Storefront Testing App

Requirements Specification

Team 4 - Bugs are Features Too

University of Kentucky - CS 499 - Fall 2019

Customer: Ben Fox, Trissential

September 27, 2019

Authors:

Seanna Lea LoBue - Editor, Project overview, Non-functional Requirements, Conclusion, Appendices (ERD, Wireframes and Mapping)

Michael Murray - Editor, Introduction, User Interactions, Functional Requirements

Eric Prewitt - Development and Target Environments, Feasibility

Nicholas Wade - System Model

Kee West - Functional Requirements

Table of Contents

1. Introduction 1
2. Project Overview 1
  2.1 Stakeholder Information 2
  2.2 Problem Description 2
  2.3 System Features 3
  2.4 Constraints 3
3. Development and Target Environments 3
  3.1 Target Environment 3
  3.2 Development Environment 4
4. System Model 4
  4.1 Existing and Proposed Systems 4
  4.2 System Layout 5
5. User Interactions 5
  5.1 Creating an account 5
  5.2 Placing an order (no bugs) 6
  5.3 Creating and executing automated tests 7
  5.4 Assigning bugs to users 7
  5.5 Adding or adjusting bugs 8
6. Functional Requirements 9
  6.1 Priority 1 9
    6.1.1 Storefront Interactions 9
    6.1.2 Toggleable Bugs 10
    6.1.3 User Bug Assignment 10
    6.1.4 Support for automated testing 10
  6.2 Priority 2 10
    6.2.1 Flexible Bug Framework 10
  6.3 Priority 3 11
    6.3.1 Multiple Bugs on one page 11
    6.3.2 Demonstration Mode 11
    6.3.3 Wishlist Page 11
    6.3.4 Checkout Features 11
    6.3.5 Bug Scoring 11

7. Nonfunctional Requirements 12
7.1 Compatibility with Current Server Infrastructure 12
7.2 Optimized for the Chrome Browser 12
7.3 Support for Multiple Users 12
7.4 Ease of Use 12
8. Feasibility 13
8.1 Bare Bones Version 13
8.2 Enhanced Version 13
9. Conclusion 14
10. Appendices 14
10.1 Entity Relationship Diagram 14
10.2 Wireframes and Web Mapping 15
10.3 GitHub Wiki/Develop Notebook 17
10.4 Word Count 17

1. Introduction

Our customer, Trissential, as part of their catalogue of services, provides quality assurance consultancy for their clients. In the pursuit of new clients, it is important to be able to demonstrate the value of your services to prospective customers; however, this can be difficult as copyright and privacy concerns often prevent transparent and thorough demonstrations. In order to address this issue and provide further avenues of value, our customer is interested in the creation of a mock e-commerce storefront that can be used for these demonstrations, with the ability to control the occurrence of intentionally inserted bugs. This ability to control bugs provides a way to dynamically demonstrate the benefits of testing to prospective clients, but is also aimed at a second problem, employee training and proficiency. Traditionally, occurrence of bugs in the real world is unknown, specifically, finding no bugs doesn't mean that none exist. Therefore, by having control over what bugs occur and for whom, this platform can be used to increase employee proficiency and provide metrics for where improvement is needed.

Our project will be to make a storefront website possessing all the core features of an industry standard e-commerce website with the ability to toggle the inclusion of preprogrammed bugs on a per-user basis. Basic users, primarily testing employees, will be able to make an account on the website, view order history, browse a simple list of products, add items to a shopping cart and finalize their order. Storefront administrators will be able to see all user profiles and individually assign them a logical combination of bugs. To facilitate efficient and reliable test automation, it is important that all page elements are well tagged, preventing element ambiguity. Also, normal users must have no way of determining what bugs they have been assigned besides successful completion of their assignment. While we will include as many bug options as possible, the larger goal is to design the bug implementation framework in such a manner to minimize the effort required for others to add new bugs in the future.

In this document, we will further elaborate the scope of the project, describe the functioning environment of our code, review our proposed system design, highlight the needs of the user, including its impact on design, and detail the feasibility of our project. In support of these sections, we have diagrams, located at the end, demonstrating our database structure, website hierarchy and an example product listing page. Altogether, this information is intended primarily to help us, as the developers, align on a singular vision for the needs, goals and implementation of this project. However, by thoroughly defining all of these requirements and design decisions, we intend to provide a way for the customer and potentially future groups, working on iterations of this project, to better understand our priorities, methodologies and solutions.

2. Project Overview

Trissential does not currently have a robust testing environment in place that mimics the e-commerce environment. This makes it more time consuming for test engineers to develop the specific automated testing skills they need that are peculiar to the ecommerce environment. Additionally, the sales team will benefit by having a tool in place, which will allow them to demonstrate the value of the Trissential automated testing product as well as to illustrate the subtle nature of some bugs that may impact customer experience.

In this section, we will expand upon the scope of the problem, reviewing the current system, and demonstrate the value of this custom programming solution over any existing market products.

2.1 Stakeholder Information

Ben Fox, Associate Technical Engineer, Trissential
Quality Assurance Team, Trissential

The primary customer group is the Trissential automated testing team, led by the Testing Manager. The Testing Manager will be the primary user of the administration and bug development system, while their team will be the primary user of the resulting e-commerce site output. The sales team will be the secondary user, using a stored demo profile to access a previously solved bug set and any resulting scripts that locate existing bugs in the demo version.

2.2 Problem Description

Trissential is an information technology services company specializing in providing “continuous quality” services via a suite of testing products, including intelligent test automation, unit and code integration testing, and mobility and compatibility testing. They are trying to improve their employee training and evaluation for their intelligent test automation employees. Flaws with previous projects have resulted in a bug development and deployment tool with limited usefulness. Additionally, previous projects have focused primarily on static websites, and not included the peculiar problems inherent in ecommerce websites. As such, there is no current solution in place.

Our proposed solution is to create a robust ecommerce website with features such as item search, category selection and filtration, product options such as size and color, order placement and account services. Our goal is to build a back end that allows an administrator to select from a bug library, allowing them to assign one or more bugs to an individual to test as a bug package. Administrators will also need to be able to load a bug, modify it to create a different error and then save it to the bug database as a new bug available for inclusion in future tests.

While there are potentially many production websites with bugs, there is no third-party website that allows the user to encounter a random assortment of bugs. It is not feasible for the management team to research websites to find one poorly written enough to justify using as an unwitting testing platform. Additionally, even if such a website was located, it does not provide a dynamic environment for testing where one employee is not encountering the same bugs as their neighbor. The needed solution is not something that is available on the market, so Trissential needs an in-house solution.

2.3 System Features

Our proposed system will involve a web page where an administrator can see a list of users, select a user, view their bug history, and assign them either a saved list of bugs or select new bugs for them to test against. The administrator will also be able to search a list of available bugs, load a bug into an editing screen, update the contents of the bug, and save the bug to the database for inclusion in later tests.

The web page will allow users to either create an account with email verification or log in. Upon logging in, they will receive a custom version of the website with the different bugs loaded in. From this point, the user will be able to interact with the web page either as a normal user or with an automated test script in order to locate the bugs.

2.4 Constraints

This project has as its major deliverable a website creation system that must run via a combination of web server and database server. Trissential has both Linux and Windows web servers, running Apache. They have not applied a limitation on the database server, but given industry standards it is reasonable to use MySQL for this project.

The Trissential team does most of their automated testing via Chrome and Firefox, and therefore our website product needs to work in these environments. To properly interface with their testing scripts, we need to use unique ids for each web element, such as numbered ids in a search results page.

The bug framework needs to be flexible enough to allow bug search and assignment. For the purpose of this project, no bug should prevent the web page from rendering. Bug code should be readable from the Admin page in order to review or modify existing bugs.

3. Development and Target Environments

An extremely important consideration in any software design project is the target environment, as if you are unaware of the customer’s target environment your development could be completely functional but completely nonfunctional for the customer. Once a target environment is identified, a development environment should be chosen that will successfully create a functional program for the target environment. In this section we look at the target environment and go over the decisions made on the development environment with that target environment in mind.

3.1 Target Environment

As this project is entirely within a web site, but a site specifically developed for bug testing application, the target environment for our project is a little different than most. Being a website, one of the first hurdles is to determine how it is to be hosted. In our case, the customer has both Linux and Windows web servers available, with no real preference. Additionally, there were few constraints set on how the website was to be programmed, which leaves most common web programming languages available.

There is also a second aspect to our target environment, the user of the website. In this case, the customer does the vast majority of their bug testing in a Windows environment using the Chrome web browser. While it is desired that the site be displayable with multiple screen configurations and sizes, a high level of cross compatibility and mobile compatibility is not required.

3.2 Development Environment

With the customer's requirements and the target environment as described above, we were left with many options to choose from in regards to our development environment. For web development, we've decided to go with PHP as the scripting language for our HTML pages. PHP is extremely commonly used in web development and widely supported by many devices and browsers, so it seems to be a great fit for our purposes. For the web server side of things, we've decided to go with Apache web servers due to its prolific nature as well as the ease of access from a development perspective. As our website will have a large integration with database systems, we also needed to choose a database system as well. We've chosen MySQL, which is readily available and something that several of us have experience with in the past due to our classwork.

4. System Model

This section will clarify our intent for the system model by detailing the layout of our proposed system, commenting on any foundational elements. We start with an explanation of the origin point of this project and lack of knowledge of any previous tangential systems. Following the explanation, there is a description of the system we have proposed and will be developing for Trissential, which is supported by figures to help reinforce our intentions for this project.

4.1 Existing and Proposed Systems

The existing system that Trissential has been using for their artificial webpage bug loading system was not disclosed to us because they wished to see what sort of solution we would come up with because previous iterations of the project have gotten bogged down with issues were the team spends most of the time figuring out what the previous teams had done before they could expand on that system for their project because of poor documentation and commenting of the code. Therefore, we have been tasked with starting from a blank slate without being influenced by the previous version. We are proposing is a bug library were each section of each webpage will have a selection of bugs that can be added for a selected user. This bug library will have the ability to have bugs added to it and previously created bugs edited by an admin user. The bugs selected for a particular user will show only for that user when they log in and the website should otherwise operate as a fully functional ecommerce storefront minus actual payment processing.

4.2 System Layout

Figure 10.1 shows the layout of our database relative to the eCommerce site. Products that are being sold on the site will be stored in one part of the database with information on the product's name, price, inventory levels, categories, color, and size to be used as searchable and filterable information that can be used by users and potentially be used as places to put bugs that can be tested for. The other half of the database will contain user accounts that have information on the users' name, address, and email along with an order history that will double as a shopping cart where the order is unprocessed. The user account will also be connected to a bug assignment table were the system will reference what bugs are supposed to be active for the user which can be set by an administrator. ***Figure 10.2a*** shows the planned map of the website. Users will first have to register on the registration page, then afterwards they will be able to log in to their account. The account page will have access to the user's order history page and the order history page will have access to the order details page. The main landing page will allow users to search for products and access each products product page. Both the product pages and the landing page will have access to the users' cart and from the cart they can move to the checkout page and enter their payment information. From there the users will move to the confirmation page to tell them if the transaction was successful. Finally, admin users will have access to the admin page where they can assign bugs to each users' profile and also access a bug editor were, they can change and add new bugs that can be applied to users. ***Figure 10.2b*** is an example mockup of the search page and the basic design of the website.

5. User Interactions

In this section, we will describe the expected interactions of the applicable user with each of the pillars of this project, by way of use case scenarios. These pillars cover the functionality of the Storefront in an error-free state, the interactions with the Storefront for the purposes of testing, the administrator interactions for controlling user bug assignments, and developer interactions for adding or adjusting bugs. Below, each of these will be expanded on further, providing a detailed account of the intended interactions from the point of view of the user. As a small note, these are written from the perspective of the ideal version of the product; therefore, some elements described may be exclusively part of the enhanced version that incorporates all desired features.

5.1 Creating an account

Primary Actor: Storefront User (Testing Employee)
Scope: Storefront use
Level: User Goal
Precondition: User has navigated to the landing page of the Storefront.
Postcondition: The user can now login to their own personal account.
Flow:

1) The system presents the user with the option to either login to an existing account or to create a new account.
2) Upon choosing to create a new account, the user is presented with an entry field for inputting desired login information as well as very basic personal information.
    a) User selects instead to login to an existing account.
    b) System prompts user to input their login information.
    c) User selects to return to creating an account.
        i) User chooses to attempt to login without valid credentials.
        ii) System prompts user to create an account.
        iii) User selects to returns to creating an account.
3) User inputs their desired login and account information.
4) System sends an authentication message to the input email address.
5) User navigates to their email and clicks the link in the email.
6) System authorizes the account so that the user may now use the login option.

5.2 Placing an order (no bugs)

Primary Actor: Storefront User (Testing Employee)
Scope: Storefront use
Level: User Goal
Precondition: User has navigated to the landing page of the Storefront.
Postcondition: The user has completed an order, logging the transaction on the server log, viewable as well in the user's order history.
Flow:

1) User selects to login to their account and inputs their login credentials.
2) System redirect user to their personalized Storefront.
3) User scrolls though a listing of items available.
4) User uses filters and search criteria to help locate items of interest.
5) System filters or orders display of item based on user input.
6) User clicks on the item they want.
7) System redirects them to the product page of the item, which contains additional information and selection options.
8) User selects any additional options that may apply to an item.
9) System updates the page with relevant changes based on selection.
10) User chooses to add this item to the shopping cart.
    a) User chooses to leave this page without adding the item.
    b) Steps 3-9 are repeated until step 10 is chosen.
11) System lets the user know the item has been added and allows the user to finalize their order (checkout) at any time.
12) User returns to the listing page and steps 3 - 11 are repeated until the user decides to finalize their order.
    a) User clicks to remove an item from their cart.
    b) System removes the selected item from the user's cart.
13) User selects to finalize their order.

14) System redirects the user to the checkout page, displaying the items that are part of the order and the options available for payment.
15) User selects payment method and clicks to submit the order.
    a) User chooses to return to item listing without submitting the order
    b) Steps 3 - 14 are repeated until step 15 is chosen.
16) System saves the order information, updating the relevant database items to properly track the event.
17) User logs out or leaves the Storefront.
    a) User clicks to view their order history.
    b) System displays user's orders and overview details, sorted so that most recent orders are displayed first.

5.3 Creating and executing automated tests

Primary Actor: Testing Employee
Scope: Storefront testing
Level: User Goal
Precondition: Tester has navigated to the landing page of the Storefront and logged into their personal account.
Postcondition: The tester is able to execute their fully formed automated testing script on the Storefront, without the need to compromise their design intentions.
Flow:

1) Tester views the page source to see unique tags and id of page elements.
2) Tester writes and runs a basic testing script to ensure that feature selection works as expected.
3) System responds to the script in the same manner as expected if a human had performed the same interactions with the Storefront.
4) Tester goes through user interaction steps listed above in Use Case 2, writing tests to evaluate whether the system responds in the same way detailed in Use Case 2.
5) Tester executes their tests on the Storefront.
6) System responds to tests in line with the unique combination of bugs that have been enabled for this tester, only producing unexpected results in the event that a test case encompasses an intentionally set bug.
7) Tester repeats this process of creating tests and executing them until they are satisfied that they have attained sufficient coverage.

5.4 Assigning bugs to users

Primary Actor: Storefront Administrator
Scope: Storefront testing
Level: User Goal
Precondition: Administrator has navigated to the landing page of the Storefront and logged into their account.

Postcondition: The administrator has successfully assigned the desired bugs to the intended testing employee so that future use of the Storefront by testers will reflect the changes made.
Flow:

1) System redirects admin to standard functioning Storefront.
2) Admin clicks on the account options dropdown and selects Admin page.
3) System redirects admin to the admin page containing a user list, with brief information.
4) Admin clicks an individual user.
5) System redirects admin to a user details page with bug assignment options.
6) Admin selects the bug assignment option.
7) System redirect admin to bug assignment page.
8) Admin manually selects bugs from a list of options.
    a) Admin selects an option to randomly assign the user a collection of bugs.
9) Admin confirms assignment.
    a) Admin rejects assignment.
    b) System takes admin back to the user details page where steps 6-8 can be repeated until step 9 is chosen.
10) System updates the bugs associated with that user.

5.5 Adding or adjusting bugs

Primary Actor: Future developer
Scope: Storefront modification
Level: User Goal
Precondition: Developer has accessed the system backend.
Postcondition: Changes made are accessible to system users and administrators.
Note: This framework is not yet cemented, therefore flow will not necessarily be a good reflection of the end product. Instead, it exists to help guide our design intentions.
Flow:

1) Developer brings up the SQL database associated with tracking and detailing the implemented bugs.
2) Developer inserts a new item into the database containing the attributes for bug name, bug category, bug description, the impacted function id and when applicable the desired modified function return that will produce the intended bug.
    a) Developer also adds a new function to the source code.
    b) System knows the impacted function id because of developer entry so minimal work is needed to determine which version of the function is run depending on bug enable status.
3) System will now contain this new information and thus when future queries are made by the system such as when the admin is assigning bugs to testing employees, this new bug will be part of the list.

6. Functional Requirements

In this section, we will describe in depth the functional requirements laid out by the customer. This will ensure that designs proposed will have considered all of the customer's needs and preferences and that testers can conclusively verify that these requirements have been met. Below, we will walk through the various requirements for this project, ordered by priority, possessing a name, priority level and a detailed description of what is entailed. When necessary, further comments will be made to appropriately ground what is entailed by the listed requirement. As a brief overview, the core needs consist of a functioning mock e-commerce Storefront and the ability for an administrator to control the introduction and distribution of bugs for individual users. All of which needs to be flexible enough so that introduction of new bugs, at a later date, is not prohibitively difficult or expensive. The remaining requirements exist as enhancements, aimed at assisting our customer in using this application for new client demonstrations, employee training and prospective employee evaluation.

6.1 Priority 1

6.1.1 Storefront Interactions

The Storefront must be a website containing a landing/login page, account page, order history page, product list page, individual product page, shopping cart page and a checkout page. The landing/login page needs to have options for both creating a new account and logging into a previously created account. The account page must be accessible at any time after a successful login and must display the information collected during account creation; however, the depth of information collected is not specified. The account page must contain a mechanism to navigate to the order history page and will also show any outstanding bills, this will conditionally appear in the event of using the Net 30 invoicing option for order checkout. The order history page will display a sorted list of all purchases made by the user. Order ID, items purchased and order total are required in the order history; however, single page display or a breakout page (click on order ID to see more information) is not specified. Product list page must contain a list of products, each with a unique image and product name, where clicking either will navigate to the product page. Product List page must have the ability to sort the items on display as well as filter by basic identifying criteria (Group A, B, C, etc. is sufficient). Product page must contain basic product information as well as the ability to make sub-selections, such as shirt size, when relevant. Product page must have the ability to add the item to a shopping cart. Shopping cart must be accessible at all times after a successful login and must show products that have been added to the cart but have not been purchased yet. Users must have the ability to remove items from their shopping carts. Shopping cart page must have the ability to navigate to the checkout page. The checkout page must prompt the user for a payment type; however, no specific payment types are defined, but a variety of options are required nonetheless. If time permits, more aspects of a typical checkout page will be implemented including storage of additional user contact information to allow a greater range of bug variations.

6.1.2 Toggleable Bugs

The main feature of our website will be the ability for the Admin account to selectively toggle bugs that are then instantiated for individual user accounts. These bugs will be implemented so that the skills and ability of prospective employees to meet the requirements/needs of the company can be tested, to train and evaluate current employees and to perform demonstrations for potential new clients. The bugs must be implemented in a way that does not sabotage the core functionality of the website in general, but effectively allows the user to be tested in the detection of certain inhibiting bugs. The Admin must have the ability to toggle specific bugs for specific user accounts at will. There must be applicable bugs to every aspect of the core website. This includes the homepage, product pages, product selection filters, product images, misdirecting links, false submissions orders, confirmation number generation and account information. The Admin must have the ability to create and load new bugs into the database to allow for this software to evolve with the needs of the company.

6.1.3 User Bug Assignment

User Bug Assignment will be implemented through the creation of a primary Administrator account with the ability to control the version of the website and it's features the individual users will interact with. The Administrator page will be separate from the users' and will be accessed through its own login. The Admin must have the ability to pre-select hindering bugs from a collection of various types and deploy them into individual user web pages. The Administrator account must have the ability to create and load new bugs into the database and switch the effectuation at will. These features must only be accessible to Administrator accounts.

6.1.4 Support for automated testing

All page elements must be uniquely identifiable by their tag. This unique tagging system will allow all elements to be identified and interacted with during automated testing by testing employees.

6.2 Priority 2

6.2.1 Flexible Bug Framework

The addition of new bugs must not be prohibitively difficult or time consuming. A Flexible Bug Framework will be designed to allow the Admin an easier way to effectively create and load new bugs into the database and change elements of the website. Through the implementation of a GUI with icons/menus to carry out intended commands and changes, we can simplify the process for the Admin, as well as, allow interactions with the website to be as straightforward as possible. This feature will also set the groundwork for many desired enhancements planned for the website, allowing simpler implementation for our group, if time permits, or future teams.

6.3 Priority 3

6.3.1 Multiple Bugs on one page

While not mandatory for our project, thus given a lower priority, the customer would prefer to have the ability to assign more than one bug to be present per page. As such, in the event of conflicts existing between bugs, it must not be possible for the Administrator to assign bugs whose impact is mutually exclusive or would otherwise cause critical loss of function. This feature will allow for greater value in regard to variation and situational robustness for training, testing and client demonstrations.

6.3.2 Demonstration Mode

Another desired feature would be the ability of the Administrator account to present comparative versions of the website to enhance presentations for potential new clients. Therefore, if such a feature were implemented, the Administrator would need to be able to define a bug set and immediately open a new tab to a Storefront configured with these settings. While it would be possible to just two users log into their accounts and demonstrate the difference between them, the desire is to accomplish this seamlessly.

6.3.3 Wishlist Page

One of the planned features for the user accounts is a Wishlist. This feature must be easy to access for the user and provide them with the option of adding or removing items from their Wishlist after successful login. This feature must have some basic sorting and filtering options and apply solely to the related user account. Items in the Wishlist must be able to be added to the shopping cart at any time. This page must be part of the possible pages impacted by Administrator assigned bugs.

6.3.4 Checkout Features

The user will be given the option to pay for their order in various forms, including among others, credit card, debit card and Net 30. The Net 30 option must be implemented in a way that adds a bill to the individual user account and keeps track of the payments over the course of the 30 days. As well as payments, the checkout page must include the option of choosing a desired shipping speed. These features must be easily accessible to the user and allow them to choose the payment and shipping options before finalizing the order. After order submission, the user must be shown a summary of the order including all items purchased, order total, payment method and shipping speed. Following this, the user must be given the ability to navigate forward to the product listing page as well as the other persistent options, detailed in Storefront Interactions, such as the user account page.

6.3.5 Bug Scoring

This feature must allow the Admin the ability to assign scores to various bugs during employee testing review. The admin must be able to review these scores when assigning future bugs sets, allowing for a more personalized and efficient user testing environment, especially in the case of multiple bug implementations. This feature must be easily accessible to the Admin account and allow them to accurately make changes as needed. Ideally, these scores should be unique to individual users, where the difficulty assigned to a bug is a composite of the experiences of all users who have been previously assigned this bug.

7. Nonfunctional Requirements

For this project, our team in conjunction with Trissential have identified four core items that will be necessary to the success of this project. Most of these items relate at least tangentially to the user experience. However, compatibility with existing server infrastructure will go a long way to make Trissential's deployment smoother.

7.1 Compatibility with Current Server Infrastructure

Trissential has access to both database servers as well as Linux and Windows web servers. Our customer has indicated that their IT team can implement other architectures if required, but in order to minimize difficulties in deployment we plan to work around their existing architecture.

7.2 Optimized for the Chrome Browser

The Trissential team do most of their automated testing using Selenium with the Chrome Driver. Therefore, we need to make sure that the bug framework and ecommerce site render properly within the Chrome browser. If there is time, the Trissential team does do some testing in Firefox as well, so any additional browser optimization would be for Firefox.

They do not intend to use this product for mobile testing. Therefore, while using a responsive framework such as Bootstrap will be helpful, it is not necessary to provide a design fully compatible with a mobile device.

7.3 Support for Multiple Users

Trissential does not intend to use this for hundreds of users at a single time, but there will be periods of time when multiple users will be logged in and receiving their custom bug websites for testing. During that time, each website needs to be rendered without impacting the responsiveness or quality of the other sites. Each page or pages should have a small enough footprint to be quickly generated and displayed to the user.

7.4 Ease of Use

The administrator needs to have a page that allows them to quickly select a user, and then assign them a list of bugs. In expectation that the list of bugs will grow and become more robust over time, the administrator needs to be able to search for bugs. The bug development UI needs to make every effort to minimize the amount of raw coding an administrator needs to do by showing existing code and how the code is rendered side by side. Other tools will be utilized to maximize the administrator utilities.

8. Feasibility

When considering taking on a project with a limited time frame such as we have, it is important to determine the feasibility and have multiple versions of the ultimate design outlined. This allows us a much greater chance of success overall rather than attempting to add all desired features from the beginning, regardless of worth. An important part of this consideration is the needs of the customer, specifically what features of the program are a requirement versus features that would be considered “bonuses”, if possible, in our given time frame. Below, we have outlined two different versions of our design: one which solely has the absolutely required functionality for the customer’s needs, and one which has all of the reasonably desired functionality that the customer desires.

8.1 Bare Bones Version

The minimum requirement for our system is to have a web store with a bug implementation feature where an admin can toggle bugs that will occur in the web store on or off for specific users. This requires a basic user account management with a login page, account creation page, user account page with history and etc. It will also require a completely functional web store with product list page, individual item pages, a shopping cart, and a checkout page. The product list page needs the ability to filter and order using traditional means i.e. dropdown boxes for sorting. The item page must include a dummy image, a “cost”, a quantity, and the ability to add items to the cart. The user must then be able to go to the checkout page and submit their order, which will send them to an “Order Received” page and put the order into a database.

The main purpose of the project is to implement bugs into this web store, so a big portion of the functionality revolves around that. The admin must have an admin page which lets him see all user accounts and select bugs to apply to their accounts. Then, when a normal user logs in and goes to a page with one of the assigned bugs, they will receive a bugged version of the page rather than the correct version of the page. The bugs can range from simply incorrectly displayed items, such as a broken image or missing text, all the way to something such as the “Order Received” page coming up but the order not actually being input into the order database.

8.2 Enhanced Version

The enhanced version of the website would include all of the features described in the bare bones version, with several additions ranging from small to relatively large. The most straightforward being an expansion to the number of testable elements per page. This opens up the possibility for more bugs and unique page element types to be tested. Along those lines, we need to have bugs that impact a majority of real-world use cases, but with an additional time, we can implement more complex or less customer critical bugs. More focused, pertaining to the admin page, we would like to allow the admin to add and modify bugs straight from a GUI on the page. Another change to the admin page would be the ability to randomly select bugs given to the user, as well as the option to group bugs together.

Another potential feature would be the addition of a Wishlist, which would effectively be the same as a shopping cart with more permanency and would also allow the user to share their Wishlist with other users. Another feature discussed would be the ability to allow a user's orders to go on a "tab" akin to a Net 30 system and then allow them to "pay" for that tab on the website as well. While certainly very useful, these features included in the enhanced version are not necessary to fulfill the customer requirements so they will be effectively bonus objectives, serving to enhance the quality of life, value and overall appeal of our product. As such, they will not be implemented unless time allows as we near the end of our project's schedule.

9. Conclusion

A custom bug deployment platform will go a long way to improving the expertise of Trissential's automated testing engineers, by providing them with a varied number of bugs to test for in a closed environment. By reviewing the team's automated scripts, the management team can determine whether any employees need additional training in located more challenging bugs or if someone has exceeded all expectations.

While employees can get this experience by writing automated test scripts for customer websites, it will be more efficient to provide them that experience as part of their overall training. This tool will allow Trissential to produce a robust, expert team in the shortest time possible.

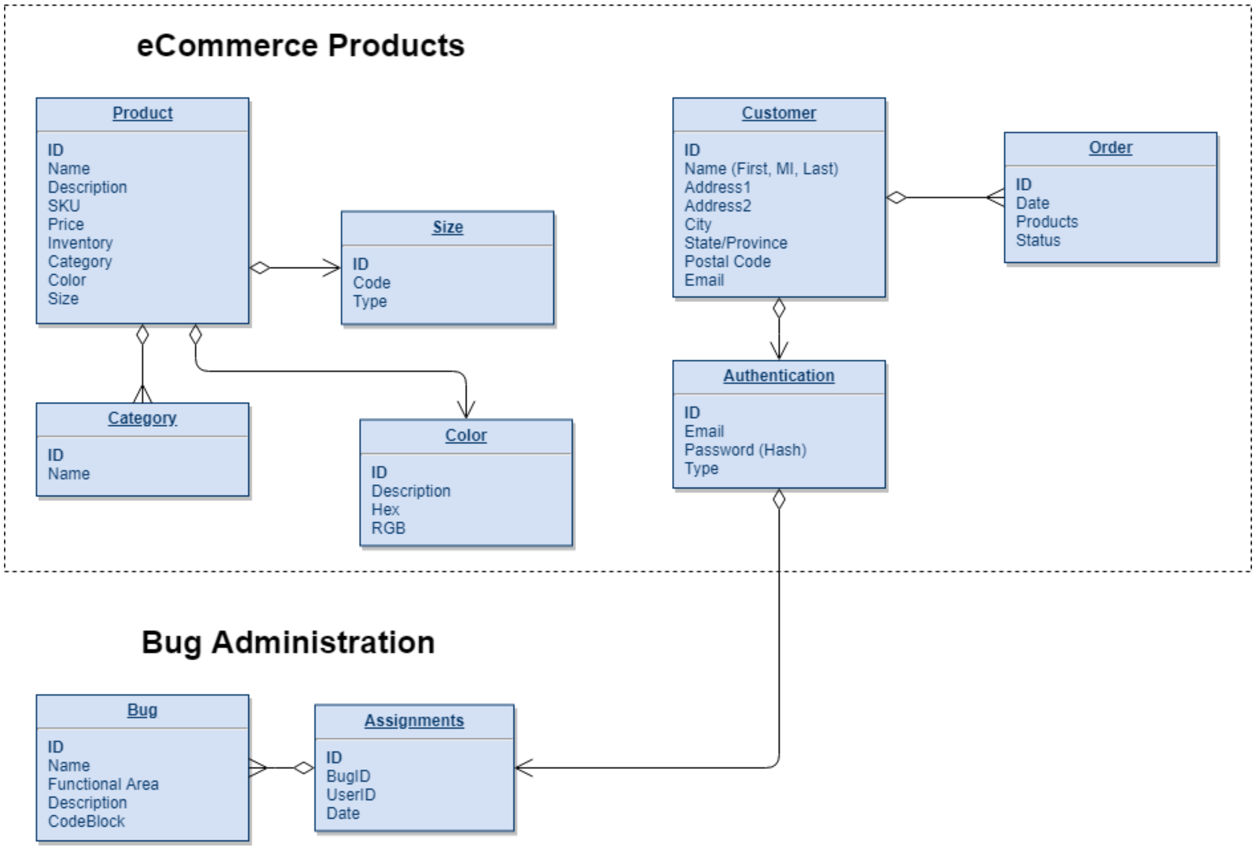
10. Appendices

10.1 Entity Relationship Diagram

The bug framework product will rely heavily on creating database-driven websites. Therefore, our database needs a number of product-related entities to allow users to search for products, make custom selections, and to add or remove items from inventory. Similar entities are needed for the customer interactions, including orders, and account information. The goal is to provide a robust database capable of handling many of the different interactions needed for a shopping website.

The only overlap between the eCommerce entities and the bug entities is the use of the customer authentication table in order to connect the account to the assigned bug list upon login.

Figure 10.1 Relationship between eCommerce Platform and Bug Framework



10.2 Wireframes and Web Mapping

Given that this project requires a functioning website for both testing and administrative functions, it is important to give some consideration of what that product will look like and what options will be available. Looking at Figure 10.2a, from the Registration and Login pages, the user will be taken to a different page depending on whether they are an administrator or a user. The ecommerce site will have multiple avenues to navigate to the same information, which will better mimic the standard user experience at a web store. Depending on the bugs loaded, the initial page for the user may vary. Some of these navigation paths are displayed in Figure 10.2b. In this sample wireframe, you can see that navigation will include search and breadcrumb.

The administrator will be able to see a list of users and bugs and assign them to build a bug package. They will have a page or tab where they can select a bug, load it into an editor, and create a new bug for the database. Depending on time, bugs may also be created from scratch with a blank editor.

Figure 10.2a Website Map

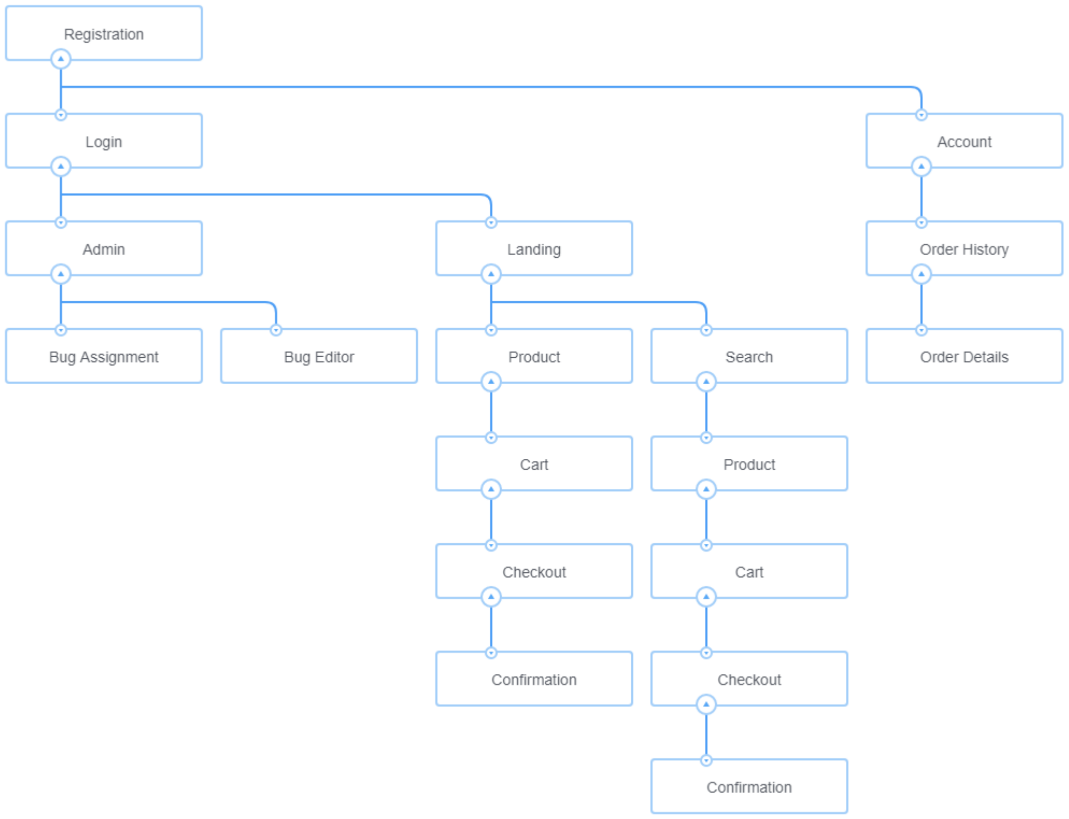
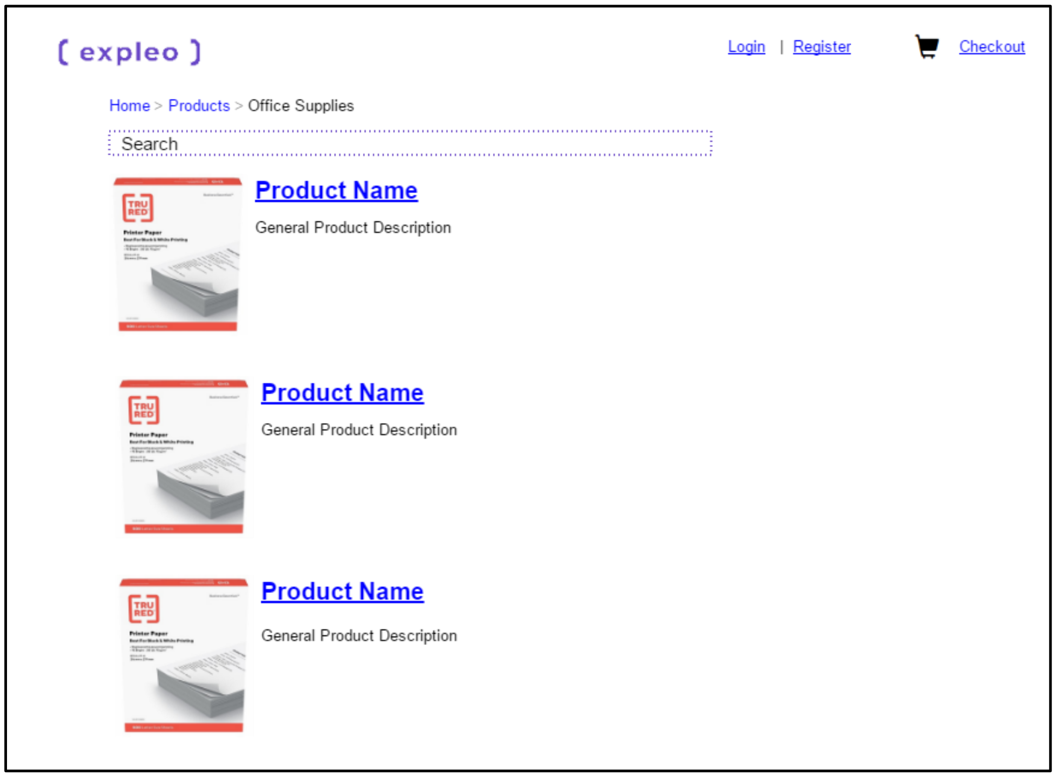


Figure 10.2b Sample Store Page Wireframe


(expleo)
Login | Register
Checkout
Home > Products > Office Supplies
Search
Product Name
General Product Description
Product Name
General Product Description
Product Name
General Product Description

10.3 GitHub Wiki/Develop Notebook

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

10.4 Word Count

Seanna Lea LoBue - 1,617

Michael Murray - 2,255

Eric Prewitt - 1,017

Nicholas Wade - 552

Kee West - 1,020