



ASHESI UNIVERSITY

CONNECTING ERP TO E-COMMERCE: NPONTU AS A CASE STUDY

APPLIED PROJECT

B.Sc. Management Information Systems

Allotei Pappoe

2021

ASHESI UNIVERSITY COLLEGE

CONNECTING ERP TO ECOMMERCE: NPONTU AS A CASE STUDY

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi
University College in partial fulfilment of the requirements for the award of
Bachelor of Science degree in Management Information Systems.

Allotei Pappoe

2021

DECLARATION

I hereby declare that this Applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

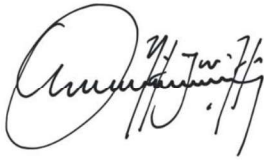


.....
Candidate's Name: Allotei Pappoe

.....
Date: 14th May 2021

I hereby declare that preparation and presentation of Applied project were supervised in accordance with the guidelines on supervision of Applied project laid down by Ashesi University College.

Supervisor's Signature:



.....
Supervisor's Name: Stephane Nwolley

.....
Date: 17/05/2021

ACKNOWLEDGEMENTS

I would like to acknowledge all the individuals who aided in completing this project: Stephane Nwolley (PhD), my supervisor, who was of immense help throughout the completion of this project; his advice and technical insight guided me throughout. Abraham Odoi and the engineering team at Npontu Technologies for providing me with the interfaces and help for which this project could not have been completed without. Also, a big thanks to my family, who provided me with unending support, notably my brother, Allotey Pappoe, for reviewing the paper. To the Beta testers, who without their user insight, the project would not have been completed, thank you. Finally, to Marian-Bernice Haligah for her unrelenting support and encouragement. Without you all, I would not have been able to complete this project.

Abstract

Enterprise Resource Planning (ERP) is a business process that enables companies to manage and integrate their essential business processes. ERP applications are available for companies to use for integrating all their business functions into one single system. It allows companies to manage their daily activities, including accounting, procurement, and inventory management. E-commerce, also referred to as electronic commerce, is the buying and selling (trading) of goods and services on the internet. E-commerce platforms give sellers the needed tools to provide 24/7 services to their customers unrestricted by distance. They allow sellers to accept and track orders from their customers, which helps fulfil their value proposition in ways that were not possible with previous brick-and-mortar stores. Studies show that ERP users in Ghana hardly use E-commerce and the internet as tools to expand their market. This project is aimed at integrating e-commerce systems into ERPs such that business owners can use their ERP and e-commerce system in one software suite to give business owners access to the online market.

TABLE OF CONTENTS

DECLARATION.....	I
ACKNOWLEDGEMENTS	II
Abstract.....	III
TABLE OF CONTENTS	IV
Chapter 1: Introduction	1
1.1 Background.....	1
1.2 Problem Statement.....	2
1.3 Motivation	2
1.4 Significance	3
1.5 Related Works	3
1.6 The Kedebah ERP case study	5
1.7 Proposed Solution.....	5
1.7 Features of the proposed solution	5
Chapter 2: Requirement Analysis	7
2.1 Introduction	7
2.2 Overall Description	7
2.3 Requirements.....	8
2.3.1 Requirement gathering:	8
2.3.2 Requirement Analysis	10
2.3.3 Requirement Modelling	10
2.3.3.1 Users.....	10

2.3.3.2 Use Case	11
2.3.3.3 Functional Requirements	11
2.3.3.3.1 ERP:.....	11
2.3.3.3.2 Ecommerce:	13
2.3.3.4 Non-Functional Requirements.....	13
2.3.4 Summary of Requirements to be worked on.....	14
Chapter 3: System Architecture	16
3.1 Introduction	16
3.2 System Architecture	16
3.3 Software Architecture.....	16
3.4 High level System Overview	18
3.6 Database	21
3.7 REST APIs for Bi-Directional Communication	22
3.7.1 High level overview:	22
Chapter 4: Implementation.....	25
4.1 Introduction	25
4.2. Software Development Technique	25
4.3 Key Technologies.....	25
4.3.1 Languages.....	25
4.3.1.1 Frontend Languages	25
4.3.1.2 Backend Languages	26

4.3.2 Frameworks and Libraries	27
4.4 Application Programming Interface (APIs)	28
4.5 Evidence of Implementation	28
4.5.1 Ecommerce Platform	28
4.5.2. Product Recommendation	33
4.5.2.1 Collaborative Filtering	33
4.5.2.2 Collaborative Filtering Using K-Nearest Neighbors	34
4.5.2.3 Neural Network-Based Collaborative Filtering (NCF)	35
4.5.3 Application Programming Interfaces for E-commerce platform	37
4.5.3.1 Seller API	37
4.5.3.2 Category API	38
4.5.3.3 Products	39
4.5.3.4 Orders	40
4.5.3.5 Connecting To Woo-commerce	41
4.5.4 Social Media Posts	44
4.5.1 Evidence of Implementation	45
Chapter 5: Testing and Results	49
5.1 Introduction	49
5.2 User Acceptance testing	49
5.3 Unit testing	51
5.4 Component Testing	53

5.5 System Testing.....	59
Chapter 6: Conclusion and Recommendation	61
6.1 Conclusion.....	61
6.2 Limitations	61
6.3 Recommendations	62
References.....	64
Appendices.....	67
Appendix A: Npontu Mall API Documentation	67
Appendix B: Connecting Kedebah ERP to Woo-commerce	77
Connecting Kedebah to Twitter.....	80

Table Of Figures

Figure 3.2 MVC Architecture	17
Figure 3.3 Buyer Sequence Diagram	19
Figure 3.4 Seller Sequence Diagram	20
Figure 3.5 Use Case Diagram	21
Figure 3.6 ER Diagram	22
Figure 3.7 RESTful Architecture	24
Figure 4.1: Shop page	29
Figure 4.2: Single Product	29
Figure 4.3 Wishlist Page	30
Figure 4.4 Cart Page	30
Figure 4.5 Checkout Page	31
Figure 4.6: Order Completed Page	31
Figure 4.7: Related Products	32
Figure 4.8: PHP Functions for related products	32
Figure 4.9: Product Recommendation Based On Users Reviews	33
Figure 4.10: Python Function for recommending products	34
Figure 4.11: KNN Recommending Products based on Rating Similarity	35
Figure 4.12: Implementation of KNN Algorithm	35
Figure 4.13: NCF Model Implementation	36
Figure 4.14: NCF Model in Action	37
Figure 4.15: Seller API	38
Figure 4.16: Category API	39
Figure 4.17: Product API	40
Figure 4.20: Order API request	41

Figure 4.19: Connecting to Woo-commerce Store Front View	42
Figure 4.20: Backend for connecting to user's Woo-commerce Store	43
Figure 4.21: Snippet of Code for connecting to Woo-commerce store	44
Figure 4.22: Logging in to the Twitter Account	45
Figure 4.23: Tweeting about a product	45
Figure 4.24: Post made successfully	46
Figure 4.25: Twitter OAuth Login	47
Figure 4.26: Twitter Callback	48
Figure 4.27: Posting a Tweet with an Image	48
Figure 5.1: Beta testing responses	50
Figure 5.2: Route Test	51
Figure 5.3: Product API Test	52
Figure 5.4: Seller API Test	52
Figure 5.5: Unit Testing Results	53
Figure 5.6: Method to store a new seller	54
Figure 5.7 Method to store new category.	55
Figure 5.8: Code to test if parameters have been met	56
Figure 5.9 Code for storing product if parameters are met	56
Figure 5.10 Code for add product to wish list	57
Figure 5.11 Code for moving product from wish list to cart	57
Figure 5.12 Code for removing product from wish list	58
Figure 5.13: Code to Add to cart	58
Figure 5.14: Code to update cart	59
Figure 5.15: Code to remove item from cart	59
Figure 1	81

Figure 2	81
----------------	----

Chapter 1: Introduction

1.1 Background

Enterprise Resource Planning (ERP) is defined as an integrated suite of business processes. Organisations use it to manage their business functions in a centralised system. Some features of an ERP include product life cycle management, inventory management, supply chain management, warehouse management, human resource management accounting, financial management, customer relationship management and sales order management. ERP software has gained a lot of traction amongst many enterprises mainly because it allows for increased effective and efficient management [1].

The internet and its myriad of benefits also provide entrepreneurs with a new platform to market and sell their goods and services. Thus, the rise of e-commerce solutions on the web. These solutions offer entrepreneurs the avenue to move away from brick-and-mortar operations and concentrate on selling to a broader market conveniently.

Despite the growing mobile phone penetration and internet access in Ghana, business owners (ERP users) do not use the internet and the e-commerce opportunities it brings [9]. This is a huge problem because there is a booming online market that is not fully being utilised. So, there is a need to integrate e-commerce platforms into ERPs to allow its users to sell on the online market.

Some business owners already sell on existing e-commerce platforms. However, these e-commerce platforms are not integrated into their ERPs [1]. When organisations use the two disjointly, synchronising their data into one for executive use (planning, budgeting, etc.) is often tremendously tedious. These businesses hire labour to manually feed data at both end-points to ensure that all data is up-to-date [1]. This is counter-productive to these businesses as manual labour is error-prone.

Moreover, this practice is inefficient as it is time-wasting. A few mistakes could arise when businesses decide to manually enter data, including: incorrect product information, outdated product prices, confusion about stock levels, missing orders, and invalid shipping addresses. Accuracy is vital for retaining customers; thus, any of these errors could adversely impact customer experience with the business, thereby jeopardising the business's goodwill and, in the long run, success.

1.2 Problem Statement

A summary of the problem space is businesses use ERPs to manage their inventory of items. Though the internet is at their disposal, they do not make use of its channels, that is, e-commerce and social media platforms, to take absolute advantage and sell to the growing online market in Ghana [9]. Furthermore, after successfully implementing e-commerce platforms to sell their items, the need arises to seamlessly integrate the two systems (ERP and e-commerce platforms) to share data for executive use.

1.3 Motivation

Online trading is booming in Africa because of the increasing accessibility of the internet. For instance, Jumia alone has served over 1.2 billion consumers and 17 million small and medium scale enterprises in the continent [2]. Web store owners are experiencing a boom in their online sales, meaning; they are completing more transactions online. It is paramount that Ghanaian business owners capture this value. Unfortunately, according to the Ministry of Trade, Ghanaian businesses hardly expose their inventory to e-commerce channels. Considering that the end goal is to enable ERP users to sell their inventory on the online market, it is pertinent that business owners are able to maintain their online sales systems and their internal ERP systems simultaneously to ensure a level of cohesion. These two systems, ERP and e-commerce, need to work uniformly to achieve this goal.

1.4 Significance

The benefits of e-commerce to Ghanaian business owners cannot be underemphasised. With e-commerce, business owners do not need a physical storefront, thus lowering the business's fixed operating costs. More so, selling online is an exquisite source of customer data for business owners, which can be used for targeted email marketing. Also, because e-commerce systems are available on the internet, they are exposed to the global online market. Finally, e-commerce systems are ubiquitous so they will always be available for customers to place orders [10].

These are just a few of the benefits of having an e-commerce store. However, when the e-commerce platform is directly integrated into an ERP, businesses stand to benefit a lot more. For instance, when an e-commerce store is integrated with an ERP, it increases trust and credibility because, at all times, the stock amount on the e-commerce website is the same as that in the ERP. The customer thus cannot be disappointed by any misrepresentation on the website. Similarly, an integrated system eliminates human errors like duplication that may arise when the business relies on manually entering data on both systems. Also, it leads to a more cost-effective way of managing inventory. Since inventory levels are always up to date, it helps to understand which products sell well, and the business can predict when to stock up on inventory to ensure that there is no downtime [3].

1.5 Related Works

Systems that integrate ERP with e-commerce

Some solutions have already been developed to allow ERP users to connect to e-commerce platforms to sell. Some of the e-commerce platforms supported by these solutions are Magento, Etsy, and eBay. This section looks at some of these solutions and which could be of relevance to this project.

Bridging the Gap between ERP Applications and eCommerce Solutions

Based on the problems posed by having independent systems, the authors of this paper [1] proved how current practices at the time were inefficient. Their solution was to build a middleware between the two platforms that would be cost-effective, efficient and easy to implement. The middleware would map ERP processes with the e-commerce solutions used by organisations and utilise APIs services for synchronisation and data accessibility between the two platforms.

nChannel

Another e-commerce ERP connector on the market is nChannel. It is a cloud-based SaaS middleware that connects two end-point systems via Application Program Interfaces (APIs) which helps to sync and automate business processes. It provides integration to many platforms, including Shopify, Magento, Amazon, eBay and Etsy [4].

SYNC Connection by Harris WebWorks

SYNC Connection by Harris Web Works is a diligently engineered integration tool designed to connect existing ERP systems to Magento e-commerce websites. With the application's synchronising online orders, customer data, and inventory information with ERP systems like SAP, Dynamics, and NetSuite [5].

eBridge Connections

eBridge Connections is an iPaaS (integration Platform as a Service) built for the seamless data flow between ERP systems, e-commerce shops and Customer Relationship Managers (CRM). The platform as nChannel integrates a growing number of end-points [6].

1.6 The Kedebah ERP case study

The Kedebah ERP by Npontu technologies was used as a case study for this project. The solutions explored in the previous section are not built for the Ghanaian market and the ERPs developed in the country. Thus, this project focuses on the Kedebah ERP and how to integrate it into e-commerce and social media channels.

1.7 Proposed Solution

For this project, working closely with the engineering team at Npontu Technologies, the proposed solution is granting the Kedebah ERP system e-commerce and social media advertising capabilities. This will involve building an e-commerce platform dubbed Npontu Mall that will enable businesses on the Kedebah platform to publish their products on the platform where customers can place orders, easily. The platform will house an AI-powered recommendation engine that will learn from customer behaviour and recommend potential products to them. The solution will also allow users of the ERP to connect to their independent online stores, including Woo-commerce and post products there. When customers complete a purchase on this platform, the order details will be sent to the Kedebah ERP, storing them, updating the inventory, and the books of accounts (sales ledgers) of the seller. It will also involve building a social media integration that will allow the ERP user to advertise their products on social media (Twitter will be explored in this project). The appendix section of this paper documents the APIs in the Npontu Mall that will enable integration with the Kedebah ERP. It also supports the modules that will have to be included in the Kedebah ERP which will allow it to connect to Woo-commerce as well as to Twitter.

1.7 Features of the proposed solution

1. An e-commerce platform to allow users of the inventory system to sell their products online. The platform will have the ability to:

- a.** Display all products of the seller.
 - b.** Support a purchase flow including:
 - i.** Adding to wish lists.
 - ii.** Cart management.
 - iii.** Checkout and
 - iv.** Payment with cash, Mobile Money or Debit card
- 2.** Application Programming Interfaces to connect the ERP to the e-commerce platforms to synchronise data between the two and connect the ERP to social media to allow the user to advertise their products.
- 3.** An AI-powered recommendation engine on the Npontu Mall to recommend products to buyers. This will be built using:
 - a.** Item-item collaborative filtering
 - b.** Collaborative Filtering using K-nearest neighbour algorithm.
 - c.** Neural Collaborative Filtering implemented using deep learning methods.

Chapter 2: Requirement Analysis

2.1 Introduction

The proposed system will be divided into two main components: Npontu mall which will allow the user sell their items, and the integration of the ERP with e-commerce platforms (Npontu Mall and woo-commerce) and social media platforms.

2.2 Overall Description

The proposed solution will essentially enable the user to connect to any e-commerce shop they own, including the Npontu Mall and any social media platform. It will provide access to the ERP to push product data to the platforms and receive customer order data.

Once the users upload a new product to the ERP, they will be provided with the option of publishing to the e-commerce platforms. If they choose to, the system will make an API call to the e-commerce platforms, sending product data such as name, price, the available stock and pictures. The e-commerce platform will in turn receive the uploaded data and store it into a database, which will then be displayed to buyers. The users will also have the option of posting products to social media. Once again, if they choose to, an API call will be made to the social media platform allowing the product data to be sent there for advertising.

At the other end-point, when a purchase is made on the online shop, an API call will be made to the ERP with the necessary order data such as the product, quantity, price and phone number of the Buyer and the payment method used. The ERP will receive this data and store it, alerting the seller of a new purchase.

Finally, on the Npontu Mall, the AI-powered recommendation engine will recommend to the Buyer what to buy based on their purchase history.

2.3 Requirements

To get the grand picture of the requirements needed for a holistic solution, the procedure followed for analysis requirements is outlined below:

2.3.1 Requirement gathering:

Requirement gathering for the solution was done by interviewing stakeholders from Npontu Technologies and having focus group discussions with teams in the company. The researcher benchmarked other similar systems, namely Jumia and Jiji for the Npontu Mall, and nChannel [4] and SYNC Connection [5] for ERP e-commerce integration, to acquire the requirements that will solve the problem while other requirements were gathered from the Functional Requirement Document (FRD) of the Kedebah ERP.

In the benchmarking research of other similar solutions, it was evident that the use of APIs cannot be ignored. It was essential to use REST APIs to send data between the systems. Also, there was the need to consider the data formats both systems use. The Kedebah ERP uses JSON format, so it was necessary to pass JSON data to it.

As part of the interviews, the researcher took into consideration the end-users of the solution and the critical features they need in the solution. Finally, the platforms the end-users would connect to the ERP were also factored in the analysis.

From the interviews, researching the FRD and benchmarking, the researcher ascertained the following requirements of the solution.

1. Building a test inventory system to mimic the Kedebah ERP:
 - a. Connecting the inventory system to e-commerce platforms
 - b. Connecting the inventory system to social media platforms
 - c. Documenting how the Kedebah ERP can be customised to the e-commerce and social media platform

2. Building an e-commerce platform to sell the inventory (Npontu Mall)
 - a. Adding, displaying and updating products.
 - b. Cart management.
 - c. Payment Processing: Cash or Mobile Money (MTN, Vodafone, AirtelTigo),
Debit Card
 - d. Requiring buyers to provide their billing and contact details on each purchase.
3. AI-powered product recommendation system to recommend products to buyers.
4. The e-commerce platform should be secure.
5. It should always be available.
6. It should be easy to use.
7. The Kedebah ERP should be integrated with the following e-commerce platforms:
 - a. Npontu Mall
 - b. Amazon
 - c. Alibaba
 - d. Woo-commerce
 - e. Tonaton
 - f. Jiji
8. It should connect to the following social media platforms:
 - a. Facebook
 - b. Twitter
 - c. Instagram
 - d. Google My Business

2.3.2 Requirement Analysis

Analysing the requirements gathered, it proved impossible to connect to Amazon and Alibaba since they do not support selling in Ghana. Discovering that these requirements cannot be completed, some agile iterations were made with the Npontu team to understand the current restrictions so Amazon and Alibaba would be taken out of the platforms to connect the ERP to. Likewise for Tonaton and Jiji who did not provide any APIs for third parties to connect to their platform. For the social media platforms, Facebook only permits registered business account to post on other business accounts pages. Likewise, for Instagram. Hence, this project will focus on only connecting to Twitter since the researcher does not have access to a business account. Google My Business also required the researcher to be verified as a business by the team at Google which proved impossible.

2.3.3 Requirement Modelling

The accepted requirements including the users, use case scenarios, functional and non-functional requirements are documented as follows.

2.3.3.1 Users

The solution will have two sets of users being the buyers and the sellers. The sellers will be the users of the ERP who want to advertise their products on social media and have an e-commerce website to sell their products. These users will add and update their products on the ERP, choose to publish their products on e-commerce websites and social media.

The buyers will be anyone interested in purchasing products from the seller. When they interact with the social media post, there will be a link attached to buy the product they are interested in. When clicked, they will be redirected to the e-commerce shop where they can add to cart, checkout and pay.

2.3.3.2 Use Case

Buyer: A scenario of how a user will use the solution will be:

1. View all products being sold.
2. Add the product of choice to their cart.
3. Checkout the order
4. Pay with Mobile Money, Debit Card or cash

Seller: A scenario of how a seller will use the solution will be:

1. Log in to the ERP.
2. Add a product.
3. Update the product when a purchase has been completed.
4. Advertise the product on social media.

2.3.3.3 Functional Requirements

2.3.3.3.1 ERP:

Publishing products to the Ecommerce platforms

Sequence:

1. The user clicks on the product they want to publish to the e-commerce platform.
2. The user chooses which e-commerce platform to publish (Npontu Mall or Woocommerce)
3. An API call is made to send the product data to the e-commerce platform.

User Requirements

1. The user must provide click on which product to add.
2. The user must choose which platform to publish the product to

System requirements

API request to the e-commerce platform

Update Product

Sequence

1. The user selects the product to be updated.
2. The user inserts the new data to be updated.
3. The system validates the data and stores it.
4. Sends an update API request to the e-commerce website to update the product

User requirements

1. The user must provide information to be updated

System requirements

1. Accept this data and update the database.
2. Make update request to the e-commerce website.

Post to Social media

Sequence

1. The user clicks on the products to list all the products in the inventory.
2. The user picks the particular product to promote
3. The user picks the particular platform to promote on
4. The user provides a caption and an image of the product and clicks the promote button to promote
5. The system makes an API call to the platform chosen to make the post

User requirements

1. The user must provide an image and a caption

System requirements

1. Accepts user inputs: images and captions and posts them to the specified platform

2.3.3.3.2 Ecommerce:

Product display

Sequence:

1. The website displays the products from the database.
2. If the user is a regular user, the system learns from his or her purchasing actions and suggests the product he or she is likely to buy.

Purchase flow:

This details how the user places a purchase order.

Sequence:

1. The user chooses the product to buy.
2. The user adds it to their cart.
3. The user goes into the cart and checks out.
4. The user chooses his preferred mode of payment and pays if it is non-cash.
5. The system records the data and sends the purchase order alongside the customer details to the ERP for quick order fulfilment.

2.3.3.4 Non-Functional Requirements

Performance: The solution should be able to handle multiple requests from several users without failure

Availability: This solution should always function. Because of the ubiquity of e-commerce, it is necessary to ensure that the system is available all the time to maximise buyer satisfaction.

Security: The system should be secure at all times. The system architecture should ensure that during data transfer, the payloads are always secure to prevent alteration by third parties. The API should be secured with a password to ensure that only authorised parties can consume or make API calls. The e-commerce should also have SSL installed since it will be processing payment.

2.3.4 Summary of Requirements to be worked on

Table 2.1: Summary of Requirements

Requirement Number	Requirement	Sub requirements
R001	Test Inventory system	<ol style="list-style-type: none"> 1. Connecting it to ecommerce stores: Npontu Mall and Woocommerce 2. Connecting it to social media: Twittter
R002	Npontu Mall	<ol style="list-style-type: none"> 1. Displaying products from inventory 2. Cart management 3. Checkout 4. Payment processing 5. Accepting customer reviews

R003	AI Recommendation systems based on user reviews on products	<ol style="list-style-type: none"> 1. Deep Neural Networks (Neural Collaborative filtering) 2. Collaborative Filtering 3. Collaborative Filtering using K-Nearest Neighbor
------	---	---

Chapter 3: System Architecture

3.1 Introduction

This chapter details high-level models of the architecture of the entire solution. It includes sequence diagrams to describe the flow of activities of the two users in their respective use case scenarios. It also depicts the Entity Relationship Diagram of the databases used in the application.

3.2 System Architecture

The entire solution on both ends (e-commerce and ERP) will make use of the three-tier client-server architecture where buyers and sellers (client) will make requests via a browser to the respective systems e-commerce platform and ERP (servers). The servers will process the requests and query a database server to return results to the user.

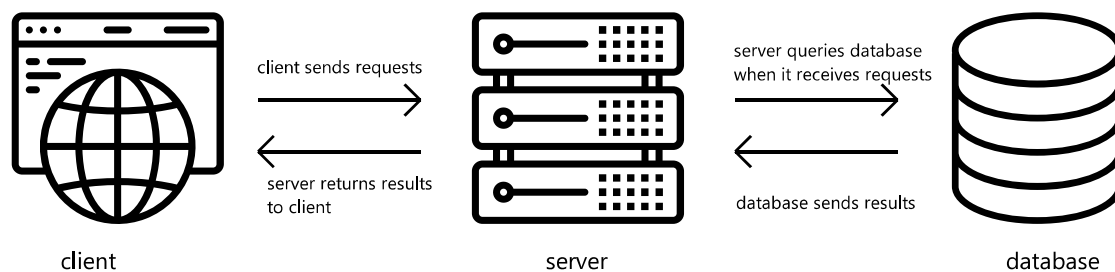


Figure 3.1 Client-Server Architecture

3.3 Software Architecture

Both systems will be built using the Model View Controller architectural pattern.

1. The Model, which is the business logic of the application, is responsible for retrieving, inserting, updating and deleting data from the database.
2. The View is responsible for displaying data to the user. For the e-commerce side this will involve showing registration and login forms displaying the products, wish list, cart and checkout pages. While on the ERP side, this will involve showing login and

registration forms, the add product form, the products the user has, the update product form, buttons for publishing products to the e-commerce platform as well.

3. The Controller serves as a middle-man between the two. It accepts user data from the View and processes it before sending it to the Model to be stored into the database.

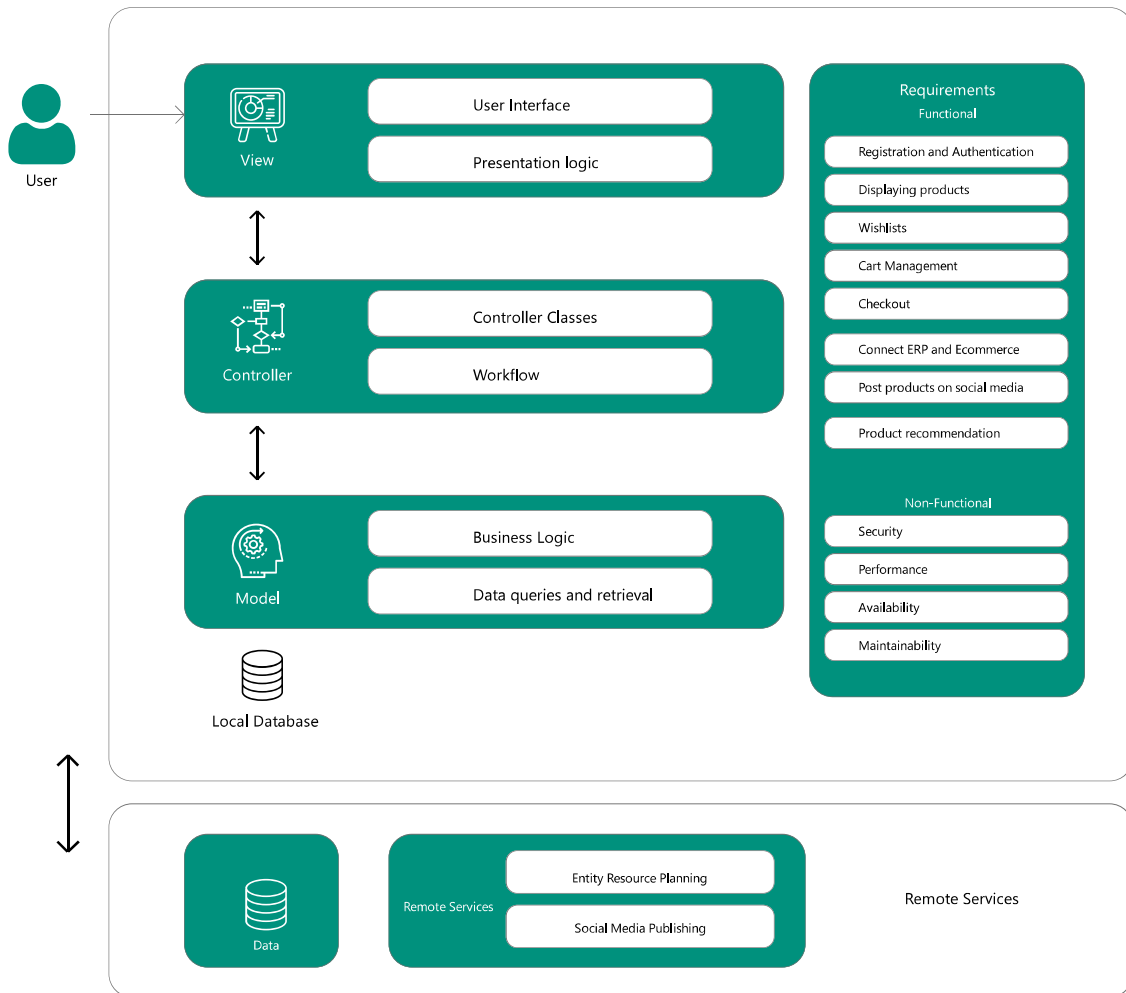


Figure 3.2 MVC Architecture

3.4 High level System Overview

When users first visit the Npontu Mall, they are allowed to browse through the catalogue and choose their preferred product. They can add the product to their cart, where they can view, update, and delete them. Once satisfied with the cart's content, they can proceed to checkout where they will be asked for their preferred mode of payment (cash, Mobile Money, Debit card). When the Mobile Money or debit card option is selected, the order is marked as paid while the cash marked as unpaid.

Figure 3.3 below depicts the flow of events for the Buyer. When the Buyer visits the page, they can browse through the product catalogue and select whichever product they want to buy. They do this by adding to their wish list or cart. When the item is in their wish list, they have the option of later adding it to their cart. When the item is added to their cart, they are given the opportunity to manage the quantity of the item. At the checkout, they will be asked for their shipping details and the mode of payment. Thereafter, the backend of the application makes an API call sending the order details to the ERP for fulfillment by the seller. Buyers also have the option of reviewing products. By doing this, the application collects data about the user to make tailored product recommendations.

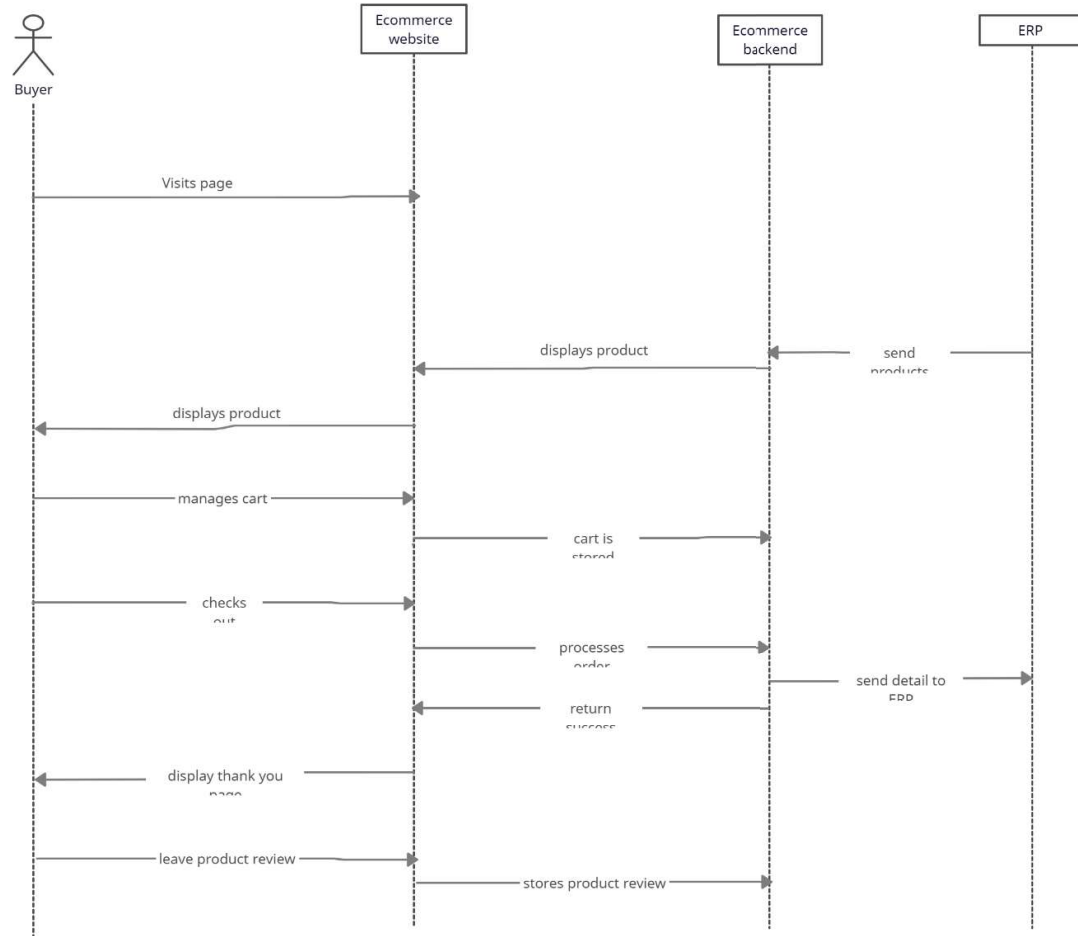


Figure 3.3 Buyer Sequence Diagram

On the ERP side, the seller has to register and login before using the application. After authentication, they have to add a category of product before they can add the product. The ERP backend makes an API call to the e-commerce backend sending the product details to be published there. The seller also has the option of posting ads on their social media platform via the ERP. They enter a caption and upload an image, and then the ad is published to the platform.

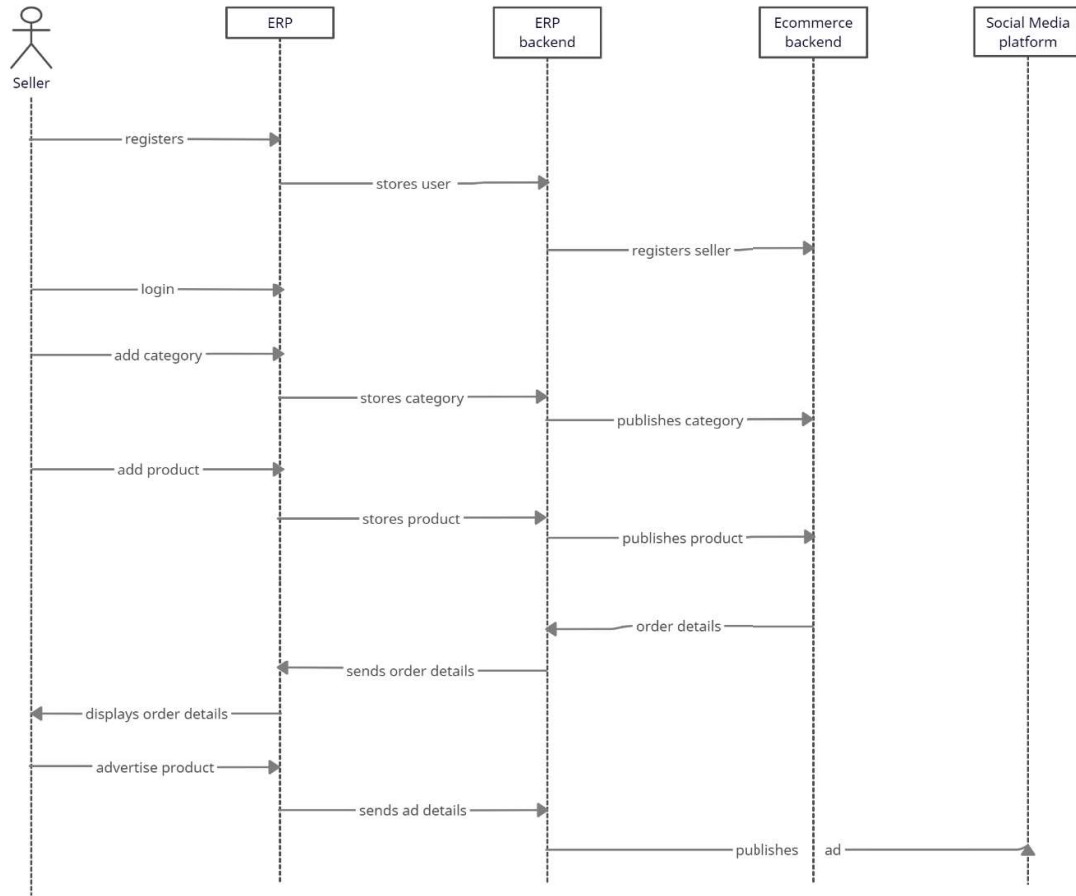


Figure 3.4 Seller Sequence Diagram

3.5 Use Case

Figure 3.5 is a Use Case view of the solution. Buyers and Sellers have different functions. The Sellers use the ERP to interact with the e-commerce platform via APIs including registering as a seller, adding and updating products, and publishing these products on e-commerce and social media platforms. The Buyer can use the e-commerce platform to view products, manage their cart and place orders.



Figure 3.5 Use Case Diagram

3.6 Database

Figure 3.6 below details the database for the e-commerce platform. It will have a category table populated by an API call from the ERP with the category details, a seller table that will contain details about the seller (user of the ERP) which will also be populated by an API call from the ERP. Finally, a products table containing each sellers products from the ERP.

To enable a complete purchase flow, there is a wish list table to store each Buyer's wish list, a cart table to enable users to manage the items in their cart, an order table to store the Buyer's orders and an order details table to store the details of each order.

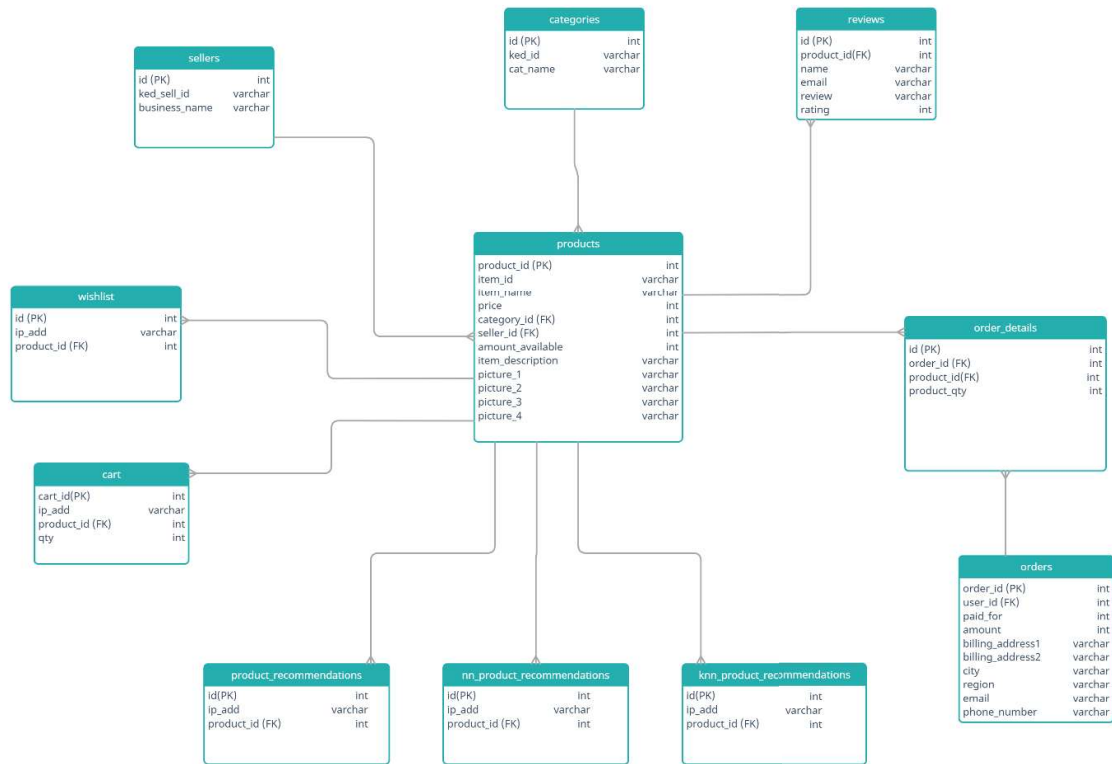


Figure 3.6 ER Diagram

3.7 REST APIs for Bi-Directional Communication

3.7.1 High level overview:

The APIs on both ends will allow the two systems to communicate with each other.

They will perform the following functions:

1. Send seller details to the Npontu Mall when a seller registers on the ERP.
2. Send category details to the Npontu Mall platform when a seller enters a new category.
3. Allow ERP users to publish their products on the Npontu Mall platform.
4. Allow ERP users to update products on the Npontu Mall platform.

5. Send order data to the ERP when a buyer places an order

The solution will use APIs on both e-commerce and ERP ends to communicate with each other that is transmitting product, customer and order data following. The APIs will be built with the Representational State Transfer (REST) architectural pattern. The key modules in this architecture are:

1. RESTful service layer: the module or layer will be responsible for the business logic of the API. For instance, when the seller wants to publish a new product to the e-commerce platform, it is the *ProductService* that will be responsible for running methods such as creating, updating and deleting products in the database of the Npontu Mall platform. The services on the Npontu Mall side are *ProductService*, *SellerService*, *CategoryService*, while the service on the ERP side is *OrderService* for storing order data.
2. Transport layer: this module will house the REST handler, which will accept HTTP requests (GET, POST, PUT/PATCH, DELETE) and map it to the appropriate method in the service layer for processing. For instance, if the ERP makes a POST request to the *ProductService* the REST handler will identify that it is a POST request and will direct it to the create method in the service for the product to be stored.
3. Middleware: this module will be responsible for auxiliary functions like authenticating an API request. This is to prevent everyone but the ERP or Npontu Mall platform from making requests to the other and the application's overall security

The figure 3.7 below summarises the REST architecture for the solution

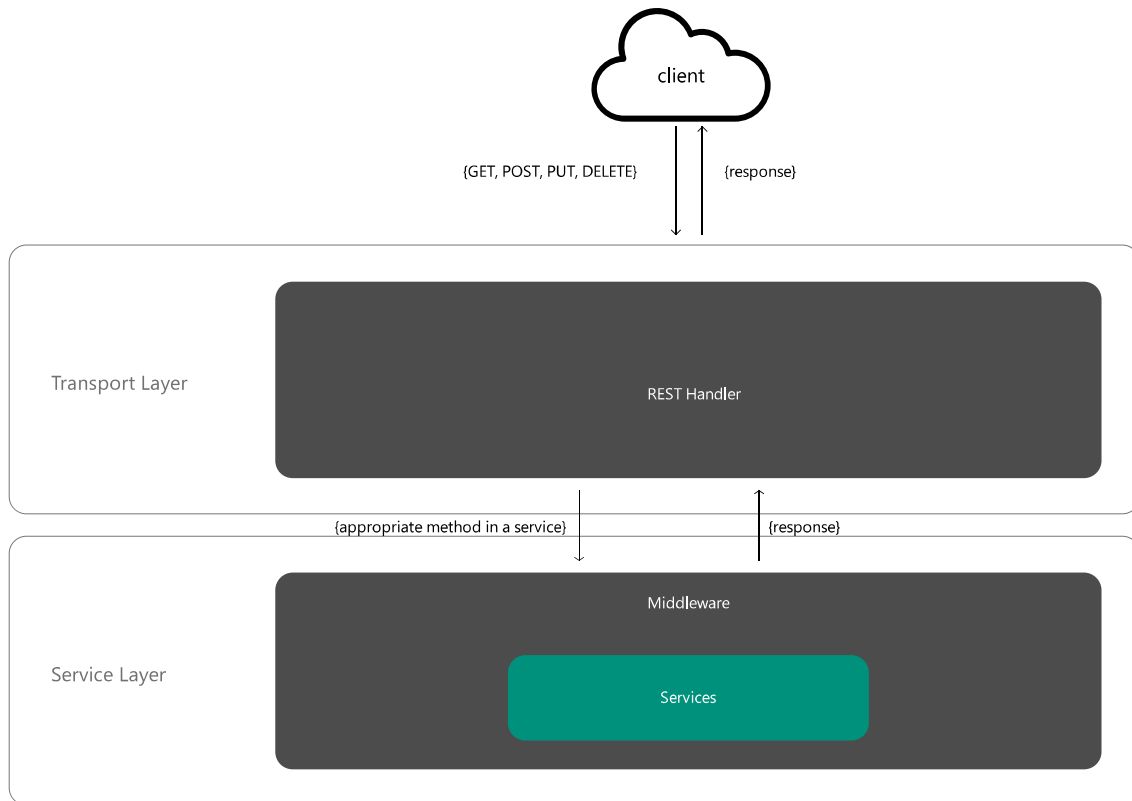


Figure 3.7 RESTful Architecture

This architecture supports the functional requirements by allowing the ERP to transmit and receive data from the e-commerce platform and vice versa. This lays the foundation for the data on both ends to be synchronised, thus ensuring the data is robust and the same at both ends at all times. The middleware also fulfils the security non-functional requirement by ensuring that only the authenticated request with an API key can be fulfilled.

Chapter 4: Implementation

4.1 Introduction

This chapter gives an overview of the implementation process of the solution. It involves all the tools, APIs, frameworks, components and modules and how they come together to build a functional prototype for solving the problem.

The solution to the problem comes in three components: a social media module for the ERP that will allow the business owner to advertise their inventory, an e-commerce platform where interested customers can place orders for products from sellers on the Kedebah ERP and finally, a module to allow Kedebah users to connect to their WooCommerce store if they have one.

4.2. Software Development Technique

The solution was built using the AGILE methodology for software development. This is a methodology that promotes continuous development and testing throughout the software development lifecycle of the project. Throughout the implementation phase, each component was built incrementally and tested concurrently to ensure it worked and did not result in any errors and wrong response codes.

4.3 Key Technologies

4.3.1 Languages

The following languages were used in the implementation of the solution. They have been broken down into two: frontend and backend programming languages.

4.3.1.1 Frontend Languages

The front end of an application is the user-facing end. It is where the user interacts mainly with the application and is responsible for getting user input and displaying output from the backend. The front end was built with HTML, CSS and JavaScript.

HTML: short for Hypertext Markup Language, is a scripting language used for structuring information on a web page. This information includes texts and images that are seen on the web page.

CSS: short for Cascading Style Sheets is another frontend language used for designing and beautifying the HTML elements of a page to make it readable and pleasing for the user.

JavaScript: this is the frontend language used for making the system interactive. Asynchronous JavaScript (AJAX) was used for interacting with backend functions such as adding to wish lists and cart management.

Despite these languages used in the development of the front end, the team at Npontu provided a responsive Bootstrap template called Shopwise [16], which was used to serve as the front end of Npontu Mall. Likewise, the front end for the test inventory system was also a responsive Bootstrap template [17]

4.3.1.2 Backend Languages

The backend of an application serves as the business logic. It is the module that handles and processes all the user requests. It is what makes the system functional. The backend of the application was written in PHP and Python.

PHP: short for Hypertext Preprocessor, is a server-side programming language. It is the primary language that the backend of the application was written in. It is mainly responsible for manipulating the data in the database. This includes inserting, fetching, updating and deleting data from the database.

Python: This language can also serve as the backend of any application, but it is more notable for its data analysis capabilities. The application makes use of Python for

analysing user reviews and making recommendations to the user of products they might like based on their reviews.

4.3.2 Frameworks and Libraries

Laravel: A programming language framework is a prebuilt platform for developing software applications. It includes predefined classes and functions that make developing applications easier since developers would not have to write them from scratch each time they build a new application. The framework used for this project is Laravel an open-source framework for PHP. Laravel comes with some packages that enabled the implementation of some features.

Twitter For PHP: With this library, the implementation of posting to social media was carried out successfully [14]. This enables users of the ERP to post and share their products on Twitter easily. Before the users can use this feature, they will be required to login to their Twitter account and grant the ERP permissions to use their account to make a post. This implementation uses Twitter's OAuth authorisation to authenticate the user.

Scikit-Learn: Scikit-learn is a machine learning library for Python. The Npontu Mall mainly uses Scikit-Learn to learn from user reviews on products and make recommendations for other products. The machine learning algorithm used here in the project was the K-nearest neighbour for recommending similar products for a target product.

PyTorch: PyTorch is an open-source machine learning library developed by Facebook for Python. It provides deep learning algorithms for creating neural networks in order to learn and make useful predictions from data. PyTorch was used in this project to create a deep learning model to learn from user interactions with products to make predictions on which products users would likely want to interact with.

4.4 Application Programming Interface (APIs)

Application Programming Interfaces are software intermediaries that allow different applications to communicate with each other and share data over the internet. This project makes use of the following APIs:

Twitter API: In order to allow ERP users to make posts to Twitter, an application needed to be created linked to a Twitter account which will grant the system access to the Twitter API. With the following credentials: Twitter Consumer Key, Consumer Secret, Access Token and Access Token Secret. By enabling OAuth authentication on the Twitter application, the feature was implemented allowing all ERP users to post products to their respective Twitter accounts.

Payment Data Exchange API by Npontu: This API by the Npontu team, affords the Npontu Mall platform the feature to allow buyers the option of paying for their purchases with their Mobile Money wallets, be it MTN Mobile Money, Vodafone Cash or AirtelTigo Cash.

Mailgun: The API provided by Mailgun allowed the implementation of sending transactional email to buyers to confirm that their order has been recorded.

Flutterwave Payment API: This API provided by Flutterwave allows the Npontu Mall platform the feature to enable buyers to pay with their Debit cards.

4.5 Evidence of Implementation

4.5.1 Ecommerce Platform

This section shows the Npontu Mall platform and walks through the purchase process from when a user visits the page up to when they complete a purchase.

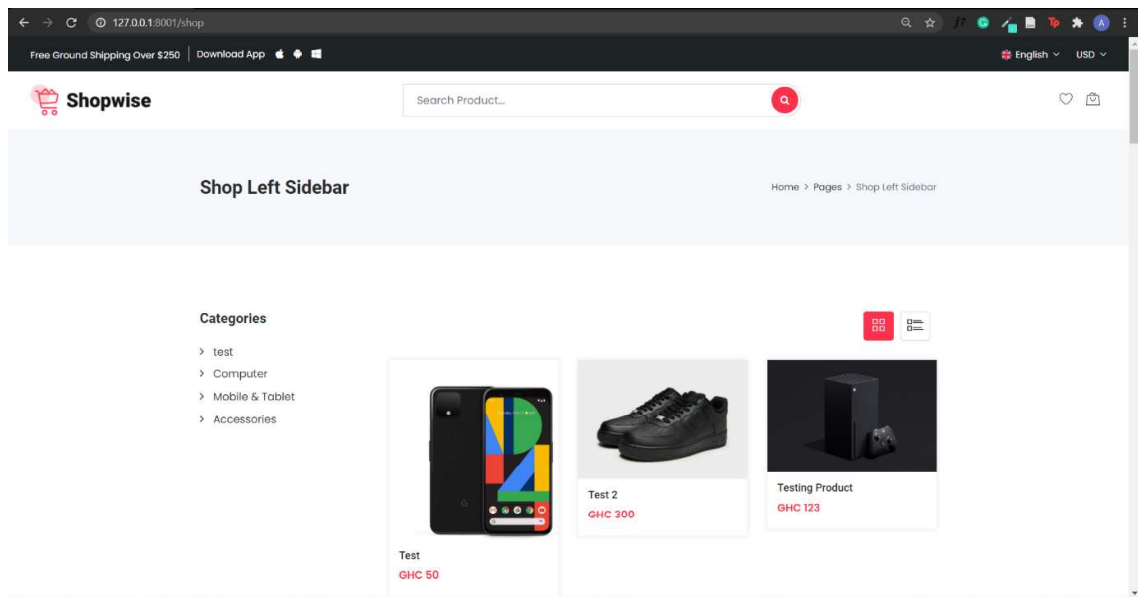


Figure 4.1: Shop page

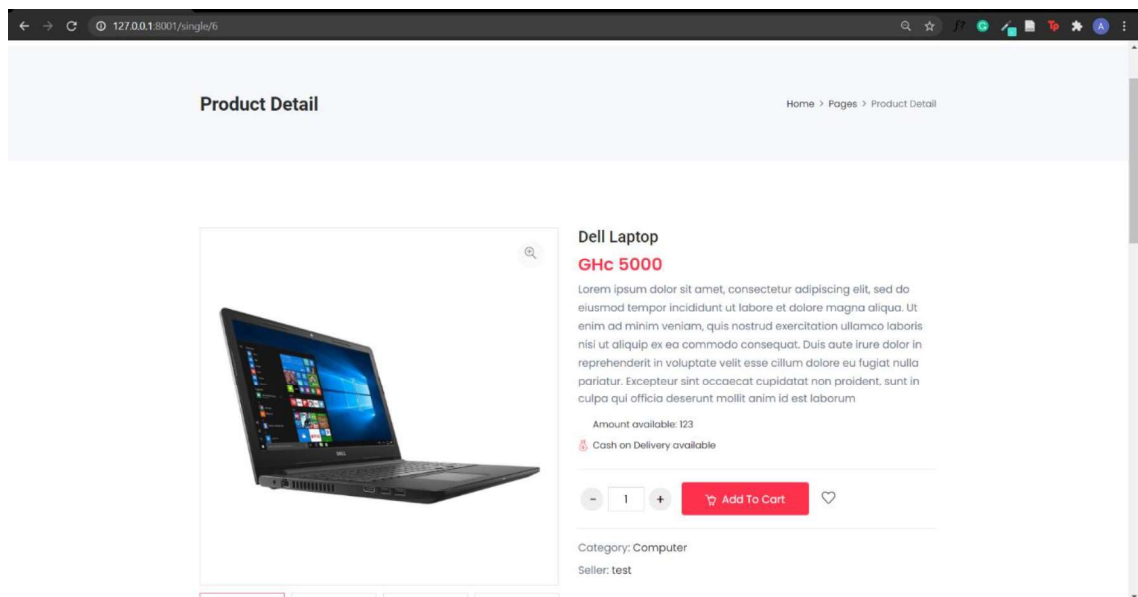


Figure 4.2: Single Product

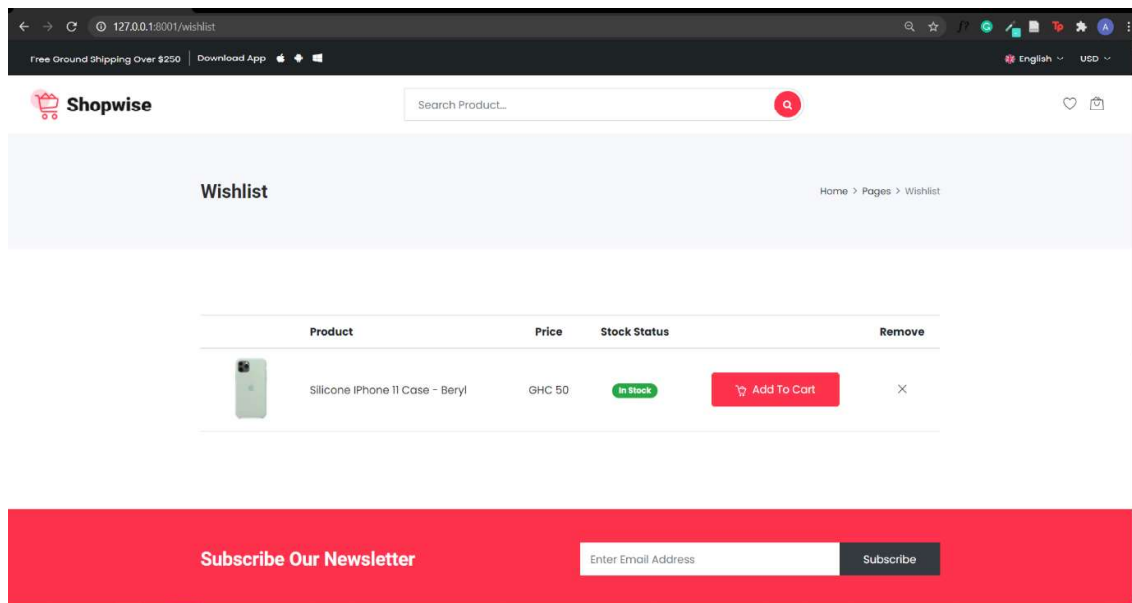


Figure 4.3 Wishlist Page

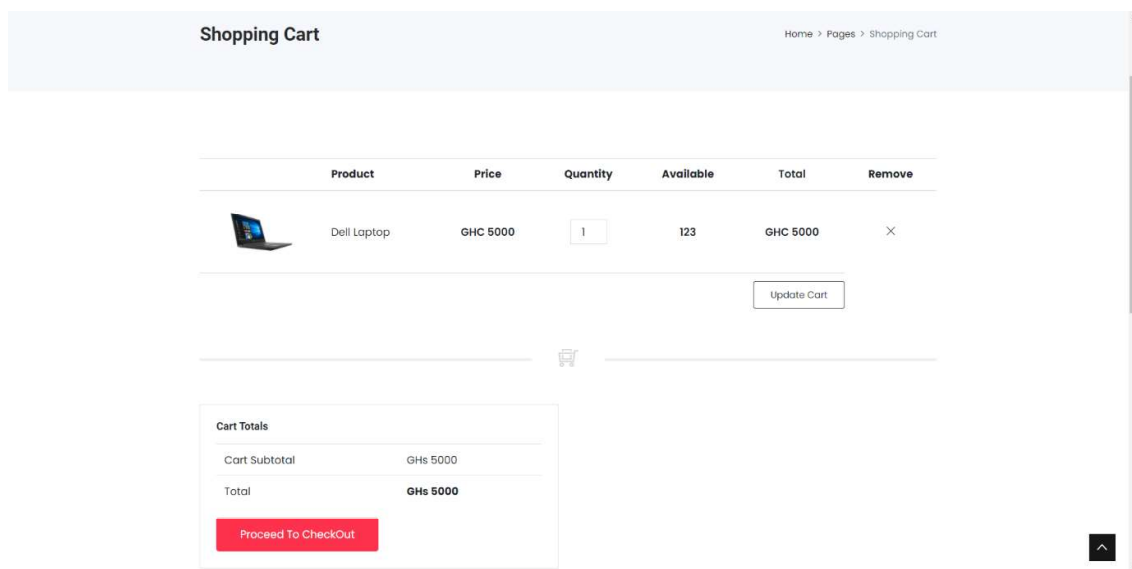


Figure 4.4 Cart Page

Billing Details

Allotei Pappoe

0557335284

allotei5@gmail.com

House Number 2, Shidaa Lane,

Adjiringanor

Accra

Greater Accra

Your Orders

Product	Total
Dell Laptop x 1	Ghc 5000
SubTotal	Ghc 5000
Shipping	TBD
Total	Ghc 5000

Payment

☐ MTN Mobile Money
Pay with your MTN Mobile Money Wallet

☐ Vodafone Cash

☒ Airtel Tigo

☐ Cash Payment

[Place Order](#)

Figure 4.5 Checkout Page

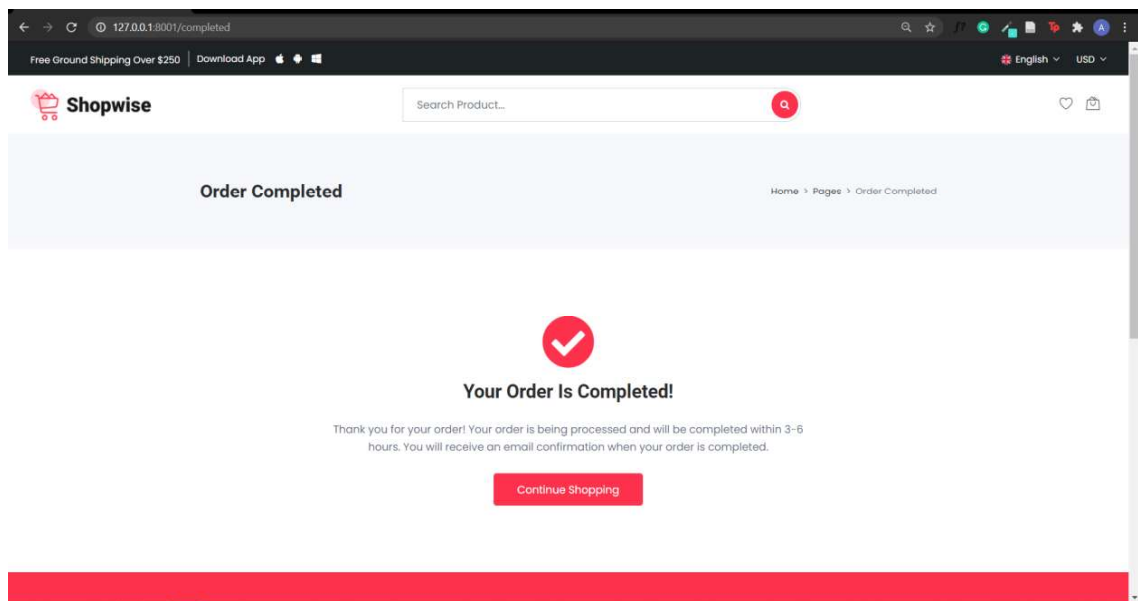


Figure 4.6: Order Completed Page

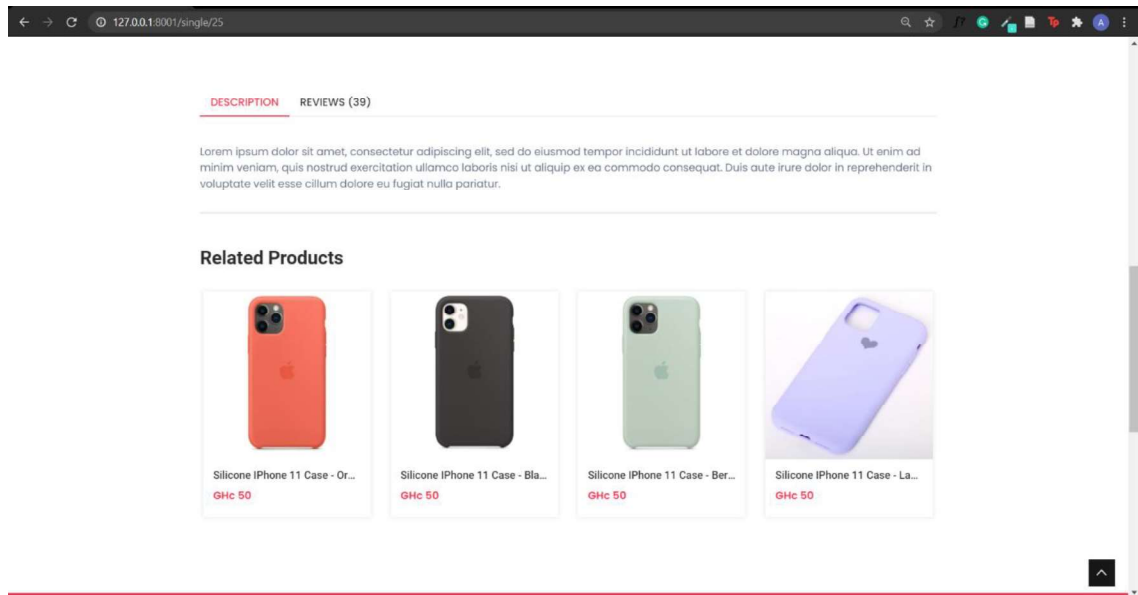


Figure 4.7: Related Products

```

public function getProductSortedBySimilarity(int $productId, array $matrix): array
{
    $similarities = $matrix['product_id_' . $productId] ?? null;
    $sortedProducts = [];

    if(is_null($similarities)){
        throw new Exception('Can\'t find product with that ID.');
```

```

    }
    arsort($similarities);

    foreach($similarities as $productIdKey => $similarity){
        $id = intval(str_replace('product_id_', '', $productIdKey));
        $products = array_filter($this->products, function ($product) use ($id) { return $product['id'] === $id; });
        if(!count($products)){
            continue;
        }
        $product = $products[array_keys($products)[0]];
        $product['similarity'] = $similarity;
        $sortedProducts[] = $product;
    }

    return $sortedProducts;
}

protected function calculateSimilarityScore($productA, $productB)
{
    return array_sum([
        (SimilarityController::euclidean(
            SimilarityController::minMaxNorm([$productA['price']], 0, $this->priceHighRange),
            SimilarityController::minMaxNorm([$productB['price']], 0, $this->priceHighRange)
        ) * $this->priceWeight),
        (SimilarityController::jaccard(strval($productA['categories_id']), strval($productB['categories_id'])) * $this->categoryWeight)
    ]) / ($this->priceWeight + $this->categoryWeight);
}

```

Figure 4.8: PHP Functions for related products

The figure above is a snippet of the PHP classes that go into getting the related products of a particular product on the platform. Just like the product recommendation, these functions and classes also use item-item based collaborative filtering to determine how

similar products are to each other based on the algorithms: Euclidean interval (for determining how close the prices are) and Jaccard similarity index (used to determine the similarity between sample sets in this case, product features: category and the seller). This code was based on the tutorial by Oliver Lundquist [8].

4.5.2. Product Recommendation

This section describes how the e-commerce platform makes product recommendations to buyers.

4.5.2.1 Collaborative Filtering

Collaborative Filtering is an approach to recommendation systems that collects feedback from users in the form of rating and makes recommendations based on them. It identifies customers with common ratings and offers suggestions based on the ratings. It assumes that people who like a product now will want the same items in the future [11].

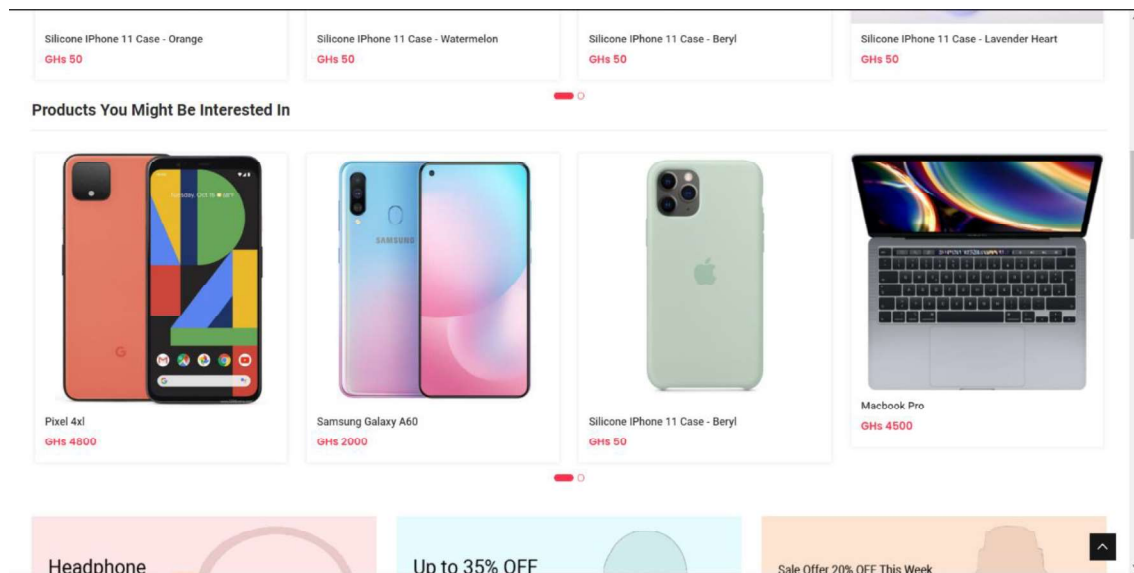


Figure 4.9: Product Recommendation Based On Users Reviews

```

def recommend_it(predictions_df, itm_df, original_ratings_df, num_recommendations=10, ruserId='127.0.0.1'):

    # Get and sort the user's predictions
    sorted_user_predictions = predictions_df.loc[ruserId].sort_values(ascending=False)

    # Get the user's data and merge in the item information.
    user_data = original_ratings_df[original_ratings_df.ip_add == ruserId]
    user_full = (user_data.merge(itm_df, how = 'left', left_on = 'product_id', right_on = 'product_id').
                sort_values(['rating'], ascending=False)
                )

    # Recommend the highest predicted rating items that the user hasn't bought yet.
    recommendations = (itm_df[~itm_df['product_id'].isin(user_full['product_id'])].
                    merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
                        left_on = 'product_id',
                        right_on = 'product_id').
                    rename(columns = {ruserId: 'Predictions'}).
                    sort_values('Predictions', ascending = False).
                    iloc[:num_recommendations, :-1]
                    )
    topk=recommendations.merge(original_ratings_df, right_on = 'product_id', left_on='product_id').drop_duplicates(
        ['product_id', 'item_name'])[['product_id', 'item_name']]

    return recommendations

```

Figure 4.10: Python Function for recommending products

The figure above is a snippet of the various functions that recommend products to users based on their reviews. Product recommendations are obtained by item-item collaborative filtering using the product feature: product reviews. The functions use Python's Sci-kit learn machine learning library to gain insight from reviews and recommend products to a buyer based on the reviews they have made. This code is based on the work by Uma Raju [7] and adapted to this project.

4.5.2.2 Collaborative Filtering Using K-Nearest Neighbors

The K-nearest neighbours (KNN) algorithm approach to collaborative filtering relies on product feature similarity (ratings in this case). To recommend a similar product, the algorithm will calculate the distance or similarity between the target product and all the other products in the database based on the distance metrics: Euclidean, Manhattan, and Minkowski [11].

Others Also Liked:

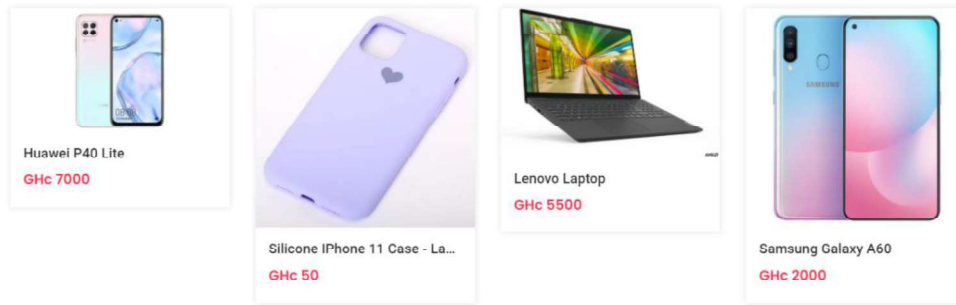


Figure 4.11: KNN Recommending Products based on Rating Similarity

```
from scipy.sparse import csr_matrix
#pivot ratings into product features
df_product_features=product_ratings.pivot_table(index='product_id',columns= 'ip_add',values='rating').fillna(0)
mat_product_features = csr_matrix(df_product_features.values)

df_product_features.head()

from sklearn.neighbors import NearestNeighbors

model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

#fit the dataset
model_knn.fit(mat_product_features)

def make_recommendation(model_knn, data, movie,first_num, n_recommendations):
    #fit model
    model_knn.fit(data)
    idx=process.extractOne(str(movie+first_num), df_products['product_id'])[2]
    #make inference
    distances, indices = model_knn.kneighbors(data[idx], n_neighbors=n_recommendations+1)
    raw_recommends = sorted(list(zip(indices.squeeze().tolist()+first_num, distances.squeeze().tolist())), key=lambda x: x[1])[0:-1]
    return raw_recommends
```

Figure 4.12: Implementation of KNN Algorithm

The figure above is a snippet of the implementation of the KNN algorithm for collaborative filtering. Given a target product, this piece of code will recommend similar products to a user when they view it. This code was based on the tutorial by Nikita Sharma [11].

4.5.2.3 Neural Network-Based Collaborative Filtering (NCF)

Neural network-based collaborative filtering is a general framework for neural architecture that can learn an arbitrary function from data. The aim of NCF is to learn from user interaction with products (implicit feedback) and make recommendations based on that.

It learns from the dataset with a deep learning neural network. This framework was proposed by He et al in their research paper [12].

```
class NCF(pl.LightningModule):
    """ Neural Collaborative Filtering (NCF)

    Args:
        num_users (int): Number of unique users
        num_items (int): Number of unique items
        ratings (pd.DataFrame): Dataframe containing the product ratings for training
        all_productIds (list): List containing all productIds (train + test)
    """

    def __init__(self, num_users, num_items, ratings, all_productIds):
        super().__init__()
        self.user_embedding = nn.Embedding(num_embeddings=num_users, embedding_dim=8)
        self.item_embedding = nn.Embedding(num_embeddings=num_items, embedding_dim=8)
        self.fc1 = nn.Linear(in_features=16, out_features=64)
        self.fc2 = nn.Linear(in_features=64, out_features=32)
        self.output = nn.Linear(in_features=32, out_features=1)
        self.ratings = ratings
        self.all_productIds = all_productIds

    def forward(self, user_input, item_input):

        # Pass through embedding layers
        user_embedded = self.user_embedding(user_input)
        item_embedded = self.item_embedding(item_input)

        # Concat the two embedding layers
        vector = torch.cat([user_embedded, item_embedded], dim=-1)

        # Pass through dense layer
        vector = nn.ReLU()(self.fc1(vector))
        vector = nn.ReLU()(self.fc2(vector))

        # Output layer
        pred = nn.Sigmoid()(self.output(vector))

        return pred

    def training_step(self, batch, batch_idx):
        user_input, item_input, labels = batch
        predicted_labels = self(user_input, item_input)
        loss = nn.BCELoss()(predicted_labels, labels.view(-1, 1).float())
        return loss
```

Figure 4.13: NCF Model Implementation

The figure above shows the implementation of the NCF model based on the tutorial by James Loy [13]. It trains a deep learning model based on the dataset of user interactions and then makes recommendations on the products a user will likely want to interact with. The figure below shows the model recommending products to a given user based on previous interactions with other products.

Recommendations Based On Your Recent Activity

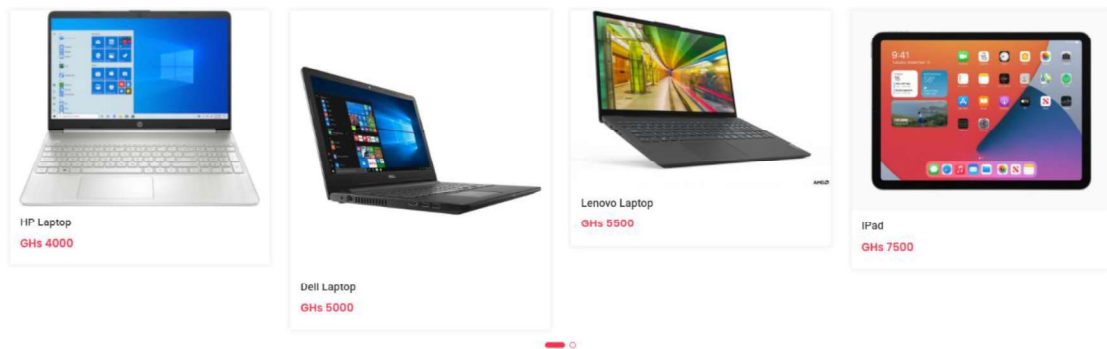


Figure 4.14: NCF Model in Action

4.5.3 Application Programming Interfaces for E-commerce platform

This section details how the ERP communicates with the ecommerce platform with the necessary data like seller, category, products, and orders.

4.5.3.1 Seller API

When a new user on the ERP is registered, they are also registered on the e-commerce platform as well. The ERP will make an API call to the e-commerce platform for the user to be registered there as a seller.

```

    */
    public function store(Request $request)
    {
        $verify = $this->ApiKeyController->verify($request->apikey);

        if($verify){
            $this->validate($request, [
                'ked_sell_id' => 'required',
                'business_name' => 'required',
            ]);

            Seller::create($request->all());
            return response(['created' => true], 201);
        }else{
            return ["message" => "User is not authenticated"];
        }
    }
}

```

Figure 4.15: Seller API

The figure above shows the store method in the seller API controller. This method is responsible for storing a new seller.

4.5.3.2 Category API

The Npontu Mall platform also has an API to receive categories from the ERP when a new category is created. The figure below shows the store method in the category API which stores new categories.

```

public function store(Request $request)
{
    //store a new category
    $verify = $this->ApiKeyController->verify($request->apikey);

    if($verify){
        $this->validate($request, [
            'ked_id' => 'required',
            'cat_name' => 'required',
        ]);

        return Category::create($request->all());
    }else{
        return ["message" => "user is not verified"];
    }
}

```

Figure 4.16: Category API

4.5.3.3 Products

Products on the platform are provided by the ERP as well. When the ERP user creates a new product, he or she has the option of publishing it to the e-commerce platform. When they do, they make an API call sending product parameters: id, name, stock, category, seller, pictures, price, currency and item description. This is received by the e-commerce platform and stored in a database. The product API also has functions for updating (mainly for stock) and deleting products. The figure below is a snippet of the store function in the product API controller.

```

Product::create([
    'item_id' => $request->item_id,
    'item_name' => $request->item_name,
    'amount_available' => $request->amount_available,
    'categories_id' => $cat_id[0]->id,
    'sellers_id' => $sell_id[0]->id,
    'picture_1' => $request->picture_1,
    'picture_2' => $request->picture_2,
    'picture_3' => $request->picture_3,
    'picture_4' => $request->picture_4,
    'price' => $request->price,
    'currency' => $request->currency,
    'item_description' => $request->item_description,
]);

$response = "Product has been added successfully";
return [
    "message" => $response,
];
}else{
    return ["message" => "user is not verified"];
}

```

Figure 4.17: Product API

4.5.3.4 Orders

When an order is completed, the Npontu Mall platform stores it in its database and makes an API call to the ERP sending the order details. These details include: an apikey, the customer name, customer number, customer email address, the amount paid, whether the purchase has been paid for or not, the billing address details, and a list of all the products being purchased. The figure below shows the API call for sending order data to the ERP. When the ERP successfully processes the request, it sends a response of new stock of the products bought which is then updated in the database of the Npontu Mall platform.

```

//api call to ERP
$response = Http::post('http://127.0.0.1:8000/api/order', [
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',
    'buyer_name' => $request->fname,
    'buyer_number' => $request->phone,
    'email_add' => $request->email,
    'amount' => $request->amount,
    'paid_for' => $request->paid_for,
    'billing_address' => $request->billing_address,
    'billing_address2' => $request->billing_address2,
    'city' => $request->city,
    'region' => $request->region,
    'order_details' => $carts->toArray(),
]);

foreach($response->json()['response'] as $resp){
    Product::find($resp['product_id']->update([
        'amount_available' => $resp['qty'],
    ]);
}

```

Figure 4.20: Order API request

4.5.3.5 Connecting To Woo-commerce

This section details how connecting the ERP to Woo-commerce requirement was implemented. For ERP users to connect their Woo-commerce store to their ERP account, they will have to provide some parameters from the Woo-commerce installation. These parameters include: the website URL, consumer key and consumer secret. When the user provides these details, they will be encrypted and stored in the database of the ERP. These parameters will be used to connect to the Woo-commerce store.

Woocommerce

Woocommerce

⊞ Connect to your Woocommerce Store

Website URL (https://website-url.com)

Enter your stores URL

Woocommerce Consumer Key

Enter your stores consumer key

Woocommerce Consumer Secret

Enter your stores consumer secret

☐ Confirm if webhook has been created

Connect to Store

Figure 4.19: Connecting to Woo-commerce Store Front View

```

public function connect_to_store(Request $request)
{
    $this->validate($request, [
        'website_url' => 'required',
        'consumer_key' => 'required',
        'consumer_secret' => 'required',
        'webhook' => 'required',
    ]);

    $woocommerce = new Client(
        $request->website_url,
        $request->consumer_key,
        $request->consumer_secret,
        [
            'version' => 'wc/v3',
        ]
    );

    //validate params with woocommerce
    try{
        $data = [
            'name' => 'Order Created',
            'topic' => 'order.created',
            'delivery_url' => 'https://alloteip1.sg-host.com/woocommerce-order'
        ];

        $validate_connection = $woocommerce->post('webhooks', $data);
    }catch(HttpClientException $e){
        $err_message = $e->getMessage();

        return back()->with('err_status', $err_message);
    }

    Woocommerce::create([
        'user_id' => $request->user()->id,
        'website_url' => $request->website_url,
        'consumer_key' => Crypt::encryptString($request->consumer_key),
        'consumer_secret' => Crypt::encryptString($request->consumer_secret),
    ]);
}

```

Figure 4.20: Backend for connecting to user's Woo-commerce Store

```

public function constructor()
{
    $woo_credentials = Woocommerce::where('user_id', '=', auth()->user()->id)->get();
    $consumer_key = Crypt::decryptString($woo_credentials[0]->consumer_key);
    $consumer_secret = Crypt::decryptString($woo_credentials[0]->consumer_secret);
    $woocommerce = new Client(
        $woo_credentials[0]->website_url,
        $consumer_key,
        $consumer_secret,
        [
            'version' => 'wc/v3',
        ]
    );

    return $woocommerce;
}

```

Figure 4.21: Snippet of Code for connecting to Woo-commerce store

The above figure shows the snippet code which is used to connect to the user's Woo-commerce store whenever a request (posting or updating a product) is to be made.

4.5.4 Social Media Posts

This subsection details how users can make posts on Twitter to advertise their products from the ERP. This section shows how the Twitter module for the solution was implemented. As stated earlier this was made possible by the Twitter API and the Twitter for PHP library [14] for Laravel. Firstly, powered by Twitter's OAuth services, the ERP user has to login into their Twitter account and grant access to the Twitter application to make posts on their behalf. Twitter then makes a callback with Access Tokens which can be used to make posts on the user's account. After the user has successfully given the application access to their account, they can now make posts about their products.

4.5.1 Evidence of Implementation

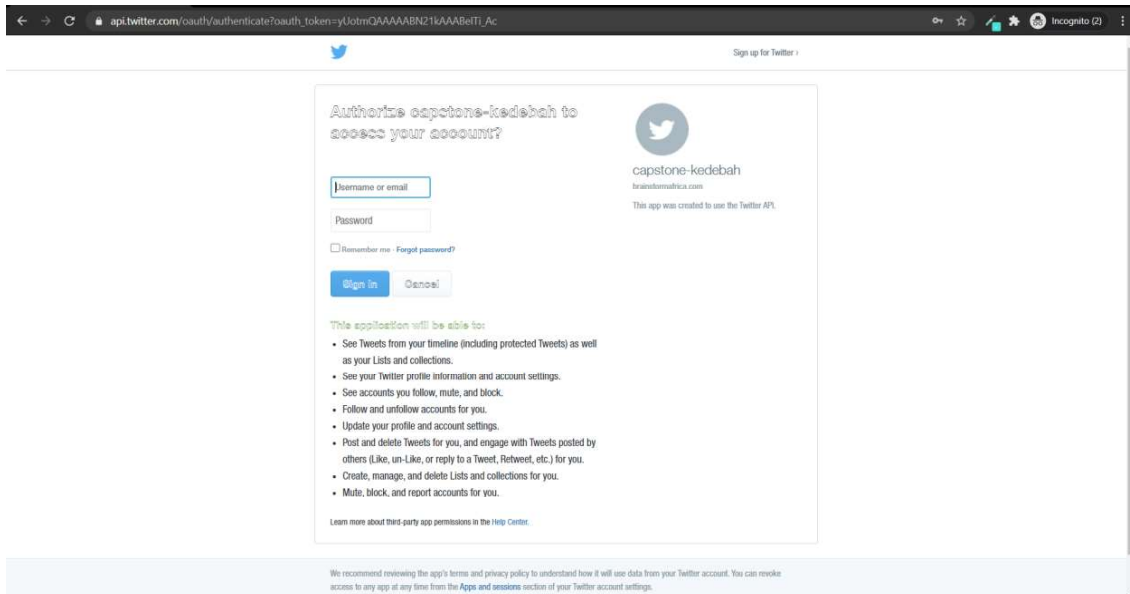


Figure 4.22: Logging in to the Twitter Account

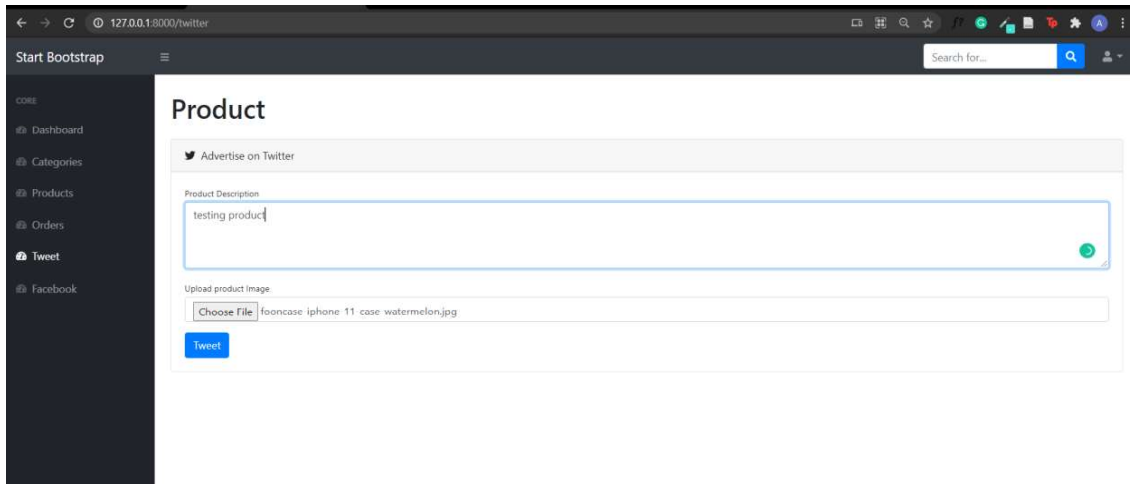


Figure 4.23: Tweeting about a product



allotei
@_allotei_



testing product



8:51 PM · Mar 30, 2021 · capstone-kedebah

||| [View Tweet activity](#)

Figure 4.24: Post made successfully.

Backend implementation

```
Route::get('twitter/login', ['as' => 'twitter.login', function(){
    // your SIGN IN WITH TWITTER button should point to this route
    $sign_in_twitter = true;
    $force_login = false;

    // Make sure we make this request w/o tokens, overwrite the default values in case of login.
    Twitter::reconfig(['token' => '', 'secret' => '']);
    $token = Twitter::getRequestToken(route('twitter.callback'));

    if (isset($token['oauth_token_secret']))
    {
        $url = Twitter::getAuthorizeURL($token, $sign_in_twitter, $force_login);

        Session::put('oauth_state', 'start');
        Session::put('oauth_request_token', $token['oauth_token']);
        Session::put('oauth_request_token_secret', $token['oauth_token_secret']);

        return Redirect::to($url);
    }

    return Redirect::route('twitter.error');
}));
```

Figure 4.25: Twitter OAuth Login

```
Route::get('twitter/callback', ['as' => 'twitter.callback', function() {
    if (Session::has('oauth_request_token'))
    {
        $request_token = [
            'token' => Session::get('oauth_request_token'),
            'secret' => Session::get('oauth_request_token_secret'),
        ];

        Twitter::reconfig($request_token);

        $oauth_verifier = false;

        if (Request::has('oauth_verifier'))
        {
            $oauth_verifier = Request::get('oauth_verifier');
        }

        // getAccessToken() will reset the token for you
        $token = Twitter::getAccessToken($oauth_verifier);

        if (!isset($token['oauth_token_secret']))
        {
            return Redirect::route('twitter.login')->with('flash_error', 'We could not log you in on Twitter.');
        }

        $credentials = Twitter::getCredentials();

        if (is_object($credentials) && !isset($credentials->error))
        {
            Session::put('access_token', $token);

            return Redirect::to('/twitter')->with('flash_notice', 'Congrats! You\'ve successfully signed in!');
        }

        return Redirect::route('twitter.error')->with('flash_error', 'Crab! Something went wrong while signing you up!');
    }
}]);
```

Figure 4.26: Twitter Callback

```
<?php

namespace App\Http\Controllers\SocialMedia;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Thujohn\Twitter\Facades\Twitter;
use Illuminate\Support\Facades\File;
use Illuminate\Support\Facades\Storage;

class TwitterController extends Controller
{
    public function tweet(Request $request)
    {
        //return Twitter::postTweet(['status' => $request->status, 'format' => 'json']);
        $this->validate($request, [
            'status' => 'required|max:280',
            'image' => 'required',
        ]);
        //dd($request->file('image'));

        //upload image
        $filenameWithExt = $request->file('image')->getClientOriginalName();
        $filename = pathinfo($filenameWithExt, PATHINFO_FILENAME);
        $extension = $request->file('image')->getClientOriginalExtension();
        $fileNameToStore = $filename.'_'.time().'.'.$extension;
        $path = $request->file('image')->storeas('public/storage', $fileNameToStore);

        $uploaded_media = Twitter::uploadMedia(['media' => Storage::disk('local')->get('public/storage/'.$fileNameToStore)]);
        $tweet = Twitter::postTweet(['status' => $request->status, 'media_ids' => $uploaded_media->media_id_string, 'format' => 'json']);
        return back()->with('status', 'You tweeted successfully');
    }
}
```

Figure 4.27: Posting a Tweet with an Image

Chapter 5: Testing and Results

5.1 Introduction

This chapter details the testing tools and techniques used to appraise the system. Testing is a major part of the software development life cycle that involves ensuring that the built system meets the requirements set, solves adequately the problem for which it was created, and if the users are happy with it in general. The testing was carried out to ensure that the solution adequately meets the requirements stated in Chapter 2. Among several kinds of testing that can be done on a system, including unit testing, regression testing, user acceptance testing, component testing, and system testing, only four testing forms were carried out: unit testing, user acceptance testing, component testing, and system testing. Unit testing is carried out to ensure that the various units of the system work well and cater for several cases. Component testing is done to ensure that the various components which are amalgamations of units also work well. System testing is carried out to ensure that the system as a whole meets all the requirements. User acceptance testing is done to check the usability of the system and how well it will be received by the average user.

5.2 User Acceptance testing

Acceptance testing is the phase in software testing used to assess whether or not the software is market-ready, complies with all requirements and meets the end-user's needs. The system was tested using both Alpha (internally with the Npontu engineering team) and Beta (external) tests.

With the Alpha tests, it was recommended that to make the Npontu Mall work better; categories should not be hardcoded into the frontend but otherwise should read from the database. Through the Alpha test, it was also decided that the customer should not login before buying so as to make the purchase flow less cumbersome.

Beta Testing: Twenty (20) users were picked at random to test the application ascertain its usability and for any features they would want in the Npontu Mall have and fill a Google form to record their responses.

Rate your experience with the website where 1 is poor and 5 is excellent.

20 responses

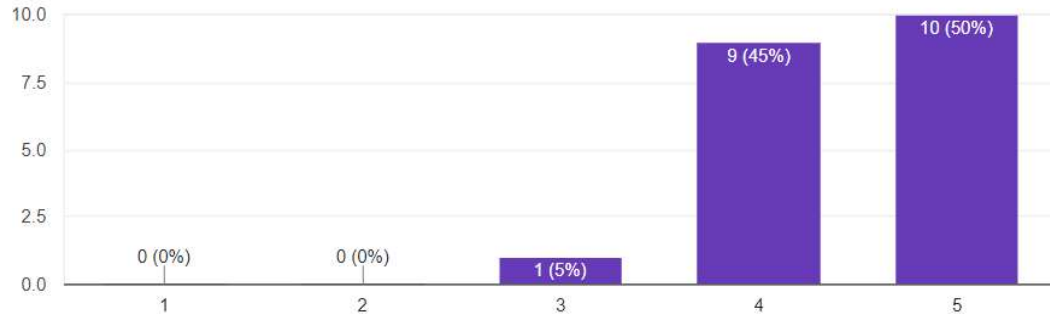


Figure 5.1: Beta testing responses

From the testing, all the beta testers found the user experience acceptable with 95% finding it very good, and the remainder found it average. The users were also asked what kind of features they would like to see on the application. Notable amongst the responses are:

1. A delivery page to keep track of the orders during delivery.
2. Live chat with the various sellers on the platform.
3. Sorting products by price and newness.
4. Login features.

Of these recommendations, only product sorting has been implemented post the testing phase.

5.3 Unit testing

To ensure the solution meets quality standards before deployment unit testing was carried out. This was done mainly to understand where the code breaks when there has been a change and to ensure that the system executes as per expectation. The following unit tests were carried out with PHPUnit testing: Route testing to ensure all the views work without fault, testing the product and seller API end-points.

```
public function test_home()
{
    $response = $this->get("/home");
    $response->assertStatus(200);
    //$this->assertTrue(true);
}

public function test_shop()
{
    $response = $this->get("/shop");
    $response->assertStatus(200);
}

public function test_cart()
{
    $response = $this->get("/cart");
    $response->assertStatus(200);
}

public function test_checkout()
{
    $response = $this->get("/checkout");
    $response->assertStatus(200);
}

public function test_wishlist()
{
    $response = $this->get("/wishlist");
    $response->assertStatus(200);
}
```

Figure 5.2: Route Test

```

class ProductTest extends TestCase
{
    //Testing product API

    public function test_api_key()
    {
        $this->json('POST', 'api/products', ['Accept' => 'application/json'])
        ->assertStatus(200)
        ->assertJson([
            "message" => "Invalid Key"
        ]);
    }

    public function test_fields_for_create_product()
    {
        $data = [
            'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',
        ];
        $this->json('POST', 'api/products', $data, ['Accept' => 'application/json'])
        ->assertStatus(422);
    }

    public function test_succesfull_creation()
    {
        ...
    }

    public function test_succesfull_update()
    {
        $data = [
            'amount_available' => 150,
        ];

        $this->json('PATCH', 'api/products/33', $data, ['Accept' => 'application/json'])
        ->assertStatus(200);
    }
}

```

Figure 5.3: Product API Test

```

class SellerApiTest extends TestCase
{
    public function test_create()
    {
        $user_data = [
            "apikey" => "d5f85e4af0a7e7b4dc0d75fae74c22c7",
            "ked_sell_id" => "2",
            "business_name" => "essentials",
        ];

        $response = $this->json('POST', 'api/seller', $user_data, ['Accept' => 'application/json']);

        $response -> assertStatus(201)->assertJson(['created' => true]);
    }
}

```

Figure 5.4: Seller API Test


```
Allotei@LAPTOP-64MQE10U MINGW64 /c/xampp/htdocs/kedebah-ecommerce (pappoe)
$ vendor/bin/phpunit
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

.....                                     12 / 12 (100%)

Time: 00:09.750, Memory: 40.00 MB

OK (12 tests, 14 assertions)
```

Figure 5.5: Unit Testing Results

The unit testing results show that all the test cases for routing work and return the appropriate HTTP response codes.

5.4 Component Testing

The testing will be carried out based on the key features to test the system's functional requirement satisfaction. Thus, the system was tested in parts according to the functional requirements.

1. Storing sellers from the ERP testing

The ERP is required to make an API call to the Npontu Mall system whenever a new user is registered and also to be stored in the platform database. This testing is done to ensure the ERP supplies the right API key as well as parameters.

Table 5.1 Storing sellers from ERP Testing

Valid Input	Response
Seller Name, API key, and ID in the ERP	Notification on storing success.
Invalid Input	Response
No inputs, eg API key, seller name, seller id	Notification on required input

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $verify = $this->ApiKeyController->verify($request->apikey);

    if($verify){
        $this->validate($request, [
            'ked_sell_id' => 'required',
            'business_name' => 'required',
        ]);

        return Seller::create($request->all());
    }else{
        return ["message" => "User is not authenticated"];
    }
}

```

Figure 5.6: Method to store a new seller

2. Storing categories from ERP test

The ERP is required to make an API call to the Npontu Mall system whenever a new category is also stored in the e-commerce platform database. This testing is done to ensure the ERP supplies the correct API key as well as parameters (id and category name)

Table 5.2: Storing categories from ERP testing

Valid Input	Response
API key, category id, category name	Category is successfully stored
Invalid Input	Response
If any of the parameters are missing	Notification on required input

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    //store a new category
    $verify = $this->ApiKeyController->verify($request->apikey);

    if($verify){
        $this->validate($request, [
            'ked_id' => 'required',
            'cat_name' => 'required',
        ]);

        return Category::create($request->all());
    }else{
        return ["message" => "user is not verified"];
    }
}

```

Figure 5.7 Method to store new category.

3. Managing products testing

The ERP is required to make an API call to the e-commerce system whenever a user manages (creates, updates and deletes) his or her products. This testing is done to ensure the user passes the correct parameters to the e-commerce page.

Table 5.3: Managing products testing

Valid Input	Response
API key, product name, product id, price, category, seller name, description, pictures	Product is successfully recorded
Invalid Input	Response
Any of the parameters are missing	Notification on required parameter

```

public function store(Request $request)
{
    $verify = $this->ApiKeyController->verify($request->apikey);

    if($verify){
        $this->validate($request, [
            'item_id' => 'required|unique:App\Models\Product,item_id',
            'item_name' => 'required',
            'amount_available' => 'required|integer',
            'sellers_id' => 'required',
            'picture_1' => 'required',
            'picture_2' => 'required',
            'picture_3' => 'required',
            'picture_4' => 'required',
            'price' => 'required|integer',
            'currency' => 'required',
            'categories_id' => 'required|integer',
            'item_description' => 'required',
        ]);
    }
}

```

Figure 5.8: Code to test if parameters have been met

```

Product::create([
    'item_id' => $request->item_id,
    'item_name' => $request->item_name,
    'amount_available' => $request->amount_available,
    'categories_id' => $cat_id[0]->id,
    'sellers_id' => $sell_id[0]->id,
    'picture_1' => $request->picture_1,
    'picture_2' => $request->picture_2,
    'picture_3' => $request->picture_3,
    'picture_4' => $request->picture_4,
    'price' => $request->price,
    'currency' => $request->currency,
    'item_description' => $request->item_description,
]);

$response = "Product has been added successfully";
return [
    "message" => $response,
];
}else{
    return ["message" => "user is not verified"];
}
}

```

Figure 5.9 Code for storing product if parameters are met

4. Wish list testing

The valid input needed to add a product to the wish list is a button click. If the Buyer does not click on the button, the function will not run. On a successful button, click the product is added to the buyers wish list. Thereafter, the user can also remove the product from the wish list or move the product to their cart.

```
public function store(Request $request){  
  
    $in_wishlist = Wishlist::where('product_id', '=', $request->product_id)->where('ip_add', '=', $request->ip())->get();  
  
    if(!$in_wishlist->count()){  
        Wishlist::create([  
            'product_id' => $request->product_id,  
            'ip_add' => $request->ip(),  
        ]);  
        return 'product has been added to your wishlists';  
    }else{  
        return 'product is already in your wishlist';  
    }  
}
```

Figure 5.10 Code for add product to wish list

```
public function add_from_wishlist(Request $request)  
{  
    $in_cart = Cart::where('product_id', '=', $request->product_id)  
        ->where('ip_add', '=', $request->ip())  
        ->get();  
    if(!$in_cart->count()){  
        Cart::create([  
            'product_id' => $request->product_id,  
            'qty' => 1,  
            'ip_add' => $request->ip(),  
        ]);  
        Wishlist::find($request->item_id)->delete();  
        return 'product has been added to cart';  
    }else{  
        Wishlist::find($request->item_id)->delete();  
        return 'product has been added to cart';  
    }  
}
```

Figure 5.11 Code for moving product from wish list to cart

```

public function destroy($id)
{
    $deleted = Wishlist::find($id)->delete();
    return $id;
}

```

Figure 5.12 Code for removing product from wish list

5. Cart management testing

The valid input needed to add a product to the cart is a button click. If the Buyer does not click on the button, the function will not run. On a successful button click the product is added to the Buyer's cart. On the cart page, the user can also remove the product from his cart or update it. Removing it requires just a button click and for when the user wants to update the cart quantity, all is needed is for the user to enter the quantity and click on the update button.

```

public function store(Request $request)
{
    $in_cart = Cart::where('product_id', '=', $request->product_id)
                    ->where('ip_add', '=', $request->ip())
                    ->get();

    if(!$in_cart->count()){
        //add to cart
        Cart::create([
            'ip_add' => $request->ip(),
            'qty' => $request->qty,
            'product_id' => $request->product_id,
        ]);

        return 'Product has been added to cart';
    }else{
        //if product is already in cart
        return 'Product is already in cart. Please consider increasing the quantity in your cart';
    }
}
}

```

Figure 5.13: Code to Add to cart

```

public function update(Request $request, $id){
    $cart_item = Cart::find($id);
    $cart_item->update([
        'qty' => $request->qty,
    ]);

    return "your product has been update";
}

```

Figure 5.14: Code to update cart

```

public function destroy($id)
{
    Cart::find($id)->delete();
    return "Item has been removed from your cart";
}

```

Figure 5.15: Code to remove item from cart

6. Checkout test

The valid inputs needed to checkout on the Npontu Mall platform are:

1. The Buyer must have at least one product in their cart.
2. The Buyer must click on the checkout button.
3. The Buyer must choose the preferred mode of payment.
4. The Buyer must provide their shipping details.

5.5 System Testing

This test is carried out to ensure that the several components of the system come together to work as an efficient system. Four techniques were used for this namely: usability testing, functional testing, security testing, and maintainability testing.

1. **Usability testing:** This test is carried out to identify how user-friendly the system is. If not, how it can be made easier to fulfill the non-functional requirement of usability. It is pertinent to do this because since buyers and sellers will use the system

mostly, it must be ensured they can use the application easily. There are navigation headers in the application to allow the users navigate their way around the application easily. In addition, the system stores the IP address of the current user to identify them, such that when they add to their cart or the wish list, the system can still retain and identify that it belongs to the particular user.

2. **Functional testing:** This test is carried out to ensure that the functionalities of the system are working and if they meet the requirements detailed in Chapter 2.
3. **Security testing:** The solution was deployed on a server with SSL installed. This encrypts the user's communication with the system and prevents malicious third parties from spying on the user's actions on the Npontu Mall especially when the user is attempting to make payments.
4. **Maintainability testing:** The solution was built based on the Model View Controller architectural pattern. This was done mainly because the architecture separates the key modules into different parts allowing easy maintenance by the main developer and the team at Npontu responsible for maintaining the application. The Laravel framework was also chosen because of maintainability because it is the primary framework utilised by the Npontu team; hence, it will be easy for them to maintain it.

Chapter 6: Conclusion and Recommendation

6.1 Conclusion

This project sought to develop a holistic solution as a means of helping ERP users expose their inventory to the online market through social media and on e-commerce platforms. Using the Kedebah ERP provided by Npontu Technologies as a case study, some requirements gathering was done with the Npontu team to ascertain the requirements specifications that will comprise the solution. The solution implemented comprises of the Npontu Mall which is integrated into the Kedebah ERP that will allow sellers to publish their products for buyers to purchase, integrating the ERP into social media (Twitter) to allow sellers to advertise their products, and an integration into Woo-commerce that will allow sellers to publish to their independent Woo-commerce stores.

The relevance of this project can be ascertained by how many orders ERP users receive through connecting their ERP accounts to the e-commerce platform. This will show how vital and relevant e-commerce is and how successful it can be implemented in the Ghanaian context.

6.2 Limitations

Through the requirement elicitation process, the Npontu team wanted to connect the Kedebah ERP to several e-commerce platforms namely: Amazon, Alibaba, Tonaton, Jiji and the e-commerce store the researcher built. Unfortunately, though Amazon and Alibaba have APIs that support connecting ERPs to them, those APIs are not available to Ghana. As a result, the researcher could not build a module to connect Kedebah. This limits the capacity to expose the inventory goods to a wider market because Amazon and Alibaba operate on a global scale. Tonaton and Jiji do not provide APIs for using their platforms externally; the only way to sell on their platforms is through their website.

Another requirement was to enable the ERP users connect to their social media accounts and advertise their products there via the ERP. The social media applications specified were: Twitter, Facebook and Google My Business. For Facebook, though the API is available to use, it is highly restricted by permissions. Some crucial aspects of the Facebook Graph API were not available for the researcher to use, notably the API end-point for making Facebook posts. Facebook states that to use that end-point, your account needs to be verified by them as a business account. A similar issue was encountered with Google My Business. To gain access to the API, the researcher needed to be verified as a legitimate business first. This also limits the capacity to expose the inventory to a wider market because of the number of users Facebook and Google has. However, this can be easily remedied when the ERP users manually post and advertise their products on their accounts.

Another limitation was the use of Mailgun to send emails. The Mailgun service requires users to have active domains for sending emails. Since this project does not have a domain it was a challenge sending emails with the service. In future works when the project is deployed on a web server with an active domain Mailgun can be effectively implemented.

6.3 Recommendations

This section describes some additional improvements and features that can be added to the system to make it better.

1. **User Authentication:** The system currently uses IPs of customers to identify them.

Though IPs are unique and can serve as an identifier for distinct customers, user authentication features, namely: registration and login can be implemented to identify customers uniquely. Using IPs has a pitfall in that when users on the same local network use the system they will be identified as one since their public IP will be the same. When different users are identified as one, it can lead to different issues

like one user adding a product to another user's cart or the recommendation system recommending products meant for one user to another. Though the decision for using IPs to identify customers was meant to smoothen the purchase process, Laravel provides the Socialite package which enables a Laravel application to authenticate users through Single Sign-On (SSO) with their Google, Twitter or Facebook accounts. This form of authentication is less cumbersome and will allow users to easily login to the system.

2. Live chat with sellers: On the application, when users have questions about a product there is no option for them to ask the seller. This can be a huge deal breaker and drive customers from making purchases if they cannot have their questions answered. Implementing a live chat feature will permit users to contact with the sellers and have any questions they might have answered.
3. Live delivery: A live delivery feature can be implemented on the application allowing users to monitor and track their products being delivered. This is to reaffirm the customers' belief that their order is being fulfilled.
4. Data for Neural Networks for product recommendation: The system recommends products to users using Neural Collaborative Filtering. This approach to product recommendation utilises implicit feedback, which indirectly reflects user preference through behaviours such as purchases and viewing a product [15]. The Model implemented in this system uses user reviews as an indicator for user interaction with the product. The system keeps track of which product pages the user visits as well as their purchases. Further work on improving the Model will involve feeding this data to the NCF model so it can make accurate recommendations.

References

[1] Eram Abbasi, Abdul Wasay Farooqui, Muhammad Faizan Batra, Muhammad Amin Rehmania and Syed Muhammad Anas. 2017. Bridging the Gap between ERP Applications and eCommerce Solutions. In *International Journal of E-Education, e-Business, e-Management and e-Learning*, 111-122.

<https://doi.org/10.17706/ijeec.2017.7.2.111-122>

[2] Jumia Group | Jumia Expand Your Horizons. Retrieved November 18, 2020 from

<https://group.jumia.com/>

[3] 7 Benefits of Ecommerce ERP Integration. Retrieved March 10, 2021, from

<https://www.comalytics.com/7-benefits-e-commerce-erp-integration/>

[4] nChannel. Retrieved March 10, 2021, from <https://www.nchannel.com/overview/>

[5] Harris Webworks. Retrieved March 10, 2021,

<https://www.harriswebworks.com/netsuite-magento-connector/>

[6] Ebridge Connections. Retrieved March 10, 2021,

<https://www.ebridgeconnections.com/home.aspx>

[7] Uma Maheswari Raju. 2020. Sentiment Analysis and Product Recommendation on Amazon's Electronics Dataset Reviews - Part 2. Retrieved March 30, 2021 from

<https://towardsdatascience.com/sentiment-analysis-and-product-recommendation-on-amazons-electronics-dataset-reviews-part-2-de71649de42b>

[8] Oliver Lundquist. 2019. Building a Product Recommender System with Machine Learning in Laravel. Retrieved March 30, 2021 from

<https://oliverlundquist.com/2019/03/11/recommender-system-with-ml-in-laravel.html>

- [9] Ministry of Trade and Industry. 2019. National Micro, Small and Medium Enterprises (MSME) Policy. Retrieved March 30, 2021 from [https://www.bcp.gov.gh/acc/consultation/docs/DRAFT%20MSME%20-%20FINAL%2026.02.2019%20\(1\).pdf](https://www.bcp.gov.gh/acc/consultation/docs/DRAFT%20MSME%20-%20FINAL%2026.02.2019%20(1).pdf)
- [10] Nici Pillemer. 2020. 10 Advantages of E-Commerce for Consumers & Businesses. Retrieved April 1, 2021 from <https://www.become.co/blog/ecommerce-advantages-consumers-businesses/>
- [11] Nikita Sharma. 2019. Recommender Systems with Python— Part II: Collaborative Filtering (K-Nearest Neighbors Algorithm). Retrieved April 21, 2021 from <https://heartbeat.fritz.ai/recommender-systems-with-python-part-ii-collaborative-filtering-k-nearest-neighbors-algorithm-c8dcd5fd89b2>
- [12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. Retrieved April 21, 2021 from <https://arxiv.org/abs/1708.05031>
- [13] James Loy. 2020. Deep Learning Based Recommender System. Retrieved April 21, 2021 from <https://towardsdatascience.com/deep-learning-based-recommender-systems-3d120201db7e>
- [14] Atymic. 2013. Twitter For PHP. Retrieved 21, 2021 from <https://github.com/atymic/twitter>
- [15] James Le. 2020. Recommendation System Series Part 5: The 5 Variants of Multi-Layer Perceptron for Collaborative Filtering. Retrieved 22, 2021 from <https://towardsdatascience.com/recsys-series-part-5-neural-matrix-factorization-for-collaborative-filtering-a0aebfe15883>

[16] Shopwise - eCommerce Bootstrap 4 Multipurpose HTML Template. Retrieved from <https://themeforest.net/item/shopwise-ecommerce-bootstrap-4-html-template/26080468>

[17] Startbootstrap. 2013. Retrieved from <https://startbootstrap.com/template/sb-admin>

Appendices

Appendix A: Npontu Mall API Documentation

This section defines the API for the Npontu Mall application. It outlines all the required specifications needed for the successful integration of the Npontu Mall developed in this paper to the Npontu Technologies Kedebah ERP. The parameters for each API endpoint are given with a detailed description on how to consume the API.

Document Details

Document Name	Npontu Mall API Reference Document
Document Date	Friday 23 rd April 2021
Prepared By	Allotei Pappoe
Purpose	Integration of Npontu Mall to Npontu Kedebah ERP

Seller Communication Request

This section defines the parameters of the request body about a user on the Kedebah platform who wishes to sell on the mall.

Api: <https://ecommerce.npontu.com/api/seller/>

Owner: Npontu Mall

Consumer: Kedebah ERP

Method: POST

Parameters:

1. `ked_sell_id`
 - a. Definition: the id of the ERP user who wants to sell in the mall.

- b. Datatype: integer
 - c. Example: 123
- 2. business_name
 - a. Definition: the name of the business of the ERP user who wishes to sell in the mall.
 - b. Datatype: varchar
- 3. Apikey:
 - a. Definition: a secret key that will authenticate the request

Response:

- 1. message:
 - a. Definition: status of the request
 - b. Datatype: varchar
 - c. Default value: created
- 2. Response code: 201

Sample Request Code:

```
$response = Http::post('https://ecommerce.npontu.com/api/seller',
[
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',
    'ked_sell_id' => 1,
    'business_name' => 'Essentials',
]);
```

Category Communication Request

This section defines the parameters of the request body about a category to be published to the Mall.

Api: <https://ecommerce.npontu.com/api/category/>

Owner: Npontu Mall

Consumer: Kedebah ERP

Method: POST

Parameters:

1. ked_id
 - a. Definition: the id of the category to be published.
 - b. Datatype: integer
 - c. Example: 123
2. cat_name
 - a. Definition: the name of the category
 - b. Datatype: varchar
3. Apikey:
 - a. Definition: a secret key that will authenticate the request

Response:

1. message:
 - a. Definition: status of the request
 - b. Datatype: varchar
 - c. Default value: created
2. Response code: 201

Sample Request Code:

```
$response = Http::post('https://ecommerce.npontu.com/api/category',  
[  
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',  
    'ked_id' => 1,
```

```
        'cat_name' => 'Electronics',  
    ] );
```

Product Communication Request

This section details the parameters of the request body for when a product will be published to the Mall.

API: <https://ecommerce.npontu.com/api/products/>

Owner: Npontu Mall

Consumer: Kedebah ERP

Method: POST

Parameters:

1. item_id
 - a. Definition: the ID of the item to published to the ecommerce store
 - b. Datatype: integer
 - c. Example: 123
2. item_name
 - a. Definition: the name of the item to be published to the ecommerce store
 - b. Datatype: varchar
 - c. Example: 'Black Air-force ones'
3. amount_available
 - a. Definition: the amount in stock
 - b. Datatype: varchar
 - c. Example: 500
4. sellers_id
 - a. Definition: the ID of the seller publishing to the ecommerce store

- b. Datatype: integer
 - c. Example: 123
- 5. picture_1
 - a. Definition: picture 1 of the product
 - b. Datatype: base64 encoding
- 6. picture_2
 - a. Definition: picture 2 of the product
 - b. Datatype: base64 encoding
- 7. picture_3
 - a. Definition: picture 3 of the product
 - b. Datatype: base64 encoding
- 8. picture_4
 - a. Definition: picture 4 of the product
 - b. Datatype: base64 encoding
- 9. Price
 - a. Definition: the price of the product
 - b. Datatype: integer
 - c. Example: 500
- 10. Currency
 - a. Definition: the currency of the product
 - b. Datatype: varchar
 - c. Example: 'GHC'
- 11. categories_id
 - a. Definition: the category of the product
 - b. Datatype: integer

c. Example: 123

12. item_description

a. Definition: description of the product

b. Datatype: Varchar

Response:

1. message:

a. Definition: status of the request

b. Datatype: varchar

c. Default value: 'created'

2. Response code: 201

Sample request code:

```
$response =  
Http::post('https://ecommerce.npontu.com/api/products', [  
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',  
    'item_id' => 1,  
    'item_name' => 'Black Airforce ones',  
    'amount_available' => 500,  
    'sellers_id' => 3,  
    'picture_1' => base64 encoding,  
    'picture_2' => base64 encoding,  
    'picture_3' => base64 encoding,  
    'picture_4' => base64 encoding,  
    'price' => 100,  
    'currency' => 'GHC',  
    'categories_id' => 1234,  
    'item_description' => 'Lorem ipsum ...',  
]);
```

METHOD: PUT or PATCH

API: <https://ecommerce.npontu.com/api/products/{product-id}>

Parameters: any of the product fields below the user wishes to update

1. item_id
 - a. Definition: the ID of the item to published to the ecommerce store
 - b. Datatype: integer
 - c. Example: 123
2. item_name
 - a. Definition: the name of the item to be published to the ecommerce store
 - b. Datatype: varchar
 - c. Example: 'Black Air-force ones'
3. amount_available
 - a. Definition: the amount in stock
 - b. Datatype: varchar
 - c. Example: 500
4. sellers_id
 - a. Definition: the ID of the seller publishing to the ecommerce store
 - b. Datatype: integer
 - c. Example: 123
5. picture_1
 - a. Definition: picture 1 of the product
 - b. Datatype: base64 encoding
6. picture_2
 - a. Definition: picture 2 of the product
 - b. Datatype: base64 encoding
7. picture_3

- a. Definition: picture 3 of the product
 - b. Datatype: base64 encoding
- 8. picture_4
 - a. Definition: picture 4 of the product
 - b. Datatype: base64 encoding
- 9. Price
 - a. Definition: the price of the product
 - b. Datatype: integer
 - c. Example: 500
- 10. Currency
 - a. Definition: the currency of the product
 - b. Datatype: varchar
 - c. Example: 'GHC'
- 11. categories_id
 - a. Definition: the category of the product
 - b. Datatype: integer
 - c. Example: 123
- 12. item_description
 - a. Definition: description of the product
 - b. Datatype: Varchar
- 13. apikey
 - a. Definition: unique secret key for authentication
 - b. Datatype: Varchar

Response:

- 1. message:

- a. Definition: status of the request
- b. Datatype: varchar
- c. Default value: 'Product was updated'

2. Response code: 200

Sample request code:

```
$response =
Http::put('https://npontu.ecommerce.com/api/products/{product-
id}', [
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7',
    'item_name' => $product_name,
    'amount_available' => $quantity,
    'price' => $ price,
    'currency' => 'GHC',
    'categories_id' => $ category_id,
    'item_description' => $ description,
    'quantity' => $request->quantity,
]);
```

METHOD: DELETE

API: <https://ecommerce.npontu.com/api/products/{product-id}>

Parameters:

- 1. apikey
 - a. Definition: unique secret key for authentication
 - b. Datatype: Varchar

Response:

- 1. message:

- a. Definition: status of the request
- b. Datatype: varchar
- c. Default value: 'Product was deleted'

2. Response code: 200

Sample Request Code:

```
$response =  
Http::delete('http://127.0.0.1:8001/api/products/' . $id, [  
    'apikey' => 'd5f85e4af0a7e7b4dc0d75fae74c22c7'  
]);
```


Appendix B: Connecting Kedebah ERP to Woo-commerce

Installation:

To install the Woo-commerce API – PHP Client run the following line in the terminal:

```
composer require automattic/woocommerce
```

Procedure:

1. Setup a form with the fields:
 - a. Website URL
 - b. Consumer Secret
 - c. Consumer Key
2. Create a table in the database which will store the woo-commerce credentials of the currently authenticated user. Columns will thus include: user_id, website_url, consumer_secret, consumer_key.
3. Create a WoocommerceConnectController which will accept the form input and validate the parameters by connecting to the store and creating an order webhook which will send orders to Kedebah as and when they are placed with the following code.

```
use Automattic\WooCommerce\Client;
use Automattic\WooCommerce\HttpClient\HttpClientException;

$woocommerce = new Client(
    $request->website_url,
    $request->consumer_key,
    $request->consumer_secret,
    [
        'version' => 'wc/v3',
    ]
);

try{

    $data = [
        'name' => 'Order Created',
        'topic' => 'order.created',
        'delivery_url' =>
        'https://ecommerce.npontu.com/woocommerce-order'
```

```

];

$validate_connection = $woocommmerce-
>post('webhooks', $data);

}catch(HttpClientException $e){

    $err_message = $e->getMessage();

    return back()->with('err_status', $err_message);
}

```

4. Encrypt the data and store it into the newly created table.
5. Create a new controller called WoocommerceController. In the class create a new method called constructor that will fetch the woo-commerce details of the user and connect to the store

```

public function constructor()
{

    $woo_credentials = Woocommerce::where('user_id',
    '=', auth()->user()->id->get());
    $consumer_key =
    Crypt::decryptString($woo_credentials[0]->consumer_key);
    $consumer_secret =
    Crypt::decryptString($woo_credentials[0]->consumer_secret);
    $woocommmerce = new Client(
        $woo_credentials[0]->website_url,
        $consumer_key,
        $consumer_secret,
        [
            'version' => 'wc/v3',
        ]
    );

    return $woocommmerce;
}

```

6. Create a new table that will keep track of the products users publish to their woo-commerce store. Columns will include: product_id, woo_product_id
7. In the WoocommerceController, create a new function called store which will publish products to the woo-commerce store and record the product ID and the product ID on the woo-commerce store with the following code.

```

public function store(Request $request)
{
    $woo = $this->constructor();
    $product = Product::find($request->product_id);
    $data = [
        'name' => $product->product_name,
        'type' => 'simple',
        'regular_price' => strval($product->price),
        'description' => $product->description,
        'categories' => [
            [
                'id' => $request->category_id,
            ],
        ],
        'manage_stock' => true,
        'stock_quantity' => $product->quantity,

        'images' => [
            [
                'src' => 'image src of the product'
            ],
        ]
    ];

    $response = $woo->post('products', $data);

    WoocommerceProduct::create([
        'product_id' => $request->product_id,
        'woocommerce_id' => $response->id,
    ]);
    return back();
}

```

8. The following functions will be used to get the categories on the woocommerce store, update and delete products in the WoocommerceController

```

public function categories()
{
    $woo = $this->constructor();
    return $woo->get('products/categories');
}

public function edit($id, $data)
{
    $woo = $this->constructor();
    $response = $woo->put('products/'.$id, $data);
    return $response;
}

public function destroy($id)
{
    $woo = $this->constructor();
    $response = $woo->delete('products/'.$id);
    return $response; }

```

Connecting Kedebah to Twitter

This subsection comments on how the Kedebah ERP can be connected to Twitter to allow its users advertise their products on the platform. To begin an application on Twitter must be created with a developer account that will grant access to the Twitter API.

Installation:

Run the following in the terminal:

```
composer require atymic/twitter
```

In the .env file set the following environment variables

```
TWITTER_CONSUMER_KEY=  
TWITTER_CONSUMER_SECRET=  
TWITTER_ACCESS_TOKEN=  
TWITTER_ACCESS_TOKEN_SECRET=  
TWITTER_API_VERSION=
```

Run:

```
php artisan vendor:publish  
provider="Atymic\Twitter\ServiceProviders\LaravelServiceProvider"
```

Create the following in the web routes that will enable the user to login to their twitter account, and grant the ERP access to make posts on their behalf.

```

Route::get('twitter/login', ['as' => 'twitter.login', function(){

    // your SIGN IN WITH TWITTER button should point to this route
    $sign_in_twitter = true;
    $force_login = false;

    // Make sure we make this request w/o tokens, overwrite the default values in case of login.
    Twitter::reconfig(['token' => '', 'secret' => '']);
    $token = Twitter::getRequestToken(route('twitter.callback'));

    if (isset($token['oauth_token_secret']))
    {
        $url = Twitter::getAuthorizeURL($token, $sign_in_twitter, $force_login);

        Session::put('oauth_state', 'start');
        Session::put('oauth_request_token', $token['oauth_token']);
        Session::put('oauth_request_token_secret', $token['oauth_token_secret']);

        return Redirect::to($url);
    }

    return Redirect::route('twitter.error');
}]);

```

Figure 1

```

Route::get('twitter/callback', ['as' => 'twitter.callback', function() {
    if (Session::has('oauth_request_token'))
    {
        $request_token = [
            'token' => Session::get('oauth_request_token'),
            'secret' => Session::get('oauth_request_token_secret'),
        ];
        Twitter::reconfig($request_token);
        $oauth_verifier = false;
        if (Request::has('oauth_verifier'))
        {
            $oauth_verifier = Request::get('oauth_verifier');
        }
        // getAccessToken() will reset the token for you
        $token = Twitter::getAccessToken($oauth_verifier);
        if (isset($token['oauth_token_secret']))
        {
            return Redirect::route('twitter.login')->with('flash_error', 'We could not log you in on Twitter.');
```

Figure 2

Create a TwitterController that will handle making posts to the users account with the following code:

```

public function tweet(Request $request)
{
    //dd($request);
    if(isset($request->product_image))
    {
        $this->validate($request, [
            'status' => 'required|max:280',
        ]);

        //dd(Storage::disk('local'));

        //get image
        $uploaded_media = Twitter::uploadMedia(['media' => Storage::disk('local')->get('public/images/'. $request->product_image)]);
        $tweet = Twitter::postTweet(['status' => $request->status, 'media_ids' => $uploaded_media->media_id_string , 'format' => 'json']);
        return back()->with('status', 'You tweeted successfully');
    }
    else{
        //return Twitter::postTweet(['status' => $request->status, 'format' => 'json']);
        $this->validate($request, [
            'status' => 'required|max:280',
            'image' => 'required',
        ]);
        //dd($request->file('image'));

        //upload image
        $filenameWithExt = $request->file('image')->getClientOriginalName();
        $filename = pathinfo($filenameWithExt, PATHINFO_FILENAME);
        $extension = $request->file('image')->getClientOriginalExtension();
        $fileNameToStore = $filename.'_'.time().'.'.$extension;
        $path = $request->file('image')->storeAs('public/storage', $fileNameToStore);
        $asdfs = Twitter::linkify($request->status);
        //dd($asdfs);
        $uploaded_media = Twitter::uploadMedia(['media' => Storage::disk('local')->get('public/storage/'.$fileNameToStore)]);
        $tweet = Twitter::postTweet(['status' => Twitter::linkify($request->status), 'media_ids' => $uploaded_media->media_id_string , 'format' => 'json']);
        return back()->with('status', 'You tweeted successfully');
    }
}

```

Figure 3

Create a form that will take text and image inputs from the user and pass them to the above function.