tested on all pages)	Homepage is navigated to.
3	Ensure account page is always accessible.	Click the Account button. (Tested on all pages)	Account page is navigated to.
4	Ensure cart page is always accessible.	Click the Cart button. (Tested on all pages)	Cart page is navigated to.
5	Ensure orders page is always accessible.	Click the Orders button. (Tested on all pages)	Orders page is navigated to.
6	Ensure account information matches login.	Click account and go to user information.	User information matches login info.
7	Ensure search bar loads product list page.	From a page that is not product list page, search for a product.	Navigates to product list page with proper content for search.
8	Ensure product selection leads to product page.	Click a product on the product list page.	Navigates to correct product page.
9	Ensure added item to cart shows on cart page.	Add an item to the cart and click cart button.	On cart page and the item selected is present.
10	Ensure that checkout button works.	From the cart page, select the checkout button.	Navigates to checkout page.
11	Ensure that placed orders show in order history	Place an order. Click Orders button.	Navigates to order history page and order is present and correct.

4. Quality Review

4.1 Review Checklist

- Ensure a coherent vision between elements written by different group members.
- Check for spelling and grammar mistakes.

- Check that all requested content is present.
- Ensure content is not overly vague and is clear.
- Check that UI designs meet general customer requirements.
- Check that created tests cover core needs.
- Check for consistent formatting.

4.2 Results

1. Initially missing design pattern acknowledgement.
2. First paragraph was of the wrong heading.
3. Two additional bug framework tests were added.
4. Minor grammatical fixes in section 1.
5. Clarity fixes section 2.1.1.
6. Grammar fixes section 2.1.8.
7. Minor clarity fixes section 2.1.9.

5. Metrics

5.1 Estimated Story Points

Based on the combined estimates of all of our team members, we believe that our project is 90 story points. Story points were assigned relative to scoring the product list page as 8 points.

5.2 Actual Lines of Code

We currently have approximately 500 lines of code written. This encompasses the code for building the database, interacting with a web page, searching and filtering elements, extracting items from the database and displaying them on the web page.

5.3 Module Complexity

Cyclomatic complexity:

- buildDatabase: 8
- filterProducts: 5
- All other files produced: 1

None of our modules are explicitly coupled.

Weighted Methods Per Class:

- productList: 6

- buildDatabase: 11
- All other files either 1 or N/A.

5.4 System Complexity

At this stage of the project, many of the elements have their own defined functions that are written in such a way to encourage reusability; however, the use of them currently is limited. Currently the depth of the inheritance tree is at most 2, but with the intended design of our project this will likely increase slightly, but not dramatically over the course of the project. In terms of coupling between objects, some files call other files (php) or constructs classes, but no methods are currently called that belong to other files. However, due to this project being primarily web server based, it does rely on html form submissions and posts, where broken input would cause the element to fail, but none of these technically falls under coupling.

5.5 Product Size

- 15 user story points implemented
- 0 test cases
- 9 code containing files produced
- 17 discrete methods
    - The other 5 code containing files are single path.

5.6 Product effort

Team/Customer Meetings - 20 Hours (4 Hours each)
User Story Preparation - 4 Hours
Project Plan Preparation - 22 Hours
GitHub wiki/Develop Notebook - 5 Hours
Project Coding - 8 Hours
Architecture Assignment - 8 Hours

Total - 67 Hours

5.7 Defects

As we have not made it very far into the development process, defects are minimal, but the ones we have noticed thus far are listed below.

- When the window page size for the product list falls below 630 pixels and header formatting changes to accommodate this display size, the height of the search box is not visually consistent with the other elements.
- On click detection does not handle properly for dropdown boxes on mobile platforms.

- If the user's mySQL installation does not give them sufficient privileges, they are unable to create the new database and requisite tables.

6. GitHub Wiki/Develop Notebook

https://github.com/nawa236/StorefrontTestingApp/wiki

7. Word Count

Seanna Lea LoBue	818
Michael Murray	2354
Eric Prewitt	349
Nicholas Wade	360
Kee West	414