Online Ordering Platform

DESIGN DOCUMENT

Team Number: SDMAY19-22
Client: Doll Distributing
Adviser: Mohamed Selim
Alec Harrison - DevOps
Caleb Olson - UI/Database
Daniel O'Neill - Testing Engineer
Mitchell Bennett - Communications
Thomas Staudt - UI/Testing
Thomas Wesolowski (TJ) - Project Manager
Team Email: sdmay19-22@iastate.edu
Team Website: https://sdmay19-22.sd.ece.iastate.edu/
GitLab: https://git.ece.iastate.edu/sd/sdmay19-22
Revised: 12/2/2018/Version2

Table of Contents

List of Figures

List of Tables

List of Definitions

VIP: Vermont Information Processing

CSV: Comma Separated Values

AWS: Amazon Web Services

SQL: Structured Query Language

SES: Simple Email Service

Penetration Testing (Pentesting): An application security testing technique in which a tester attempts to gain access to secure data.

Cron: A UNIX process scheduling utility.

DNS: Domain Name Server

Agile: Development methodology focused on incremental implementation of small tasks over short periods of time.

PHP: PHP Hypertext Preprocessor

HTML: Hypertext Markup Language

CSS: Cascading Style Sheets

BrandFinder: Part of the Doll website showing recent retailer purchases

1 Introduction

1.1 Acknowledgement

Team 22's Client: Doll Distributing LLC

Team 22's Advisor: Mohamed Selim

1.2 Problem and Project Statement

Currently Doll Distributing has an ordering system that requires sales reps to be in person to make sales. This can cause their clients to feel pressured into buying other products or being inconvenienced to place orders at not optional times. There is also not a way for clients to learn about new products or deals on their own time. They currently must call Doll's rep and hear about deals if their schedules work out.

The task is to create an ecommerce system for Doll Distributing that will interact with their current inventory system. A system will be created that will allow clients to view Doll's inventory, place orders that will be sent to account reps, and be able to view their historical orders. This will be linked from their current site and not allow anyone without an authorized account with a valid liquor license to enter.

1.3 Operational Environment

The operating environment of this project will be an AWS Linux server hosting a docker container running a Laravel [3] web app. This application will also have to have a mySQL database with a user's table to host users and their liquor licenses. This system should be able to run in any environment a smartphone can connect to the internet and a climate it can run.

Order Placing- Launching a php job to send an email via SES to account reps, associating users and orders in the database so users can see past orders.

User Interface – Linux Operating System, Route 53 route to hit to bring up web app, User Browser (like chrome, edge, or firefox)

1.4 Intended Users and Uses

The project with Doll Distributing is a unique system that required some research to come up with intended users for the app. This app will be designed with the core users in mind and also work well with other users, too. The users have been narrowed down to clients, account reps, IT, and sales.

This project has four main users with varying levels of needs that need to be addressed by the app. For the IT users the app must be maintainable and capable of working with their current inventory system as close to real time as possible. Another role is the clients, or the people who buy the

alcohol from Doll. This user is the primary use case and their bare minimum needs of placing an order and seeing their past orders must be satisfied. Next, the account rep user was identified. They would need to be able to authorize users to be active once their liquor licenses are validated. The last user identified would be the sales people. This system needs to be visually appealing enough that the sale department should be able to pitch this system as enough of a pro to bring clients to Doll from their competitors.

1.5 Assumptions and Limitations

Assumptions:

- Ability to interact with VIP's inventory system better and quicker than the 20-minute manual CSV export.
- VIP to give the team access to an api or a testing database.
- The end product will be used in the United States.
- The website will be written in English.
- Able to achieve goals using a MySQL database.
- Able to extend Doll's current website style to the new system.
- An email to an account representative when an order is placed will be the end goal of the product.
- The app will be maintained in the future (so it requires clean and well tested code).

Limitations:

- The user shall be validated before they enter the site (client requirement).
- The app shall use VIP's inventory data (client requirement).
- The app shall work on desktop and popular smartphones.
- Currently, the fastest time that a CSV exports from VIP is twenty minutes.
- The website will not allow people without liquor licenses to enter.
- The site will be dependent on Amazon Web Services.

1.6 Expected End Product and Deliverables

This section will go over the project's deliverables and the times at which they are expected to be available. This project will be divided into several major milestones. Namely the UI mockups, a working prototype, a testable ordering system, and the final Ecommerce platform. Documentation will be included with each of these deliverables to help Doll as well as the users be well informed about usage of each of the product's phases.

- UI mockup
    - Diagrams that display the UI in enough detail for all stakeholders to visualize it properly
    - Will have HTML and CSS files so that UI can be demonstrated in a browser

    - Must be properly branded with Doll's colors and logo
- Working Prototype
    - Users will be able to log in using a registered account and have a valid liquor license.
    - Users will be able to see the inventory.
    - Users will be able to place an order request.
    - Will be testable by select clients of Doll's
- Final Platform
    - The final product will be a refined version of the original prototype with documentation.
    - The platform will be used by Doll Distributing and their clients.

2. Specifications and Analysis

2.1 Proposed Design

The table shown in Figure 1 outlines a list of functional and non-functional requirements derived from discussions with the client, and the team's understanding of the frameworks and services available for use within the scope of this project.

Requirements

Functional (What functions the project needs to perform)	Non-Functional (How the project's functions must perform, constraints)
Stage 1	
Retailer ordering portal (General Base Requirement)	• Access should be through a convenient web-based portal • Access must be restricted to only registered retailers • Retailers must not be able to access information from other retailers • Web portal must integrate with current client website, using already established links on client site.
Retailer information/orders shall be stored in a database	• Database should be of some standardized format/structure • Database must be private and secured such that only client is able to access
Retailers shall be able to view products	• Products should be shown with a

	status indicating availability ● Products shown should include description of the product, or link to the client's BrandFinder page
Retailers shall be able to place orders	● Retailers should be able to submit orders through the online web-based portal ● Ordering screen should display quantities of each product ● Ordering screen should display itemized and total cost
Orders shall be sent to client for review	● Orders should be exported to compatible format to integrate with client's CMS and manual entry **OR** ● Orders should be exported directly to the client's CMS
Retailers shall be able to view previous orders	● Previous orders should be displayed with ability to download/view ● Previous orders should include itemized product list, total cost, and order/delivery dates
Stage 2	
Retailers shall have the ability to reorder a previous order	● Reordering should be combined into the same view as past orders
Reorders shall be sent to client for approval	● Orders should be exported to compatible format to integrate with client's CMS and manual entry **OR** ● Orders should be exported directly to the client's CMS
Retailers shall be able to contact client	● Web portal should provide a simple contact form ● Contact form should notify (email, text) the appropriate person within client company
Client shall be able to export retailer information/orders	● Export format should be able to integrate with VIP/RANKER
Stage 3	

Retailers shall be able to see recommendations based on local orders.	• Web portal supports recommendations while shopping
Real time updates to database	• With API to database, be able to update the product amount after Recorders approve the order.
Retailers shall be able to see when items are on sale for a special deal	• Website emphasises on the special deals to entice the Retailers.
Full production	• Website passed the development stage and testing, and ready for full deployment and use

Table 1: Project Requirements

To fulfill these requirements,

- The team has reached out to the client's website hosting company to gain knowledge of how the existing website is deployed, and the challenges associated with integrating the proposed requirements with the client's existing website.
- Research was done to determine which web development framework would be most efficient for the project. Factors such as team experience, flexibility, and documentation of the framework were all taken into account.
- Additionally, the team has also reached out to the software company behind VIP, the client's order management and inventory system, to request access to the client's existing database, in the form of an API frontend.
- The team has tested and begun the development of a simple ordering platform, utilizing Laravel [4].

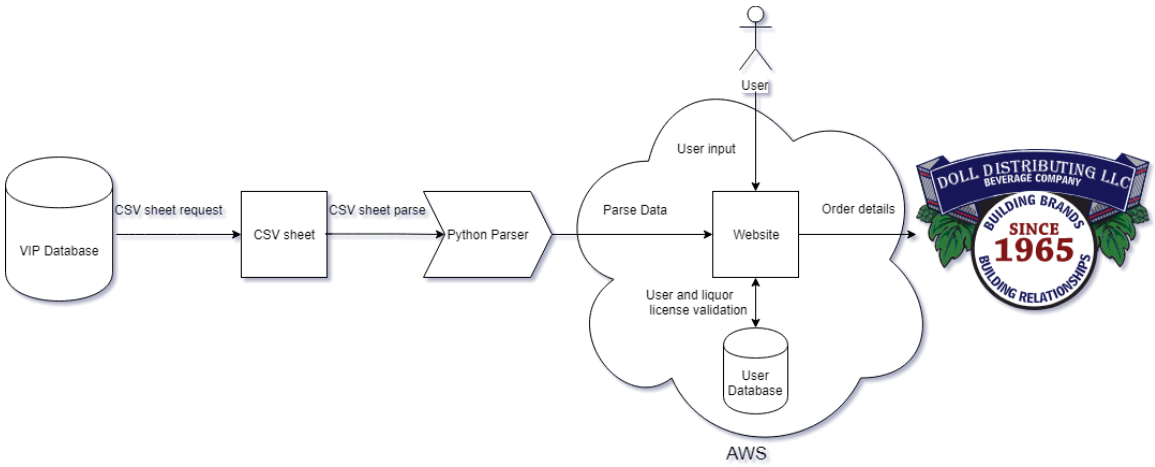

Figure 1: Proposed Architecture Diagram

To achieve a cohesive platform that allows retailers to view and submit orders, the system shown in Figure 1. has begun to be implemented. This system uses a web based frontend to effectively ensure that the interface is viewable on both mobile and desktop platforms; fulfilling the client's need for the platform to be as accessible as possible. Using Laravel [4], visual templates were created for the platform that closely resembled the client's existing website, making the experience as seamless as possible for the end user.

Additionally, a combination of backend technologies have been chosen to accommodate the client's existing environment. While the use of Laravel [4] allowed implementation of a simple Model-View-Controller design pattern, it also allowed integration of other scripts and background code. This allowed the design of a Python [6] script to process exported data from the client's existing database and retailer management system. Because of a lack of API access to the VIP database, this functionality was crucial in ensuring that the ordering platform was able to keep up to date with the client's inventory. This structure will also allow modification of the backend code in the future should such API access become available. Other web application frameworks, such as Django [3], were discussed. However Laravel [4] was chosen because of experience that several group members had, as well as its flexibility for templates that make development simpler.

Algorithm Overview

- Python Parser
    - The Python [6] parser receives a CSV sheet that is generated from Doll's VIP database. It systematically goes through each of the rows in the CSV sheet and converts them into JSON. Objects in the JSON format is easily able to be added to the app's database.
- Website Registration and login
    - The website contains forms that the user can fill out for either registration or logging in. When they are filled out, the website checks if they are valid entries, and then checks applicable entries in the database to ensure that there are no duplicates (if it is a registration), or that the username and password match (for logins).
- Website Product Viewing
    - If a user is logged in, the website displays products by systematically pulling items from the application's database. Their attributes, such as price, product image and quantity, are then placed into VUE.js card objects. Those cards are then rendered through the application so that the user can see the relevant details about all of the available products.
- Order Sending
    - When an order is placed through the website, the algorithm is simply packaging the items in the shopping cart into a standardized format containing the details such as quantity of each item, their prices, and applicable taxes. An automated script will then send that Email to the Doll representative that is related to the customer.

2.2 Design Analysis

The project has been set up in a Docker [2] environment, provided by Laradock. Also set up are initial databases, a Python [6] script to read CSVs, CSS/HTML for the app's base pages, and basic item viewing and shopping cart functionality.

The Docker/Laradock [2] setup allows for concurrent development while locally hosting the website and any other needed services, PostgresSQL etc. individually on each developer's machine.

It also allows for a consistent environment between developers; internalizing dependencies and eliminating bugs and issues during testing caused by inconsistent deployment. This speeds up the development significantly as code changes can be easily tested between developers on a consistent platform.

The project also makes use of the Laravel PHP framework, which provides several benefits. In addition to provided templates and middleware for things like user account registration and page authorization, the framework also provides a convenient database framework, Eloquent, which allows for easy integration between code models, and their records within the database. This eliminates the need for manually writing SQL queries, and also provides useful features like database seeding that are useful for testing. Finally, Laravel also provides the expandability needed for adding additional frameworks like Vue.js, which helps to make dynamic content easier and more efficient to manage. Overall, Laravel provides many of the features commonly sought after in a web application from a development and testing perspective, while also providing a functional and secure end product for the client.

Python [6] also plays a part in the project by acting as the translation between the client's existing database system, and the more modern equivalent used by the project. Due to the lack of an API-based connection to the client's existing client database and order management system, the project is utilizing a Python script designed to parse several CSV files from the client's inventory system. This script converts the needed data into a JSON format for easy integration into the project's database. Because the client also wishes to have the ability to manually verify their inventory metrics using their current system, this became the best way to allow order managers to work within the same software they are already familiar with; while also providing a level of automation that ensures the ordering platform continues to be a benefit rather than a hindrance. However, due to the nature of CSV database exports, if the format of inventory changes the script will to be updated accordingly. This is one of the many reasons for the choosing Python as the scripting language, as its easy-to-read nature and lack of compilation makes future drop-in updates far easier.

Finally the final product will function as a architecture stack utilizing Amazon Web Services [7]. AWS has already proven itself to be a stable and very affordable platform, and after an assessment of the client's hosting needs, it was determined that AWS provides the best hosting platform for the project. AWS also provides things like dynamic service scalability, allowing for hosting resources to be scaled according to use. This allows for more retailers to access the ordering site during peak times without overloading the application; all while keeping costs low and providing the stability and security of a cloud platform. In addition, the flexible Cloudformation deployment system provided by AWS integrates well with continuous integration pipelines, and allows for adding additional resources to the ordering platform should the scope of the project change during development or future use.

3 Testing and Implementation

3.1 Interface Specifications

Testing the front-end web application is done with a combination of both automated testing using Laravel Dusk [1] as well as manual project member verification. Dusk uses a headless Chrome browser to verify that the app's systems are working on the physical equipment. This is shown by testing database interactions, verifying the server is serving the website, and that the DNS is responding. If any of the physical hardware fails the testing suit will fail, because either the site won't be reachable or won't respond correctly. It will also need some Pentesting from a variety of toolsets to make sure that the application is secure and no personal information can be leaked to the public. Laravel web development testing, Laravel Dusk, will be used for the testing of the

application from a user standpoint. Then have another set of tests towards the end to make sure that all inputs have been sanitized.

3.2 Hardware and software

For testing this application, Laravel Dusk [1] testing for web applications will be used. It will be used throughout the entire project to perform testing as development progresses. PHPUnit [5] will also be used for unit testing on all of the php code. For this project, hardware for testing will not be needed since the project is entirely software based.

Laravel Dusk - provides an expressive, easy-to-use browser automation and testing API. It will run the application and make sure that it passes test cases and not go to any unexpected places. This will be a very helpful tool since every part added on to the project will be able to be tested individually and for the integration to all the other segments at the same time.

PHPUnit - a programmer-oriented testing framework for PHP. Used to verify that the behavior of the specific components.

3.3 Functional Testing

Unit Testing: PHPUnit

PHPUnit [5] uses assertions to verify that the behavior of the specific component being tested behaves as it is expected to behave. After code is created, it should go through tests that are made to be sure that any changes that was made to the PHP code will function as expected and not do anything that isn't expected.

Integration Testing: Laravel Dusk

This type of testing will be from a user's standpoint. The test will run in a browser-type environment and test the app's links to make sure that they go where is expected of them. This will also automate the need to test to make sure that a new implementation won't break the rest of the implementations. This also involves checking that a user won't be able to access certain sites that they normally shouldn't. This will greatly speed up testing all edge cases on new features.

If something were to go wrong in the above two tests, then the results would be interpreted as whatever was just pushed to the server wasn't implemented correctly with the rest of the services. Which will be easier to pinpoint with automation from the above testing frameworks.

System Testing: Since the application will be a separate instance from where Doll currently has their website, testing for integrations will not be necessary except for manual/automated testing to see if links go back correctly to their main page.

Acceptance Testing: Acceptance testing will be getting approval from Doll.

Doll will check over the website on new features that are implemented to make sure it is what they expected it to look like and agree to go ahead with the implementation. This also includes after approval to go live, Doll will be able to receive suggestions from their clients so that the application can be implemented/improved on.

3.4 Non-Functional Testing

Using Laravel Dusk [1], team members are able to design a series of test-cases that mimic the performance and security constraints mandated by the non-functional project requirements. This system allows each group member to run a simple series of pre-created test cases against their code before commiting to the project repository, allowing for manual verification of non-functional requirements during the coding process. This system also allows for continuous integration testing enabled through features available on GitLab; though this process has been reserved for later use, as some non-functional requirements were expected to change.

Dusk provides a simple output of passing and not-passing tests, allowing the user to quickly verify whether a page or database connection takes too long to respond, or whether a particular request is handled insecurely. It also provides to the user an overall summary of the tests that were unsuccessful, enabling the user to quickly identify problem areas, speeding up the development cycle. Additionally, this provides a very simple means of verifying code written by other group members, and provides the pathway to continuous integration previously alluded to. With this functionality, the desired outcome is that all established tests pass without error, and that any changes to functional requirements are reflected in the test code, so that the overall project can be quickly re-evaluated and updated accordingly.

Overall, this system allows each member to test the project using a common set of test code, enabling for testing during development, and easier verification of code during merge requests. Additionally, the functionality provided by Dusk allows for simulating all the requirements of the project with consistent and measurable metrics.

3.5 Process

Section two will be tested in one of two ways depending on the developer. It will either be test driven development or tested as it's being created. Any code the has a very heavy computation or logic will be test driven. This will reduce bugs and produce a higher quality of code. If the piece of code is more of a UI element with little to no logic, tests will be produced after the code is written as the UI is improved.

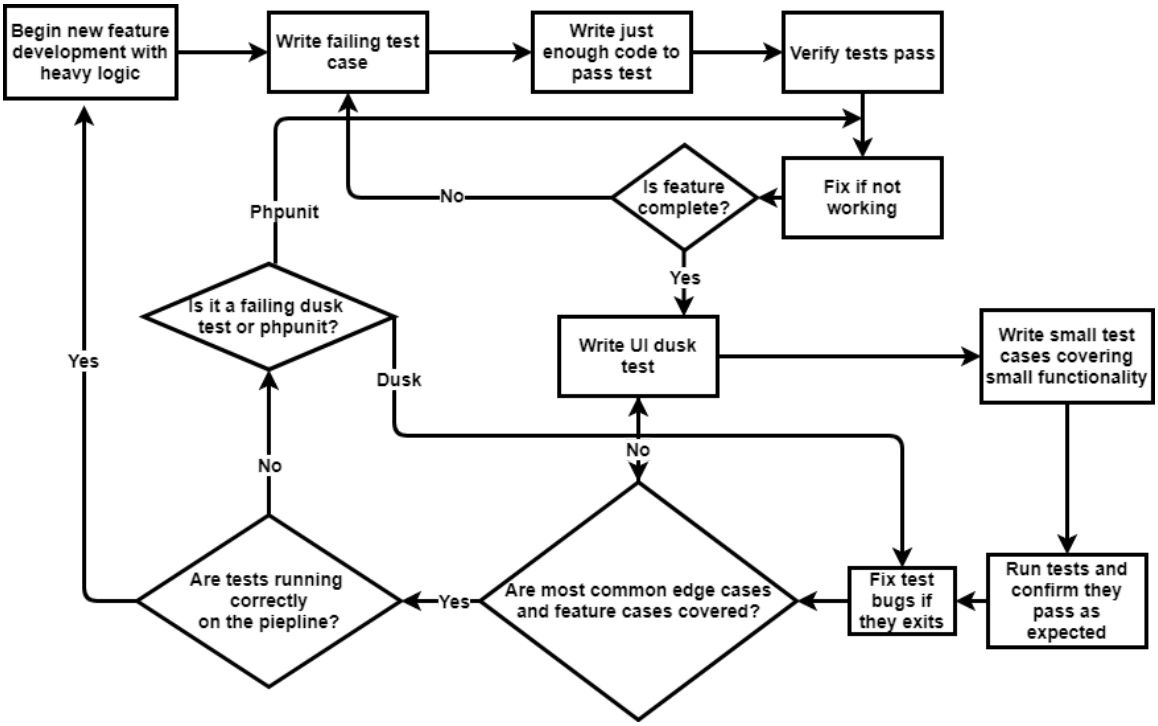

Figure 2: Heavy Logic Flow

The diagram above is showing shows how new features will be developed with heavy logic (like a new registration page). First, a new feature will begin by writing tests for it. Next, code will be written to pass those tests. After it is confirmed that all tests pass, UI tests will be written. When that is accomplished it will be confirmed that the UI tests pass as well. If all tests pass and cover all edge cases more tests will be run on the build server. If they pass there work on another new feature can be started.

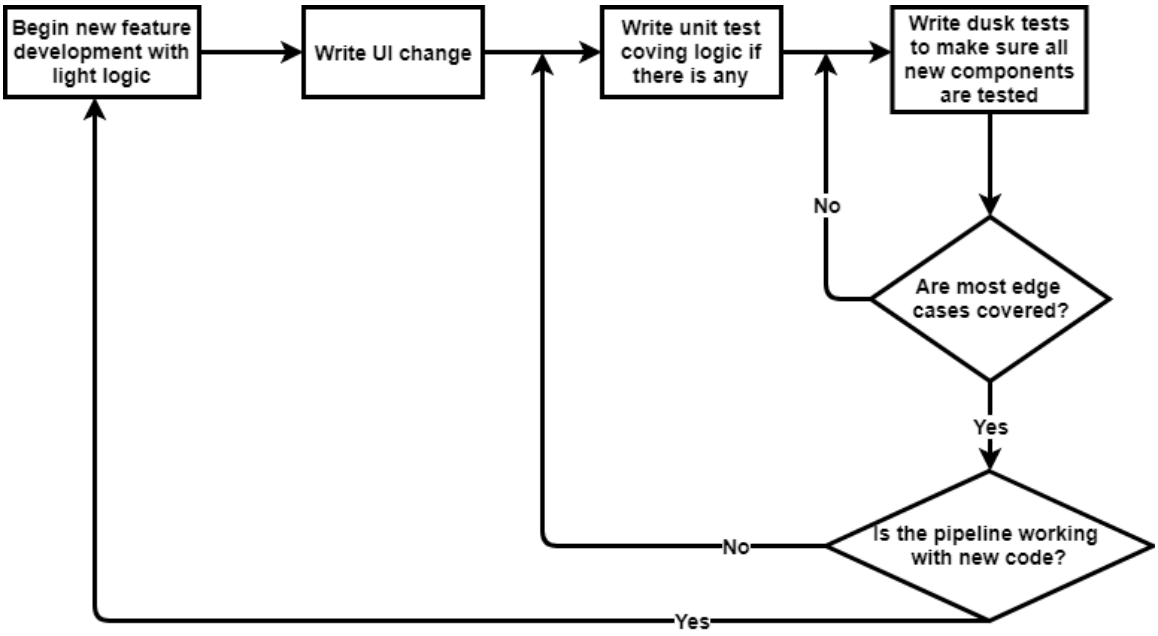

Figure 3: Light Logic Flow

This is the new feature flow for light logic (like a plain HTML page with little to no logic). It will be started by coding a new UI for the project. Next, tests will be written covering any logic that is created. Dusk shall then be used to confirm any UI elements added are there and keep doing that until all obvious edge cases have been covered. Then, once the code is approved for merging by two other teammates it will be run on the build pipeline. If that passes, then it is ready for production.

3.6 Results and Challenges

The biggest implementation issue is not having API access to the inventory system that Doll has in place. This is due to the third party company having an existing system that does something similar, but is extremely dated. Due to this fact a python script will be implemented that will parse an exported CSV for inventory data and storing it in the app's own database for inventory purposes. When Doll's account reps approve the orders that will update in the third party system and will propagate in later CSVs to the system. It is also planned to have some sort of web scraper to run the exports due to the fact that each export can take upwards of twenty minutes. This will be run as a cron job, so that the app can have as up to date inventory as possible. Having a system that updates around every twenty minutes is more responsive than the current systems Doll has in place that update daily.

Some major issues that were found from using the testing techniques was that the login and register pages wasn't hashing properly. This was a major issue since whenever a new user would be added to the system, the code would first take the password and hash it and salt it for encryption security, in which it did correctly. Then afterwards it would take the already hashed and salted password and rehash that with another salt value. When a user went to log back in the password would try to authenticate against the double encrypted passcode and fail. This is related to the design because currently it wouldn't be able to authenticate any user and no one could get into the system. Without testing whenever code was passed in to the project, it would've been very difficult to be able to detect where the problem lays that broke this.

3.7 Modeling and simulation

As this is a software project, there are not any models to show in this document. The project will run when built in docker and display the current build of the page when navigated to localhost.

4 Closing Material

4.1 Conclusion

So far, the basic features for the Online Ordering Platform are set up. There is currently a website that can be locally hosted on the team's machines. On that site a user can register an account and

log in. The user can view various products when logged in, and can select items to purchase via shopping cart functionality. The site has also been modified to look similar to Doll's website.

The current goals include improving the front end of the site, then focusing on the necessary back end functionality. One of the most important goals is to implement product ordering, which will require both front end and back end functionality. After this is completed, database requirements can be focused on, such as allowing the data to be easily exported by Doll, and updating the inventory and pricing data as often as possible. A way for users to communicate with a representative of Doll if the user has a question about a product will also be implemented.

The design choice of using Laravel provides a solid foundation for the project. VIP has not given the team access to Doll's database, so the data must be obtained using CSV data files. Laravel allows the project to include a Python script that will be used to parse these CSV files. Docker allows for the website to be hosted while it is developed. AWS will be used to host the product when it is ready to be used by Doll. This allows the client to have the platform hosted for a minimal cost.

The end product will be a fully functional online ordering platform that Doll Distributing can use to make their ordering process more convenient for their customers. The customers will be able to view Doll's inventory, place orders, and communicate with Doll representatives using this system.

4.2 References

[1] T. Otwell, "Browser Tests (Laravel Dusk)" *Laravel - The PHP Framework For Web Artisans*. [Online]. Available: https://laravel.com/docs/5.7/dusk. [Accessed: 2-Dec-2018]

[2] "Docker," *Docker*. [Online]. Available: https://www.docker.com/. [Accessed: 2-Dec-2018].

[3] "Django Homepage," *The Web framework for perfectionists with deadlines | Django*. [Online]. Available: https://www.djangoproject.com/. [Accessed: 2-Dec-2018].

[4] T. Otwell, "Laravel Homepage," *Laravel - The PHP Framework For Web Artisans*. [Online]. Available: https://laravel.com/. [Accessed: 2-Dec-2018].

[5] S. Bergmann,"PHPUnit Manual" *PHPUnit - A programmer oriented testing framework for PHP*. [Online]. Available: https://phpunit.readthedocs.io/en/7.4/ [Accessed: 2-Dec-2018].

[6] "Python Documentation," *Python.org*. [Online]. Available: https://www.python.org/doc/ [Accessed: 2-Dec-2018].

[7] Amazon Web Services, Inc. "Web Hosting" [Online]. Available: https://aws.amazon.com/websites/?nc2=h_m2 [Accessed: 2-Dec-2018]

4.3 Appendices