School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc Computer Science

Project Title:
Maki: Multichannel Listing Management with Selling Agent Automation.

Supervisor:
Mustafa Bozkurt

Student Name:
Michael K Peres

Final Year
Undergraduate Project 2022/23

Date: 01/05/2023

Abstract

Online marketplaces have become an easy way in for individuals wanting to sell products online, however with the saturation of marketplaces, and need for platforms that maximise sales, is reliant on both marketplace visibility and price of product, tools like these are only available to large companies and require multiple tools to achieve the overall goal. In this paper, we discuss an implementation for a platform that achieves both, provides connectivity of different marketplaces to individual sellers without the expectations of underlying infrastructure like warehouses in place and guided selling, based on metrics and suggestion data for products.

Contents

Chapter 1: **Introduction**

Chapter 1 will explain the supporting material of the problem that the project is solving, which then goes into the specific problem statement. Aims and research questions are then discussed and lead to the literature review surrounding the project.

1.1 Background

In the last few years, the number of online sales in Great Britain has jumped from 19.7% in Feb 2020 to 26.6% in Dec 2021 (Lewis, 2022). In the same period, the retail footfall in shopping centres has decreased by 36.6%, further supporting the claim that shoppers are adopting the change to online outlets for their shopping needs rather than brick-and-mortar stores. Although the pandemic may have some effect, recovery is slow and even stagnate in some places.

For businesses trying to obtain another stream of revenue in the online retail space, the range of online marketplaces out there for selling products has become an easy and lucrative way in, marketplaces such as eBay, Amazon FBM, OnBuy and Etsy have become major places to sell products. It is evident, businesses have a variety of options to choose from to list their products.

In addition, a growing shift in e-commerce is starting to emerge named Social Commerce which is a new construct of merging “social media and social networking technologies” into the way we shop for products online, (Zhou, Zhang et al, 2013), for example, platforms such as TikTok, Facebook and Instagram now all provide the capability to list purchasable products from posts, the space for listing products is becoming endlessly vast.

Consequently, with such a saturated market, this increases the level of effort and coverage required for greater sales while selling online. Furthermore, businesses deciding to deploy their own store could cost a lot more to advertise products, rather than using a social marketplace like Instagram. To demonstrate, on average, 1 image-based post on Instagram reached 2706 users between 2021 and 2022, which is done with minimal costs compared to advertising (Dixon, 2023).

1.2 Problem Statement

Individuals and businesses attempting to join the online retail space have a plethora of options to choose from when it comes to listing products. However, listing on each individual site can be tedious and overwhelming due to the variety and multitude of components, such as policies, fee structures and usage conditions, that vary between these marketplaces. In addition, there is no connectivity between marketplaces, for example, if inventory changes on eBay, this change becomes a manual and time-consuming process in order to alter the remaining inventories on other marketplaces this product is listed on. The problem becomes more complex when we include actions such as modifying product price, title, or descriptions as they require more manual work from the individual/business.

Furthermore, with the steadily increasing number of marketplaces, competition to sell identical products becomes fierce due to price being “the strongest predictor of customer choice behaviour “(Smith, 2002). It is crucial that in order to compete with other sellers, pricing adjustments should be made to obtain the best possible sales outcome.

Therefore, the need for a platform that can provide connectivity between marketplaces and pricing changes algorithmically based on market conditions is a necessity for online survival.

Solutions to this problem include multichannel listing platforms, however, these platforms are usually catered for large businesses with over 15,000 listings as a minimum and thus have pricing models that are unattainable for individuals or small businesses starting out, including software built for warehouses. Furthermore, a solution for the problem of managing multiple products on multiple marketplaces and understanding the product via metrics is lacking and explained later in Chapter 2.

1.3 Aim

The aim of this project is to create a web application catered to individuals and small businesses, who need a platform that allows them to complete actions that are then relayed on to multiple marketplaces, which also permits pricing changes of products to be made on their behalf algorithmically. Ultimately the application aims to provide an easier way to perform repetitively tedious tasks and achieve greater sales in the online retail space that is accessible to individuals with low investment needs. It will do this by catering for users from the initial selling decision to the creation of listings and the repricing of products, which is not present in any one tool.

1.4 Objectives

The following objectives are needed to solve the problem outlined and are derived from aims.

- To analyse and contrast current approaches in order to identify key characteristics and limitations of these platforms.
- Create a platform that allows users to oversee their listings on multiple marketplaces.
- Create a functionality that grants users CRUD operations on their listings across these marketplaces.
- Allow users more insights to product performance in different marketplace with the ability to investigate key metrics and marketplace insights.
- Creation of a selling agent that deals with automatic repricing of products in relation to market conditions and seller.

1.5 Research Questions

To delve deeper into this project, the following research questions were formulated in order to get a better understanding of the overall goals of the project.

- Understand the needs of sellers/ small businesses in the online marketplace setting.

- How can we reprice product prices based on other sellers algorithmically?
- What specific aspects of marketplaces aid users in purchasing and giving more insights to buyers before selecting a product?
- How can product categories of multiple marketplaces be dealt with?
- What features of current solutions make it easy to sell or buy on marketplaces?
- What are the effects of ease of use in website design and flow?

Chapter 2: **Background Research**

Understanding the most recent research being done on multichannel listing platforms, repricing algorithms, and other relevant technologies was crucial to this project. Therefore, an extensive literature review was undertaken as a result. Additionally, a review of existing applications currently available and their advantages and disadvantages is laid out.

2.1 Literature Review

2.1.1 Online Marketplace

In this section, the report will first discuss what an online marketplace is, the benefits of a marketplace and product identifiers.

An online marketplace is a platform where manufacturers or distributors sell their products directly to online customers and share their revenue with the platform (Alaei, Taleizadeh and Rabbani, 2020). Some prime examples of marketplaces include eBay, OnBuy, Etsy, Gumtree and Amazon, where users can list their products.

For individual sellers, the use of e-commerce to list products comes down to increased product visibility, compared to trying to sell locally where the visibility of a product may only be limited to the individuals' personal connections. For small businesses, it is stated the reasoning for using online marketplaces was due to it providing a "competitive advantage" and "perceived benefit," (Fahri, 2008).

In order to keep track of products, the introduction of product identifiers has been created, and depending on the environment, different codes are used. This section provides brief definitions of the identifiers that will be used in the following chapters of the report and project implementation

GTIN, Global Trade Identifier Number, is used by a company to uniquely identify all of its trade items.

EAN/UPC numbers are used to identify consumer products in the world and have different variants of codes depending on region and product type.

SKU refers to Stock Keeping Unit and is mostly used by platforms like our solution for creating references to products in our stock.

EPID/ASIN are platform-based identifiers, used for eBay and Amazon respectively for identifying specific unique products in their marketplace.

2.1.2 Multi-Channel Listing Software

Multi-Channel Listing Software, MCLS, is software that can automate the managing of product listings across different marketplaces, it can be used to reduce the amount of time required to utilise the benefits of using multiple marketplaces without the disadvantages associated with it (Zabidi et al., 2022).

Why MCLS are beneficial based on Search Theory:

Search theory is the study of economics which investigates the inefficiencies and friction of the practical world when it comes to things such as transactions between two parties (Boyle, 2021). The benefits of MCLS can be studied for the seller and buyer through research on search theory. It has two main components, the benefit of search and the cost of search. In the context of online marketplaces, this means that consumers will continue to search/seek for something that benefits them for as long as it outweighs the costs of that search (Kuruzovich and Etzion, 2018).

Search theory can explain why higher product prices may lengthen the time to find a willing buyer and why a seller may want to alter their pricing after some time. To expand on this, buyers will only carry on searching for a product until the cost outweighs the benefits, as explained by Kuruzovich. To put this in perspective, a person would not walk to another retailer that is an hour away if they knew it was only 10 pence cheaper. This same notion can be applied online with sellers, they can choose to increase the price, but only if they are willing to wait for a customer that does not mind searching for a product extensively.

Effects of MCLS and online sales:

Research into whether MCLS can be seen as effective was conducted where it was concluded that “retailers with a higher degree of interchannel coordination tend to achieve higher online sales”, (Pentina and Hasty, 2009) and the possibility of interchannel coordination is the reason to provide a competitive advantage within the online retail space. MCLS allow users to handle a higher degree of interchannel coordination and thus does provide greater online sales.

Effect of MCLS on overall retail space:

Building upon these statements, it is argued that by reducing the cost of searching for a buyer, the benefits of searching can outweigh the costs longer, in the online world that will mean searching more stores to find the best price before stopping, however in an offline context, such as a high street, the cost of the search is much higher and thus the search of products may be limited to a much smaller number of stores. This puts pressure on online retailers to sell at a lower price compared to offline retailers based on the low cost of searching other competitors (Bakos, 1997), and in fact, online sale prices are typically lower than their offline counterparts (Bajari et al, 2003) (LUCKING-REILEY et al., 2007).

MCLS Conclusion:

Although most buyers favour lower prices, MCLS can allow sellers the ability to sell items at much higher prices using search theory to its advantage, by placing such items in multiple locations with ease, this should lower the cost of searching for a high valuation buyer by increasing product visibility to a bigger pool of potential buyers.

2.1.3 Algorithmic Repricing Strategies

Although search theory can help find customers, it is not the major deciding factor in purchasing a product online, as mentioned in the background chapter, price is. A review of currently researched algorithmic repricing strategies was undertaken to determine the best approach to this problem, we use the solutions mentioned here as a starting point in our own implementation.

Buy Box and Sellers Objective:

When an identical product is listed by multiple sellers, sellers compete to obtain the best “online placement,” as it comes with improved sales. The best online placement for example on Amazon is called the buy box, this is because as depicted in Figure 1 it is the first option buyers see when they are looking to buy a product. Buy box allows sellers to be 1 click away from the purchase, compared to being in the Other Sellers category which contains multiple sellers. Price changes has the effect on the win of the buy box as sellers using algorithmic repricing obtain the buy box more frequently than normal sellers (Chen et al, 2016).

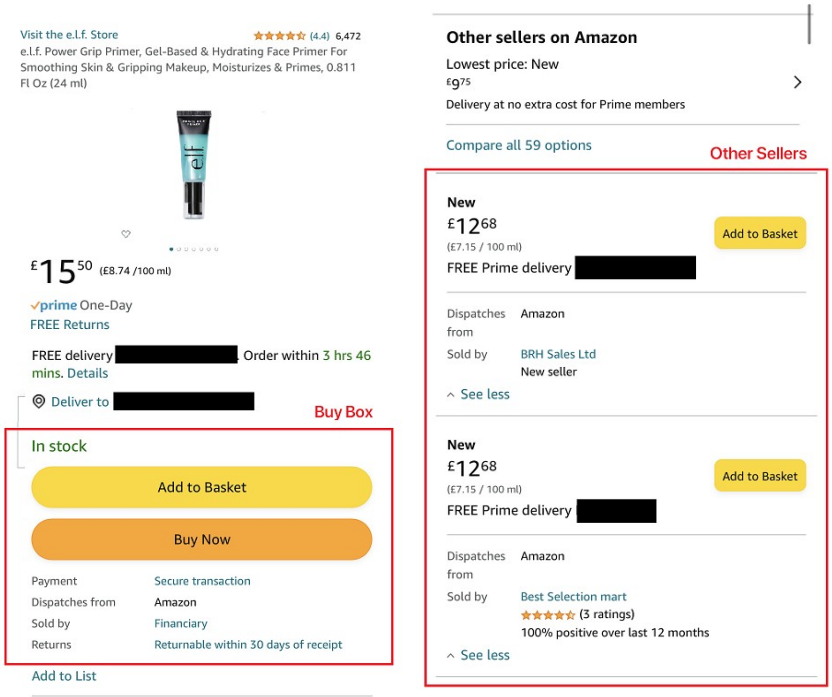


Figure 1 Buy Box and Other Sellers Box on mobile app.

Alternative to buy boxes on different marketplaces:

The 2 main jobs of marketplaces are:

- To understand exactly what product the seller is selling.
- Match a buyer’s request to a specific seller.

Given this knowledge above, the buy box in this case would be obtaining the first item within this query response, to do such a thing, optimisation to title, category and pricing may shift its position up the list to the first position.

There are many ways to obtain the “buy box” but this is specific to the platform. For example, Figure 2 shows the range of products based on the query we can see which items are presented first and could suggest lowercase product titles performs better than title case. However, one factor that is prominently used to determine order is pricing (Chen et al, 2016). where “Price Difference to the Lowest Price” was determined to affect the buy box by a weighting of 36% obtain via scraping the Amazon website. So it is no surprise that pricing algorithms have been adopted by companies trying to obtain the upper hand in the marketplace. Companies such as British Airways introduced pricing algorithms as early as 1970s (Calvano et al., 2019), showing us that this is an important and crucial feature for business in general as well as e-commerce.

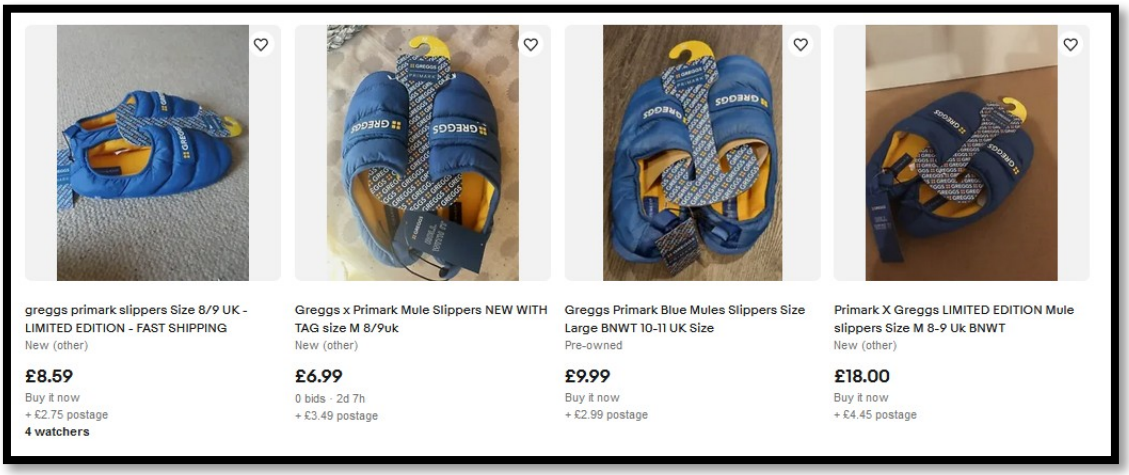


Figure 2 eBay Search Results.

Algorithmic Repricing Strategies in the Wild:

This strategy includes the modifying of the pricing of items automatically, we extend this by basing it on several metrics or features of the current marketplace. The paper talks about multiple different approaches are talked about when it comes to pricing strategies and so we will go through a few of them. Overall, the strategy aims to obtain the maximum profit possible for a product while limiting the time to do so. Such a strategy is tedious to do in an offline setting, as it is labour-intensive, and requires a lot of information that is not readily available online. An approach to determining the correct strategy to use is mentioned (Hwang and Kim, 2006). It describes 3 phases in creating a dynamic pricing algorithm as shown in Figure 3.

Algorithmic Repricing Strategies are usually formulated in 3 different steps. Data collection, strategic analysis, and strategy formulation.

In short, data collection obtains data related to customer info, other competitor prices and price fluctuations on a periodic basis. Strategic analysis determines the estimated revenue, so we can determine the relationship between pricing, profit, and volume, by looking at sales volumes of related products. Finally, strategy formulation, is based on 3 different possible objectives:

- “Maximise long-run or short-run profits”.
- “Increase sales volume or return on investment”.
- “Build store traffic or price for market survival”.

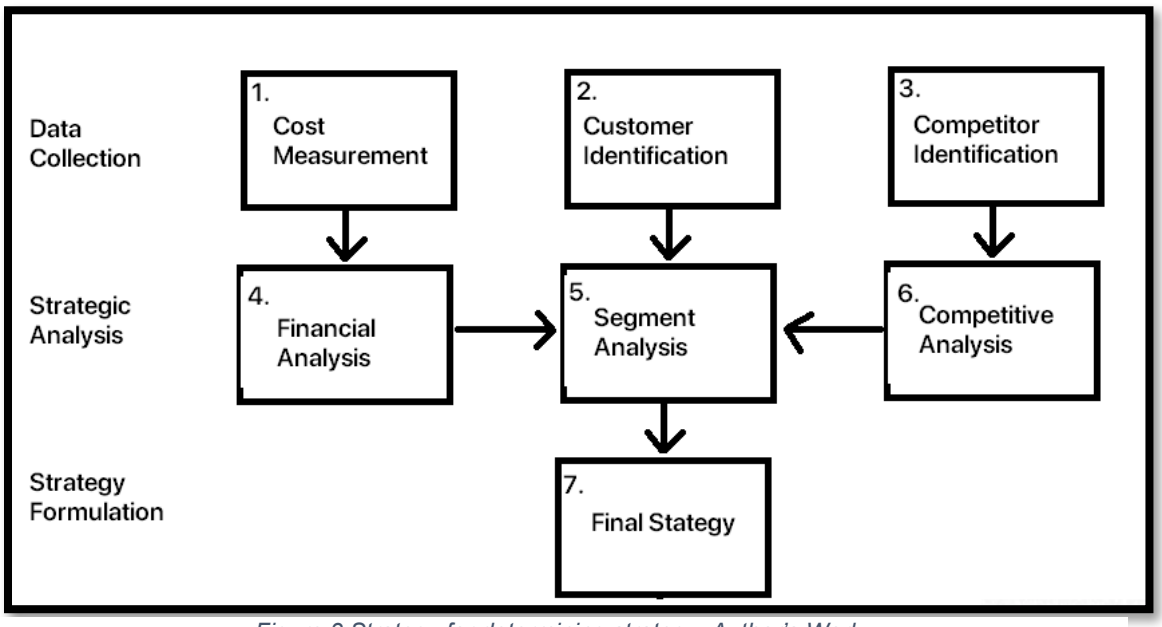


Figure 3 Strategy for determining strategy. Author's Work.

All of these can be achieved by adjusting item prices below or above the line shown in Figure 4.

The values on the red line show a strategy where profit is frequent as the price is

Figure 4 Meanings of price above and below the predicted graph.

lowered less than the average price drop over time.

The main goal of these selling agents is to sell at a specific desired price and if that is not possible, slowly decrement to the lowest acceptable price instead (Maes, 1996). Figure 4 depicts this. The value of the item is slowly decremented over time to eventually reach a minimum, where the price tries to sell at different product price levels gauging buyer interest.

Maes describes a strategy which is relevant to this project. They describe a selling agent that is based on the desired sell-by-date variable, initial price, and lowest acceptable price. It then uses these to determine the price to sell at. The main assumption with this agent is that the price is inversely proportional to buyer interest and ideally, prices will shift between different price brackets until it reaches a sale. The trajectory from the desired price to the lowest asking price is the secret sauce in algorithmic pricing. In essence, the agent uses a linear decay function to determine the

price to sell at, which can gauge the interest of a product based on its price, by constantly checking for purchases at that price point, and recording sale history for later use. Background information such as market data can also be used to update the prices. This is implemented in Chapter 5.

2.2 Existing Applications

This section discusses existing solutions or applications that provide a subset of the features that this project aims to fulfil.

2.2.1 StreetPricer

StreetPricer[1], is an existing application that reprices a user's items based on rules set by the user. Figure 5 shows their dashboard. They have two main approaches to repricing which are explained below.

- **Velocity-Based Repricing** is for non-competitor and unique items. Price is controlled by the selling rate and if products sell too fast, then the price is increased to slow sales down.
- **Competitor-Based Repricing** tracks competitors within eBay and allows different strategies of either price matches, undercuts, or hovering above current selling prices.

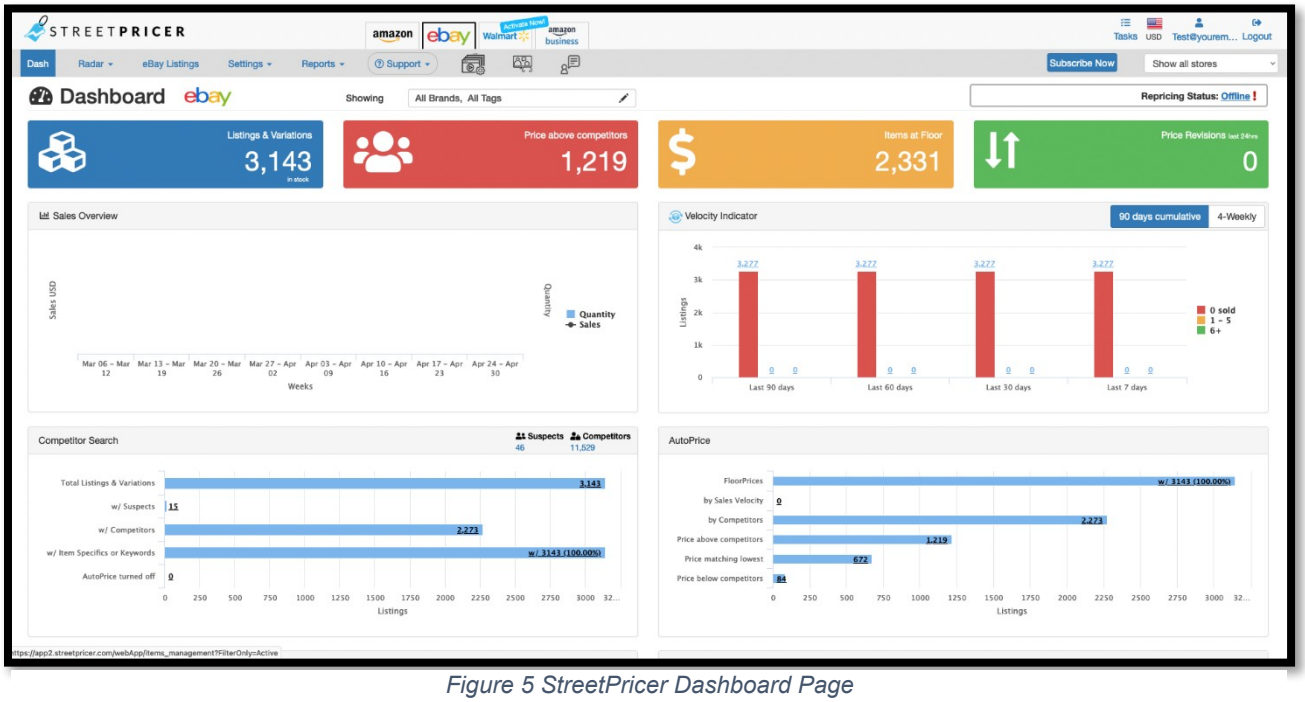


Figure 5 StreetPricer Dashboard Page

Advantages:

- The application allows users the ability to switch from different approaches and strategies depending on the scenario and product.
- Official eBay re-price partner.
- Use of AI to analyse other repricer

Disadvantages:

[1] StreetPricer, https://streetpricer.com/ebay-repricer/

- This requires quite an expensive investment to access their services, targeting small to medium businesses selling up to 10k items.
- Repricing is only limited to Amazon and eBay.

2.2.2 SellerSnap

SellerSnap[2] uses AI to reprice items as if it were a real Amazon seller and makes pricing adjustments in order "to outsmart their competitor".

Advantages:

- Use of AI and strategies are catered towards specific competitors, use of velocity-based repricing.
- This means if sales of an item occur frequently, increasing the price may maximise profit.

Disadvantages:

- The use of AI is limited to only the Amazon marketplace and does not support other marketplaces.
- The expensive pricing model starts from $250 per month for up to 15,000 items.

2.2.3 SkuVault

SkuVault[3] is an e-Commerce Inventory Management Software, which offers a service that covers all aspects of e-commerce activities, from picking to listing and shipping. It states at the time of checking, 87% faster fulfilment, 90% fewer stocks, and 83% fewer mis-ships.

Advantages:

- Has several integrations, with marketplaces Amazon, BackMarket, eBay, Etsy, Newegg, Reverb, Walmart, Wayfair BETA.
- Allows the sync of inventory throughout multiple e-commerce.

Disadvantages:

- The majority of marketplaces are tailored for US customers, such as Newegg and Walmart.
- The expensive pricing model starts from $329 per month and only allows.
- Includes other services such as dealing with warehouse picking locations, which the typical seller does not really need or is concerned with.

2.2.4 Unicommerce

Unicommerce[4] is the largest Indian e-commerce focused SaaS platform and provides solutions such as Multichannel Order Management System. This deals with the fulfilment of multiple products on multiple marketplaces, for example, listing and shipping and fulfilment status.

Advantages:

[2] SellerSnap, https://www.sellersnap.io/
[3] SkuVault, https://www.skuvault.com/
[4] Unicommerce, https://unicommerce.com/multichannel-order-management/

- Has an immense number of integrations with well over 100 marketplaces and karts.
- Includes shipping, ERP/POS, and accounting systems also.

Disadvantages:

- An expensive pricing model such as multichannel order management software requires the Enterprise level, suitable for “large-scale businesses.”
- The majority of integrations are with marketplaces that have a higher presence in Southeast Asia, Taiwan, and include retailers where stock and business presence requirements are necessary before selling.

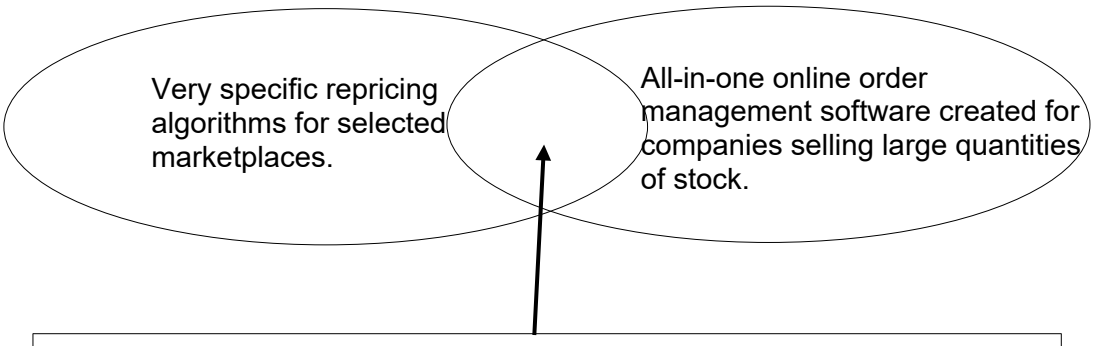
2.2.5 Summary of Existing Applications

Application Name	Re-pricing/ Multichannel Ecommerce Automation	Features
StreetPricer	Re-pricing	Tries to achieve higher sales for clients, by changing the pricing based on user preferred rules.
SellerSnap	Re-pricing	This application uses AI to reprice items as if it were a real Amazon seller, and making pricing adjustments in order “to outsmart that competitor”
SkuVault	Multichannel eCommerce Order Management	Removes manual tasks and synchronising with marketplaces. Deals with the entire process of pick, pack, and ship procedure. Comes as complete package with software for warehouses.
Unicommerce	Multichannel eCommerce Order Management	Application aims to increase sales by utilising multichannel marketplaces and seamless integrations. This includes a centralized dashboard for a unified view across all sales channels linked.

Figure 6 Table summarising existing applications.

2.3 Gap in the Market

Pick what each one did well and explain what they have not done and implement these things in the project.



Gap in the market: Software that allows repricing for x number of platforms and is more abstract then specific, with functionality of online order management that is more accessible to the average eBay seller instead of small businesses.

2.4 Summary

To summarise, the following points explain why these existing applications are invalid for this project:

- Software is mostly catered for businesses where prices charged are not accessible to everyday people that want to sell with this tool.
- Repricing software is catered for getting buy boxes for specific platforms.
- Platforms require a great amount of financial investment and require you to already have some sort of functioning business.
- The platform is seen as an improvement of efficient service rather than a platform to use.
- Platforms usually target only massive marketplaces that are already really saturated and do not integrate with smaller mobile marketplaces such as Gumtree via private APIs (App integration).

The background knowledge obtained from the literature review and existing applications will aid the development of this project.

Chapter 3: **Analysis and Design**

This section includes domain, requirements, use-case analysis and the design of the project.

3.1 Domain Analysis

3.1.1 Introduction

This document describes background information that has been gathered about listings within different marketplaces and the automation of selling agents as well as how they are handled. With this, it can aid the development of software to allow the management of listings within multiple marketplaces. Multichannel e-commerce automation involves the listing of products on multiple sale channels and managing of sell prices algorithmically.

3.1.2 Stakeholders

The stakeholders in this system are:

- Company owner.
- Maintenance / Normal Operators
- Customers want to sell their products online on marketplaces.
- Online Marketplaces.
- Purchasers of online marketplace listings.
- Other competitors (negative stakeholders, opposed to the system.)

3.1.3 Customers and Users

- Seller

  This is a user logged into the system and attached at least one marketplace, so can list their products on the system.

- Seller Specific Agent

  All seller users are assigned a seller-specific agent, that helps submit jobs for that user automatically, things such as finding the best optimal price today for the listed product. Dealing with KPIs.

3.1.4 Tasks and Procedures currently performed:

User:

Logging in/Registering:

Users will need to register an account and log in. The system will store information about the user such as tokens for specific marketplace integration and other user settings.

Delete account:

Any user in the system is allowed to delete their own account. This will also delete all listings within their account and subsequently all attacked marketplaces.

Allow integration with the specified marketplace:

Users will need to log in to each third-party marketplace and allow this application to act on behalf of them before they are able to list anything on the platform. This makes a specific marketplace available to be listed on when added.

Create a product listing:

Users can create a listing on the system and will be automatically listed on each selected platform. The user must fill in information about the product and upload some images.

Get suggested price for product:

Users can request a suggested price for a product they are listing, more information about the type of product may be requested if the process cannot find such product on the platform.

Remove a listing:

Users can completely remove a listing from the system, which will also remove the listing from each attached marketplace.

Update a listing:

Users can update a listing from the system by changing different things of the product such as the title etc, this added information will be used to update attached marketplaces.

Seller Specific Agent:

Determine Price Change for listing:

Agent can obtain a value on whether a listing should increase or lower its price by a specific magnitude (-1 -1) based on KPI, and overall marketplace data for that specific listing. If auto pricing is enabled, the seller agent can update the price of a listing automatically periodically depending on setting.

Get platform specific KPI for listing:

The agent can obtain record of indicators from platforms based on a specific listing, things such as likes, watches etc, depending on the platform and how KPIs are implemented.

Update a listing pricing for listing:

The agent can update the selling pricing of a listing; this can be done after price change is determined.

3.1.5 Domain Model

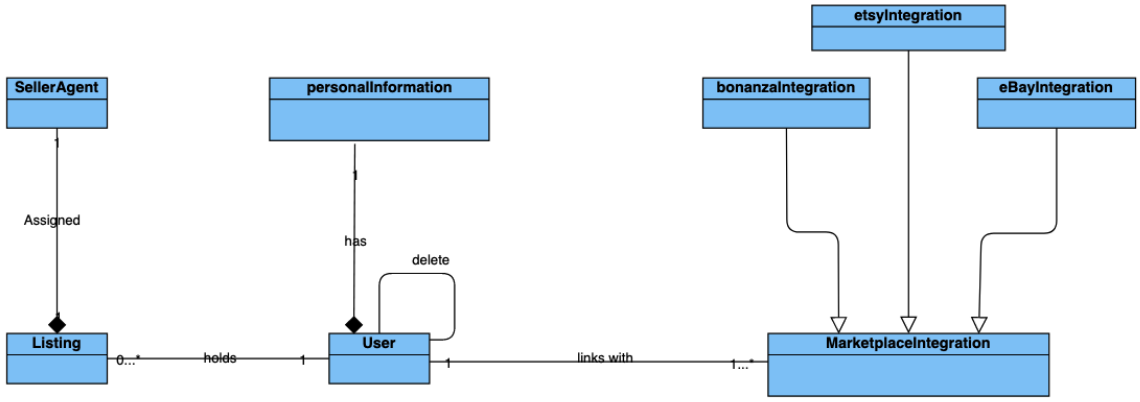


Figure 7 Domain Model of the current system.

Figure 7 shows the model used in the system, where the user entity has a link to both marketplace integration, personal information and listing. Each listing is assigned a SellerAgent entity that is in charge of re-pricing the product. If the user is deleted, the personal information object will also be deleted.

3.2 Requirement Analysis

3.2.1 Functional

These are actions a user of the system would need to know regarding what the system does.

ID	Requirement	Type	Priority
1	Users must be able to register an account.	Functional	Core
2	Users must be able to login into their account.	Functional	Core
3	Users must be able to delete their account.	Functional	Core
4	Users must be able to connect to marketplace integrations.	Functional	Core
5	Users must be able to remove marketplace integrations.	Functional	Core
6	Users must be able to select integrations before listings.	Functional	Core

7	Users must be able to deselect integrations before listings.	Functional	Core
8	Users must be able to fill in information about products that they want to sell.	Functional	Core
9	Users must be able to fill in postal information about a listing they want to sell.	Functional	Core
10	Users must be able to link an image URL at least.	Functional	Core
11	Users must be able to add a listing to the system.	Functional	Core
12	Users must be able to update a listing.	Functional	Core
13	Users must be able to delete a listing completely.	Functional	Core
14	Users must be able to view all listings	Functional	Core
15	User must be able to view state of listing, [Draft, Active Stopped]	Functional	Core
16	User must be able to see which integrations are applied to a listing	Functional	Core
17	User should be able to remove stopped listings.	Functional	Non-core
18	User should be able to activate, draft or stopped listings	Functional	Non-core
19	Users should be able to see KPI and metrics specific to integrations for listings	Functional	Non-core
20	User should be able to upload images to the system	Functional	Non-core
21	User should be able to search for products on multiple marketplaces based on keywords	Functional	Non-core
22	User can update user account details.	Functional	Non-core

3.2.2 Non-functional

Everything that would concern any other system that has to interface to this system.

27	The system must be available for maintenance from 1:00am to 3:00am.	Non-Functional	Core
28	The system must be available 95% of the time.	Non-Functional	Core
29	The system must have a user/event response time when requesting web resource of no more than 3 seconds.	Non-Functional	Core
30	Acceptable time the GUI can take to respond to action performed by user, e.g., click button to go to next screen is 1 second	Non-Functional	Core
31	The system must have a probability of data corruption on failure of less than 5%.	Non-Functional	Core
32	The system must restart after a failure in less than 25 mins of it occurring.	Non-Functional	Core

3.3 Application Design

First part of the implementation is the design, created with Adobe XD. My initial steps were creating the design of the app, including what functionalities the app should contain, with a focus on the interfaces related to listing CRUD functionality. Designs were all centralised based on the main functionality of the app. Figure 8 shows the initial design for creating a listing with options for integration selection. The aim of these designs is to keep the interface intuitive and simple, with an effort for listing to a minimum, and implicitly set values for efficiency. Figure 9 shows the dashboard and integration interfaces made.

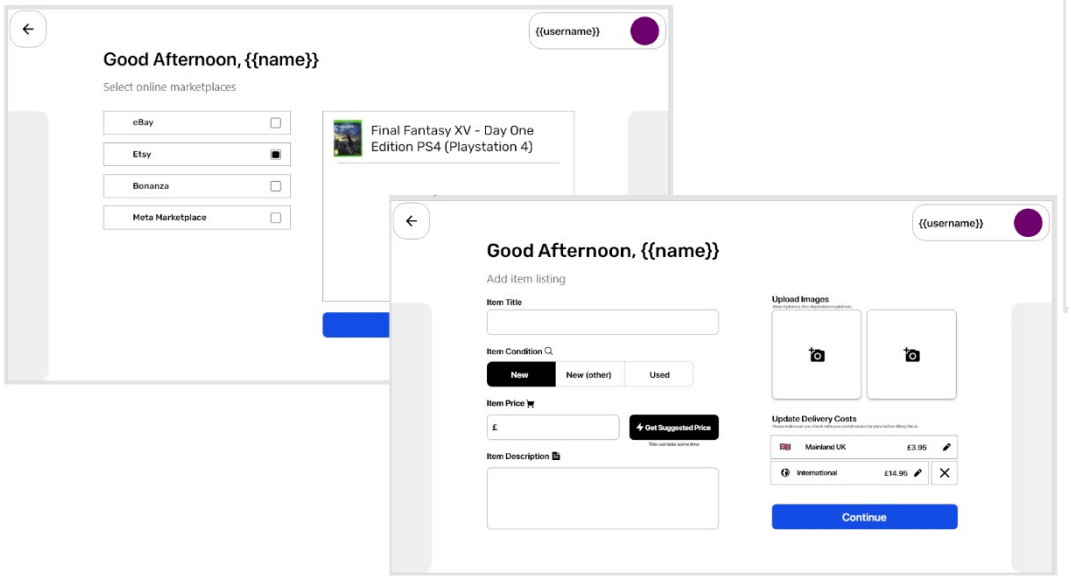


Figure 8 Add Listing Interface

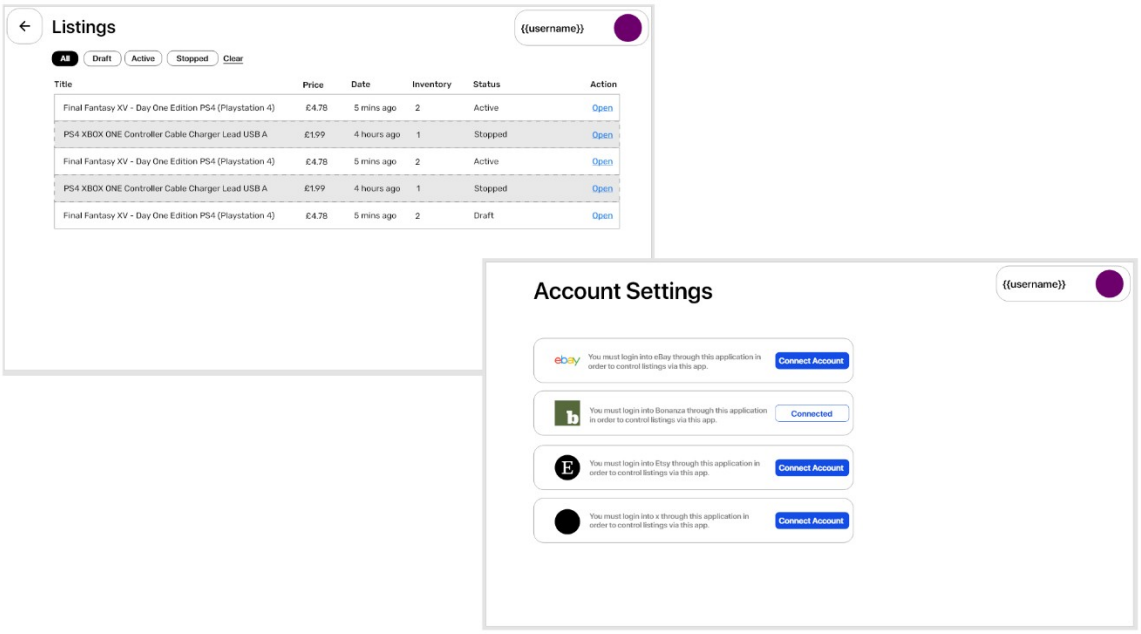


Figure 9 Dashboard and Integrations

Initial design of the database created contains Entities that were common with that of a multi-listing channel platform, where Users have both listings and integrations. Figure 10 reflects these assumptions in the diagram.

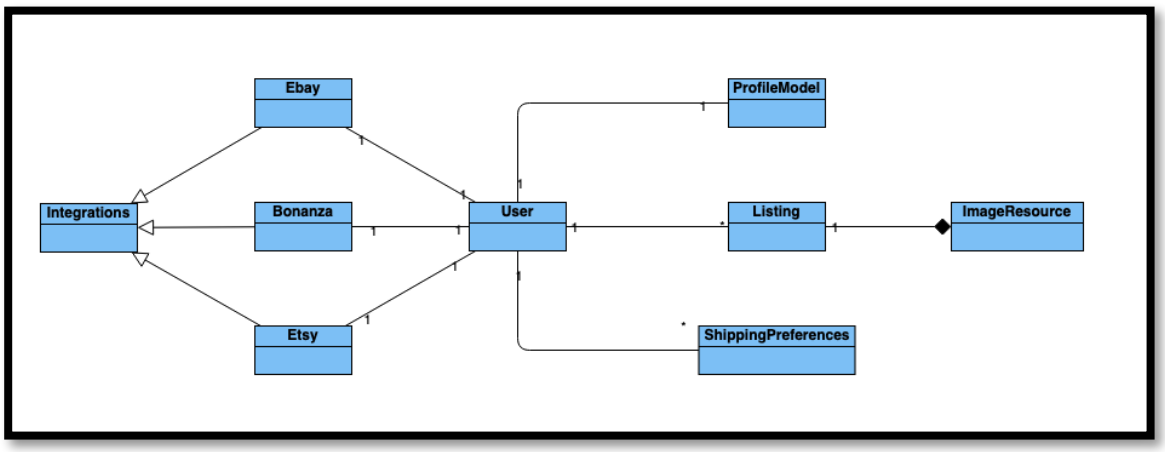


Figure 10 Current Implementation Model ER Diagram

3.4 Alternative Designs

An alternative design for the add product has been made. When developing add product, there was a dilemma, in which, the add product form should be quick and easy to submit. However, users may want to also submit very precise information and thus require a longer form. This design includes a lot more form fields, including specific categories and shipping preferences, under the URL path: **/listings/create/full/,** shown in Figure 11.

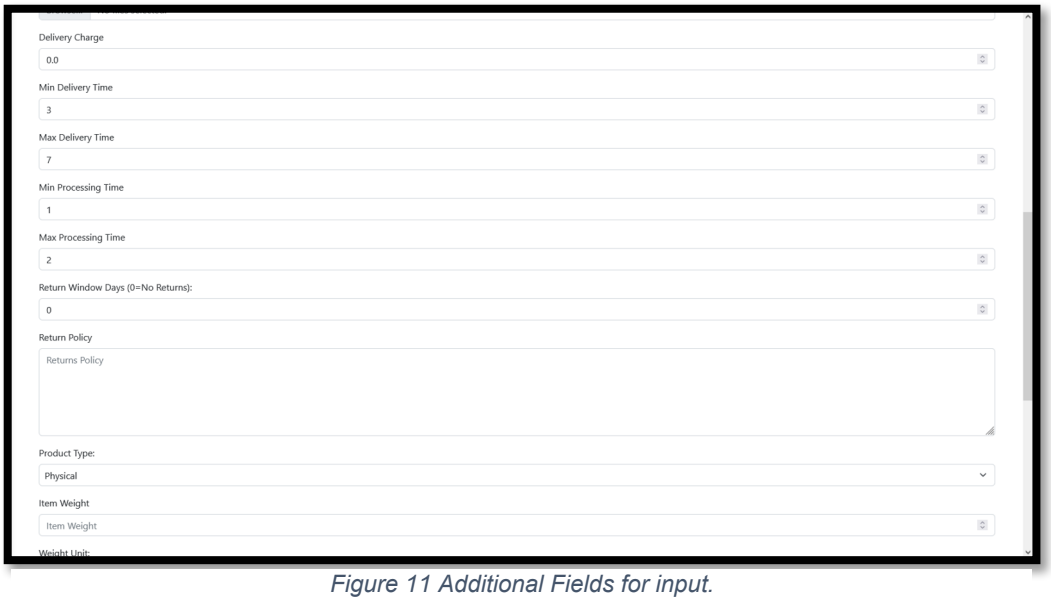


Figure 11 Additional Fields for input.

An alternative design was made for the overall site shown in Figure 12, where functionality was laid out in a list, illustrating all the choices to the user from one page. However, this can provide the user with too much information to view that isn't necessary, such as the ability of creating a listing, when editing account information.

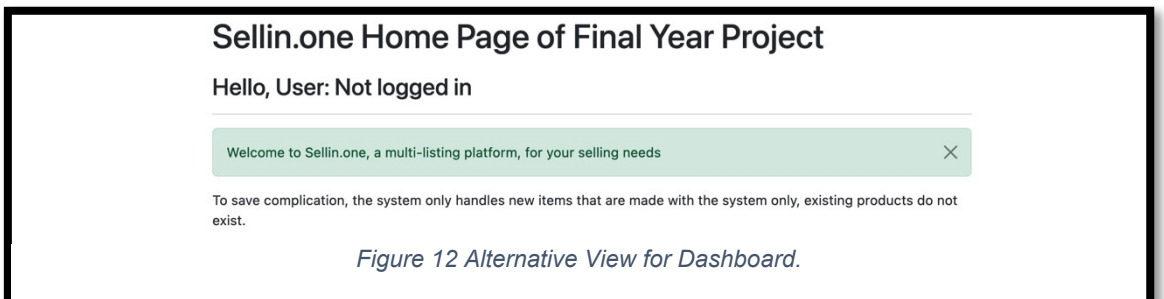


Figure 12 Alternative View for Dashboard.

Chapter 4: **Implementation**

4.1 Rationale for the Platform and Technologies

Based on my previous extensive experience with Python and web server-related projects, Django was an excellent choice for the backend as it is powered by Python and provides ORM by default. **Appendix A** contains all the third-party libraries used.

For the frontend, Django's built-in template system was used to focus more on functionality rather than aesthetics and feel, this allowed creation of frontend pages using HTML, CSS, and vanilla JS. The entire code was submitted in the supporting material section on QMPlus.

The report will follow the structure of the initial setup and then will go into the project's main tasks, including:

- Account Based Functionality
- Multi Listing Functionality
- Taxonomy, Category Handling Functionality
- Metric and Search Functionality
- Marketplace and Product Insights Functionality

The report will now explain some crucial technologies implemented in this project to provide clarity around the solution.

Django Templates:

As the project doesn't use a dedicated frontend framework such as React or Vue, an explanation of Django templates and forms is required. This project used templates for the frontend views. Templates are partial HTML documents with the ability to add logic such as loops, conditional branches and permit the passing of data. These are then parsed by Django's template engine which outputs complete HTML code. The project also uses Django forms, which allows for the creation of custom forms based on class definitions. Instances of these classes, when defined in templates, will automatically be converted to HTML code via the template engine, with features such as form validation and input sanitation.

Conda (Environment):

Conda manages packages, dependencies, and environments. In this project, the use of this software is also used for maintaining the project environment including all its environmental variables and libraries separate from other projects within the local machine.

Daphne (ASGI):

Daphne is a Python based ASGI webserver. The project uses this because the server is required to handle more than just HTTP(S) traffic. Due to the project also handling WSS protocol, a simple WSGI webserver is not sufficient.

Backblaze (Cloud Storage):

This is IaaS platform that offers cloud storage with S3 Compatible APIs, they offer 10Gb of free storage for usage within their platform. The project uses this as the source

of temporary storage (30 days max) to upload user images that will eventually be hosted on the marketplaces themselves.

Whitenoise (Static Handling):

This is a Python package that's self-contained and can run anywhere, this gives the webserver the option to serve static files such as JavaScript, HTML and CSS files. The project requires this because when running Django in production mode, static handling of files is automatically disabled by Django.

Redis (Caching):

This is an in-memory option for storing data, whether it be in a database or cache. In this project, Redis is used as a cache and is configured on Django for this functionality. Redis is used in the project for rate limiting users' requests and caching of specific JSON and HTML responses.

API Wrapper Function (Network):

Services and marketplaces offer APIs (Application Program Interface) that allow programs to interact with them via HTTP GET and POST requests, which this project relies on. These services communicate through network calls, API wrappers allow users to use functions rather than creating the raw network call to the server, these functions usually also handle errors and data formatting.

Docker (Deployment):

Docker allows for the containerizing of applications, which can be run on any system through virtualisation. The project uses this in order to easily deploy the solution on an instance on Oracle Cloud.

4.2 Environment Setup and Architecture

4.2.1 Environmental Setup Process

The following steps were required for developing and setting up the project.

- Creation of Conda virtual environment so that all libraries and processes are isolated in their own context to the local machine.
- Development of API wrappers to marketplaces that build the integration support.
- Building of the specific concrete backend models that will store data and the views, so full API was accessible.
- Frontend templates and corresponding JavaScript functionality and CSS styling was required to allow interaction between the frontend and the backend.
- Implementation of containerized application so it could be deployed on the cloud.

4.2.2 Network Internal and External Architectures

The internal architecture of the system is illustrated in Figure 13.

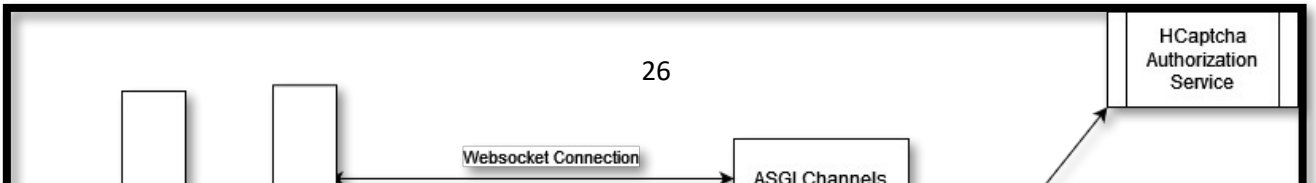


Figure 13 Internal Architecture of Software

Internal Architecture:

- Handling of WebSocket protocol, using Channels and Daphne. Daphne handles both HTTP and WebSocket protocols for ASGI.
- Static files serving, this uses a separate storage location for serving JS, CSS and SVG using Whitenoise.
- Usage of Redis in place for caching expensive API calls, can handle caching of calls and rate limiting features of web server.
- Usage of Backblaze here to provide cloud storage with the usage of AWS (Amazon Web Services) S3 bucket.
- Usage of Captcha is used for preventing bots' usage on my site and is one of my risk mitigations.
- Usage of Cloudflare is to prevent security attacks that are beyond my time resource for my FYP, and obtain a TLS certificate for the domain `sellin.one`

The external architecture of the system is shown in Figure 14 and explained below.

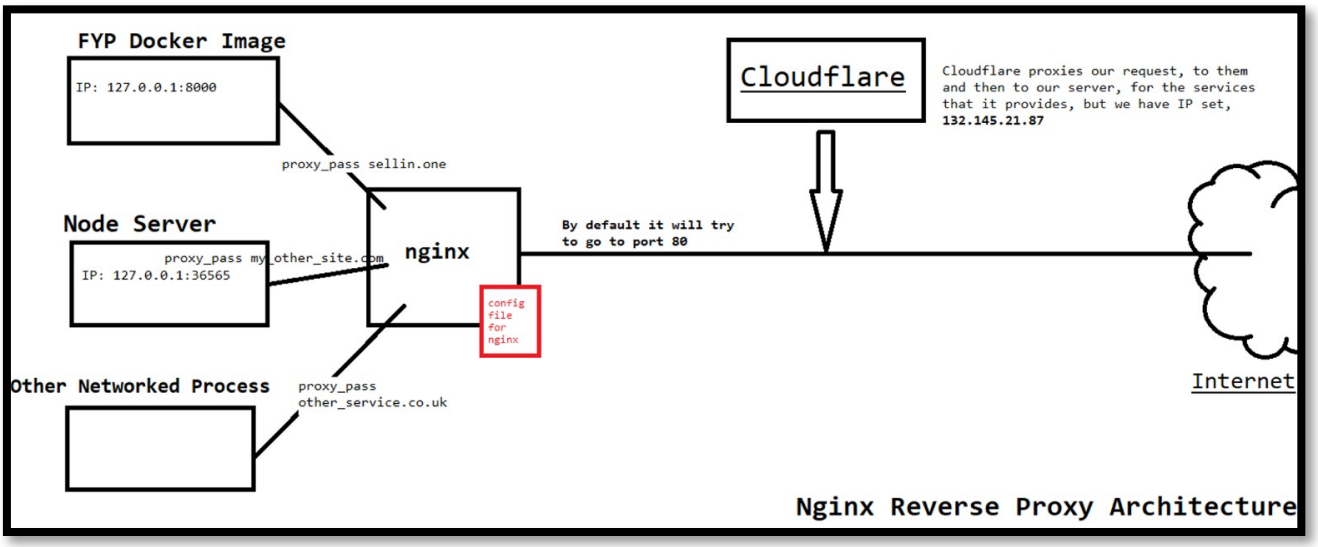


Figure 14 External Architecture of System.

External Architecture:

- App deployed on an Oracle Linux Ubuntu VM (Virtual Machine), with 24 cores.
- Project running in a dockerized container, on port 8000.
- Redis container on port 6379.

The project then opens port 80, and configures nginx as a reverse proxy, allowing incoming connections for specific domains to be relocated to the docker image containing the web server.

The reverse proxy currently redirects domain "sellin.one" to docker contained configured with port 8000:8000.

4.3 Account Based Functionality

When visiting the web app, the user will be redirected to the login/register page if not logged in and presented with options to access the system. This also includes a hCaptcha input field on both actions to stop bot accounts from spamming system resources.

4.3.1 Login

Users wanting to log in will be prompted for both username and password, as well as hCaptcha input before being sent to the backend. Shown in Figure 15. Requests are handled in the backend view function, where the system checks whether credentials are correct, and redirect to the home page if valid.

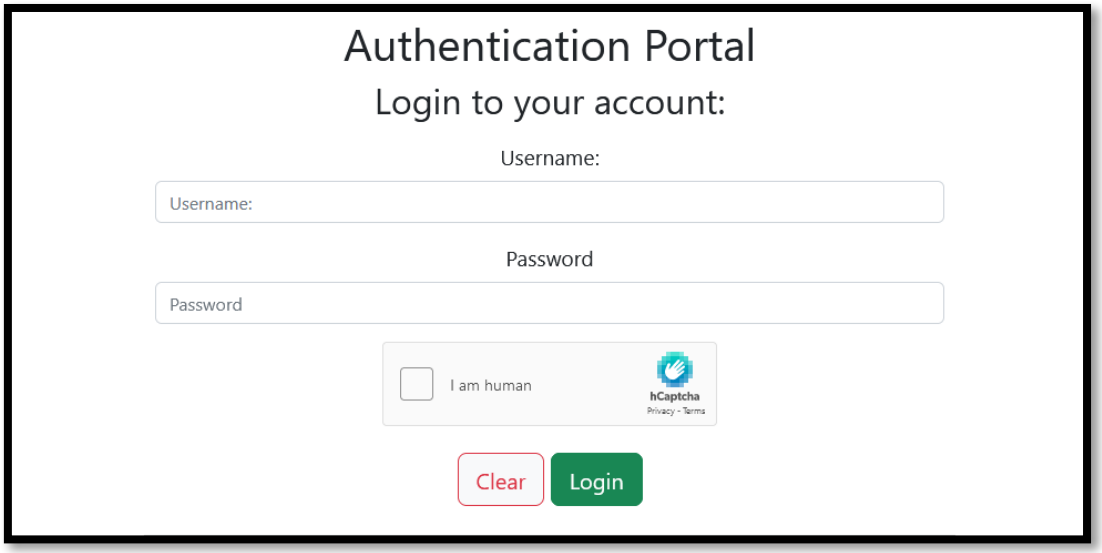


Figure 15 Login Action

4.3.2 Register

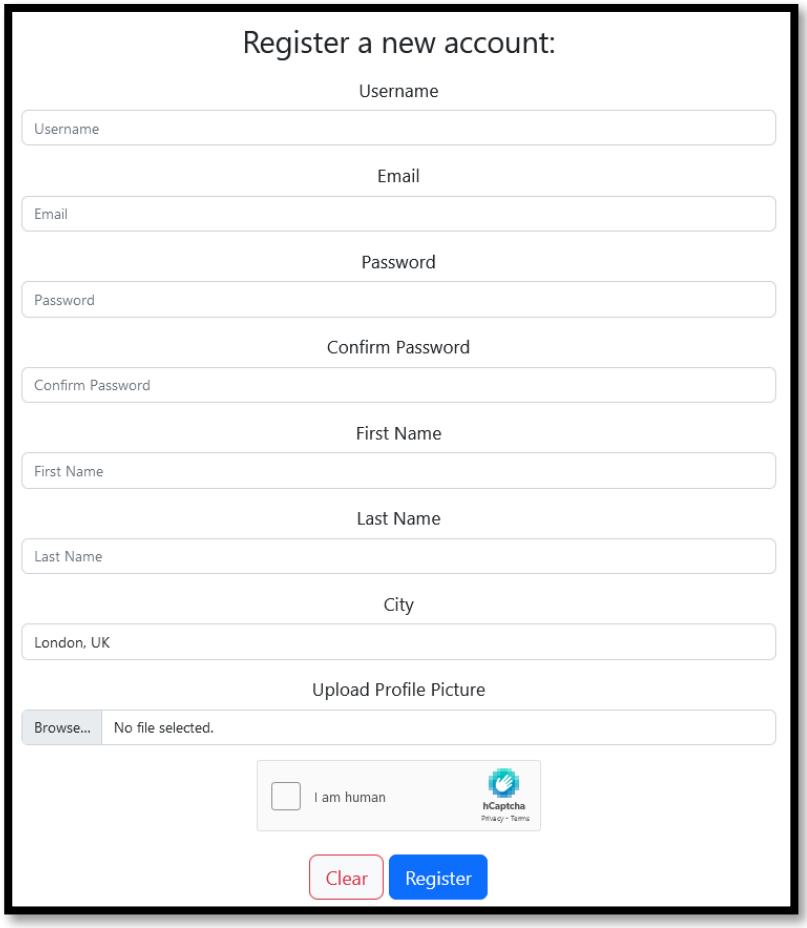


Figure 16 Register Account Form

Figure 16 shows the current register form, when users sign in successfully a User model object is created with association to 3 integration objects that are empty, including eBay, Etsy, and Bonanza as well associations to ProfileModel and login history. All of which are empty classes that are yet to be filled when the user interacts more with the system, this is shown in Figure 17.

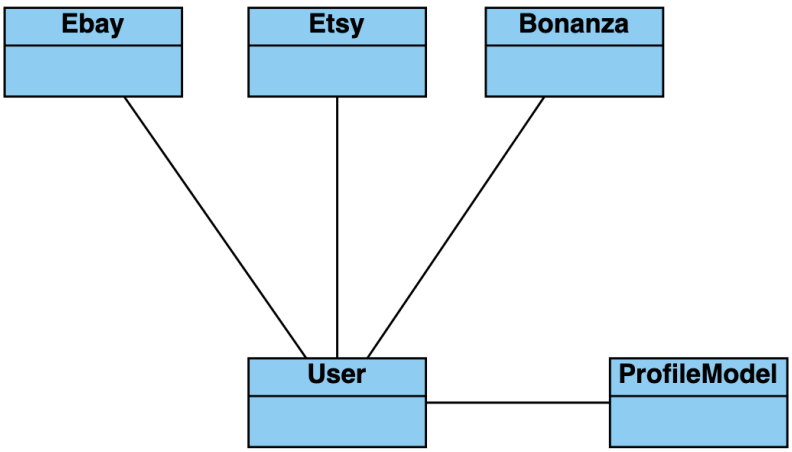


Figure 17 User Model with initial model mappings.

5.3.3 Adding and Removing Shipping Preference

A major part of all marketplace's listings requirements is a valid shipping location and return address, to remove the tedious action of users constantly adding addresses into every listing they create, shipping preferences is linked to the account which can be added and removed and allows users to select a preference when listing an item.

Adding Shipping Preference

Users can fill in the form adding information about their shipping preferences, which then can be later used. With additional option to add notes and whether it is the default shipping preference. Figure 18 shows adding shipping preference form.

Figure 18 Adding Shipping Preference

Removing Shipping Preference

Users will be able to view all their shipping preference list and delete ones that they no longer want to use. Figure 19 shows an example shipping address item.

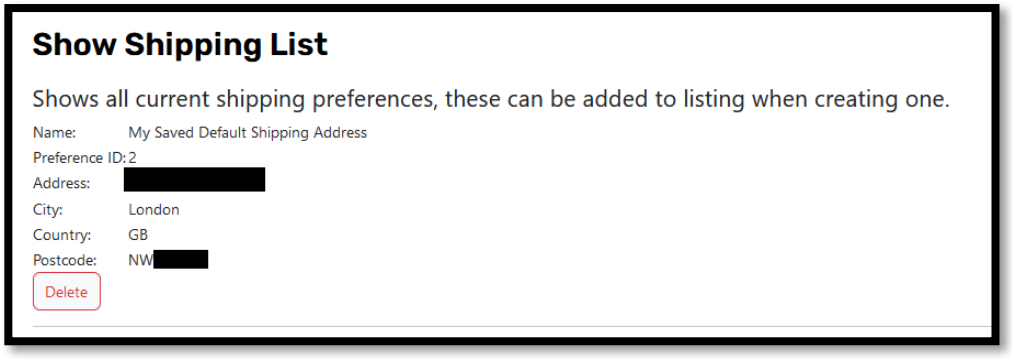


Figure 19 List of Users Shipping Preferences

4.3.4 Show Account Page

When logged into your account you can check out your account page by clicking on your account icon in the dashboard. Figure 20 shows account settings page.

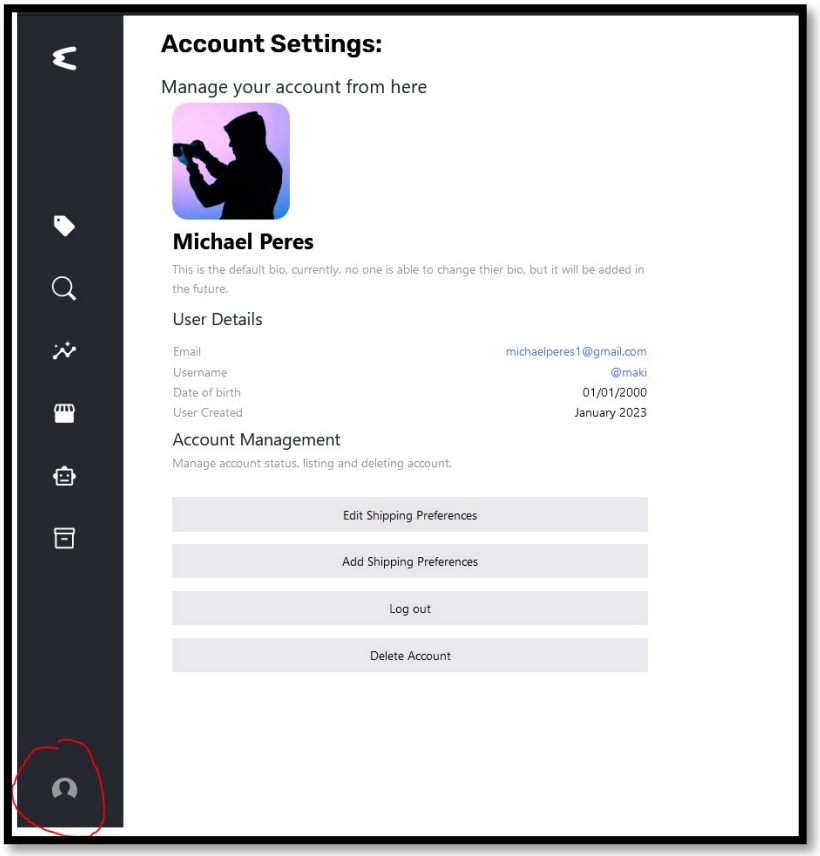


Figure 20 Profile Page

5.3.5 Delete Account

This allows users to delete their account via the account dashboard, the user is prompted beforehand just to make sure that this was the intended action.

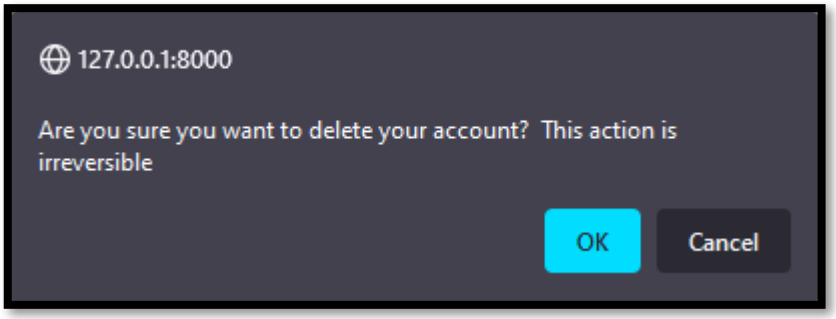


Figure 21 Confirmation of deletion.

Delete function, checks if the user is authenticated, and then deletes the user, and redirects to the account login/register page. All listings associated with that users are kept on their respective marketplaces.

5.3.6 Account Marketplace Integrations

When users initially create an account, to start uploading to marketplaces, they need to first login into their marketplace via their respective accounts and allow this application to act on behalf of them. The three functionality that are mentioned include:

- Showing all integrations
- Connecting and removing specific integrations.

Show All Account Integrations:

Users can check all currently supported integrations the system has and the status of each of their account integrations, with action options.

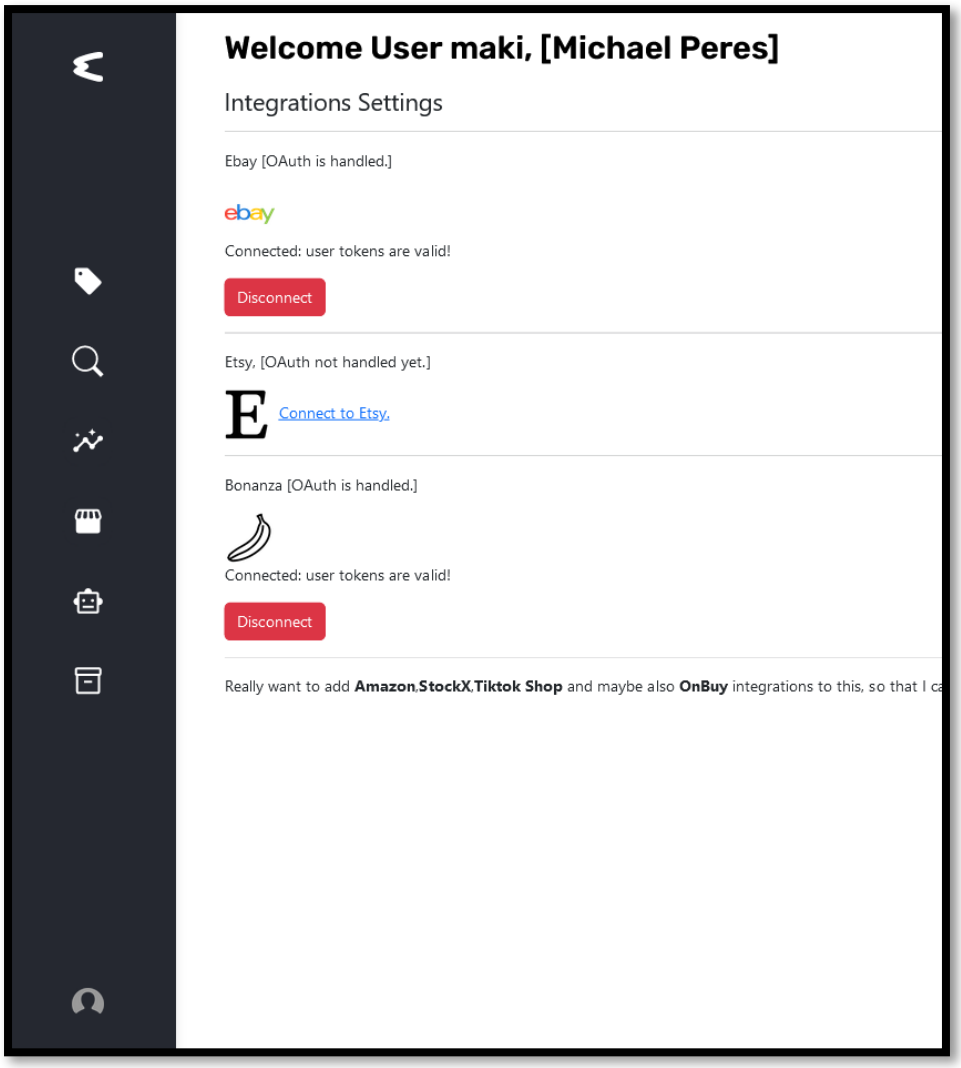


Figure 22 Showing Account Integrations Settings

Here this shows current integrations that users can connect and disconnect from.

Adding Integrations:

The 3 supported integrations currently are eBay, Etsy, and Bonanza. In all three marketplaces, the following steps take place:

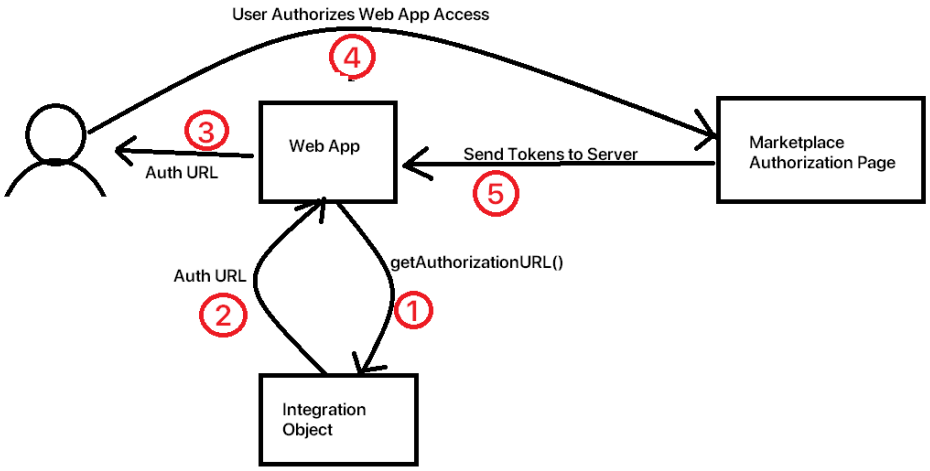


Figure 23 Application integration flow.

The server receives an authorization URL, that is send to the user. The user logs in and allows the app to perform actions on its behalf. After this, the marketplace will redirect the user back to the server where the success URL will be verified with things such as the state and an authorization token. The authorization token can be used to obtain a refresh and access token for that specific user.

For eBay, Bonanza and Etsy, the process is defined in Figure 24, 25, 26.

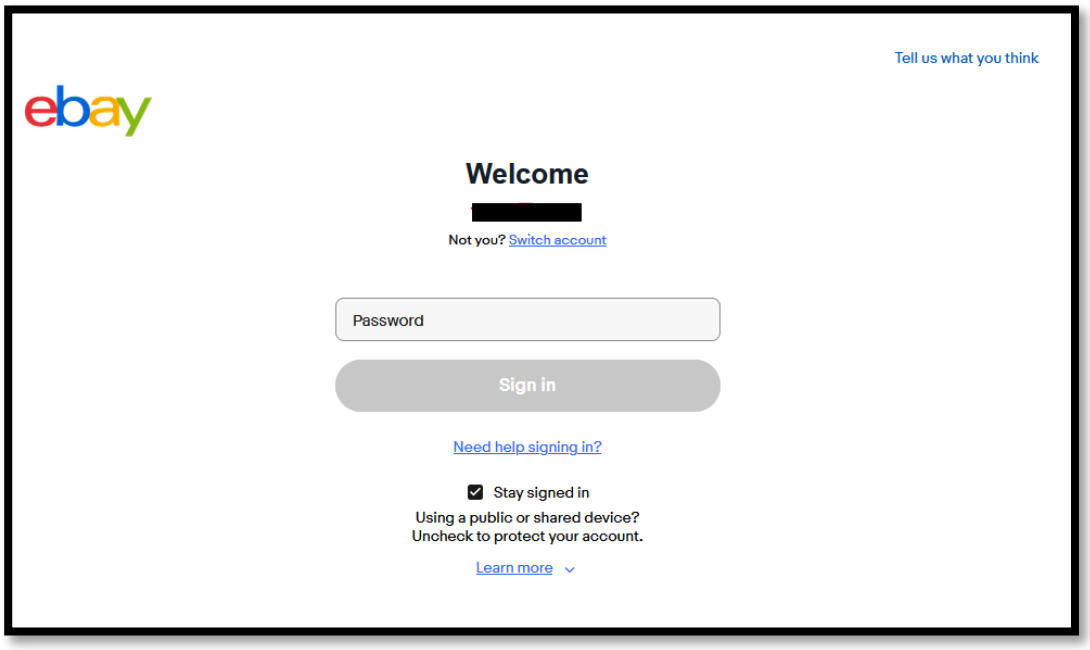


Figure 24 eBay Integration Login

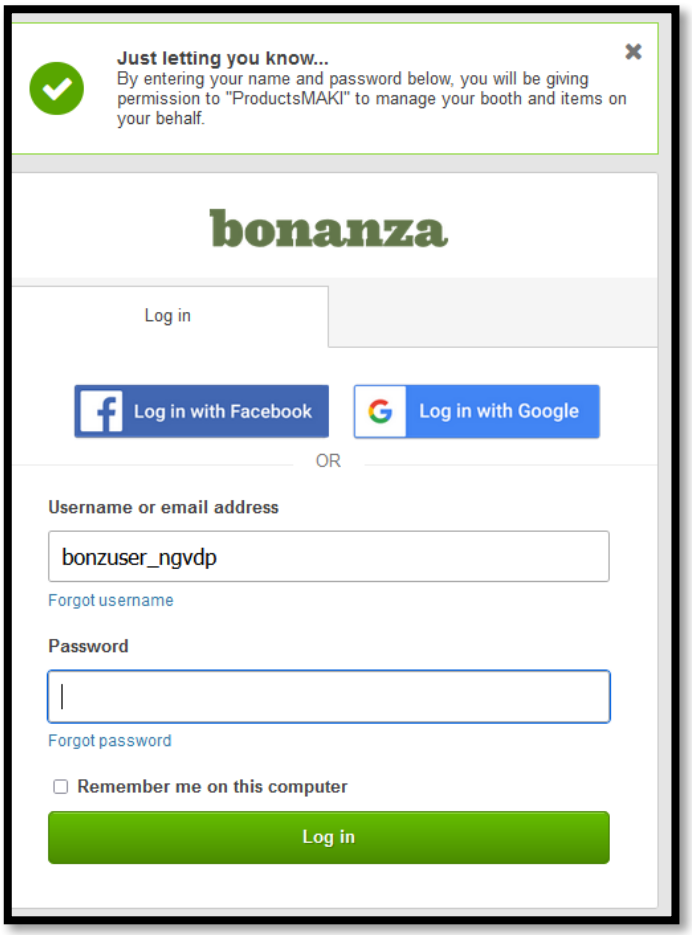


Figure 25 Bonanza Integration Login

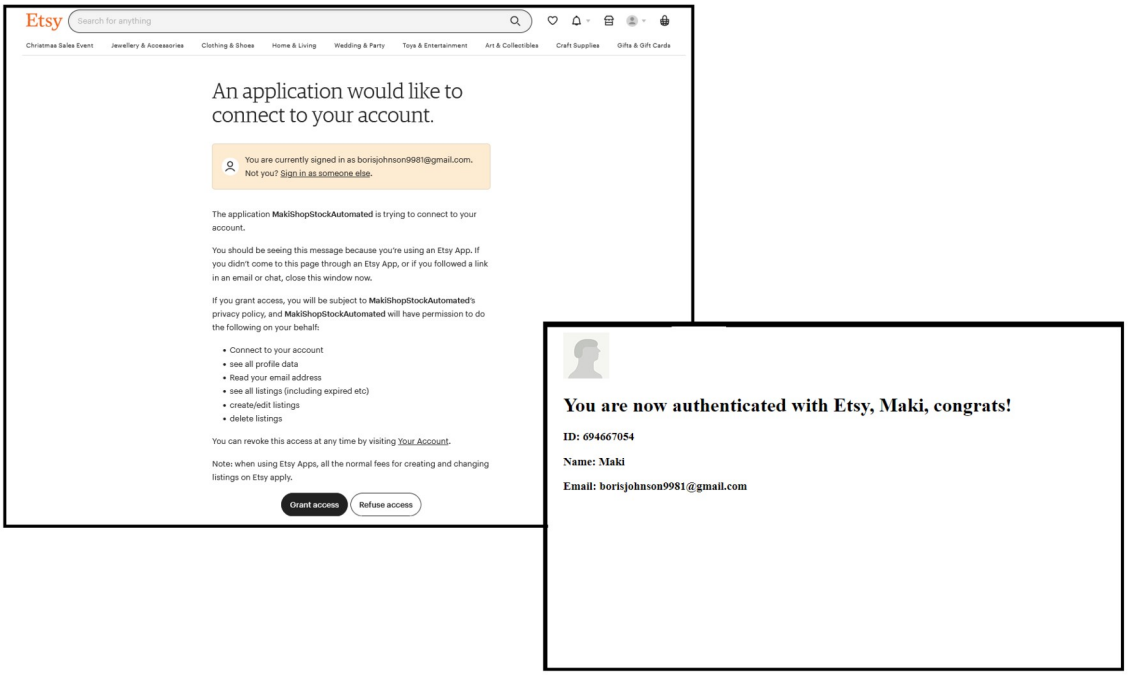


Figure 26 Etsy Integration Login and Testing

4.4 Listing Based Functionality

4.4.1 View Listing Management Dashboard

When users want to check their current progress of different listings, they can go to the dashboard and look at draft and active listings they currently have. The 2 possible states listings are active or draft.

- Draft state means no current integrations are connecting and associated with the listing.
- Active state means that at least one integration is currently associated with the listing and is listed on a marketplace.

Figure 27 illustrates the dashboard interface, it includes options to scroll through all listings, with title, inventory and price shown. Actions such as edit, delete, and create listings are accessible from the dashboard.

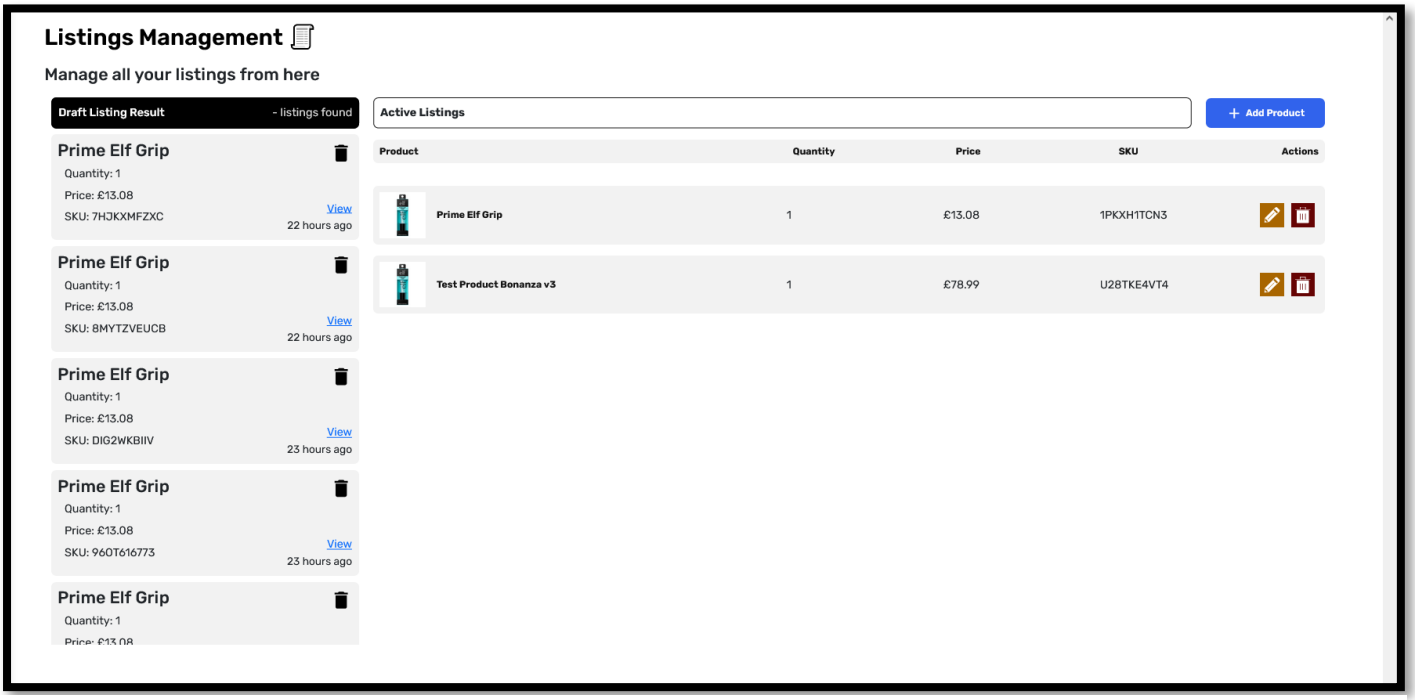


Figure 27 Listing Management Dashboard

4.4.2 Add Product Listing

Users who would like to list their product will arrive at this page shown.

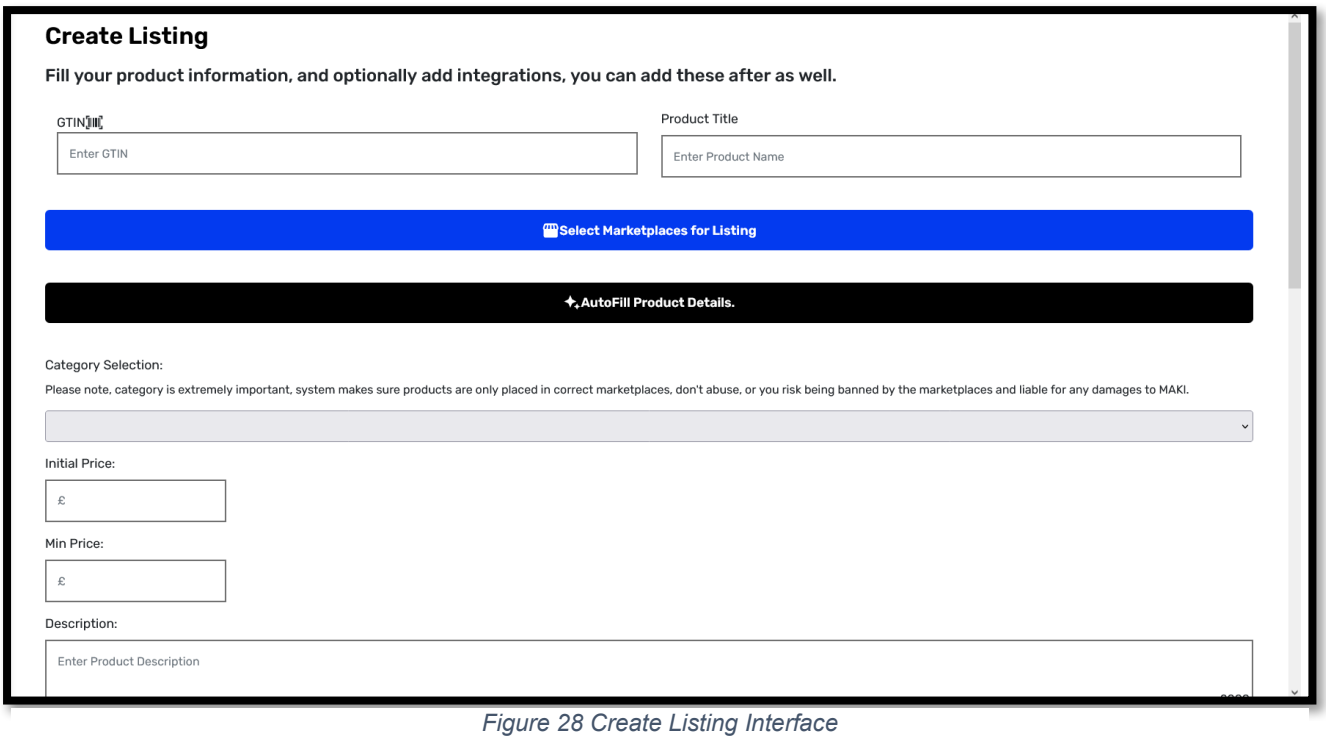


Figure 28 Create Listing Interface

The first input is a barcode, where a GTIN/EAN/UPC value can be inserted. The aim of this input type is to allow users to scan items and list products as fast as possible, as majority of the fields in Figure 28 are automated. Inputs are automatically filled by reusing parts of the metrics API and suggestions system.

When provided a barcode, data is automatically provided for:

- Selecting a category using an implemented Taxonomy solution, explained in Chapter 5.5.
- Product Title
- Max and min product price.
- Product description which is generated using the ChatGPT API with a prompt based on category suggested and the product name.

Based on whether product dimensions are available, postage can also be selected.

Figure 29 shows the remaining information required from the user in order to submit, which includes postage type, return window, images and quantity.

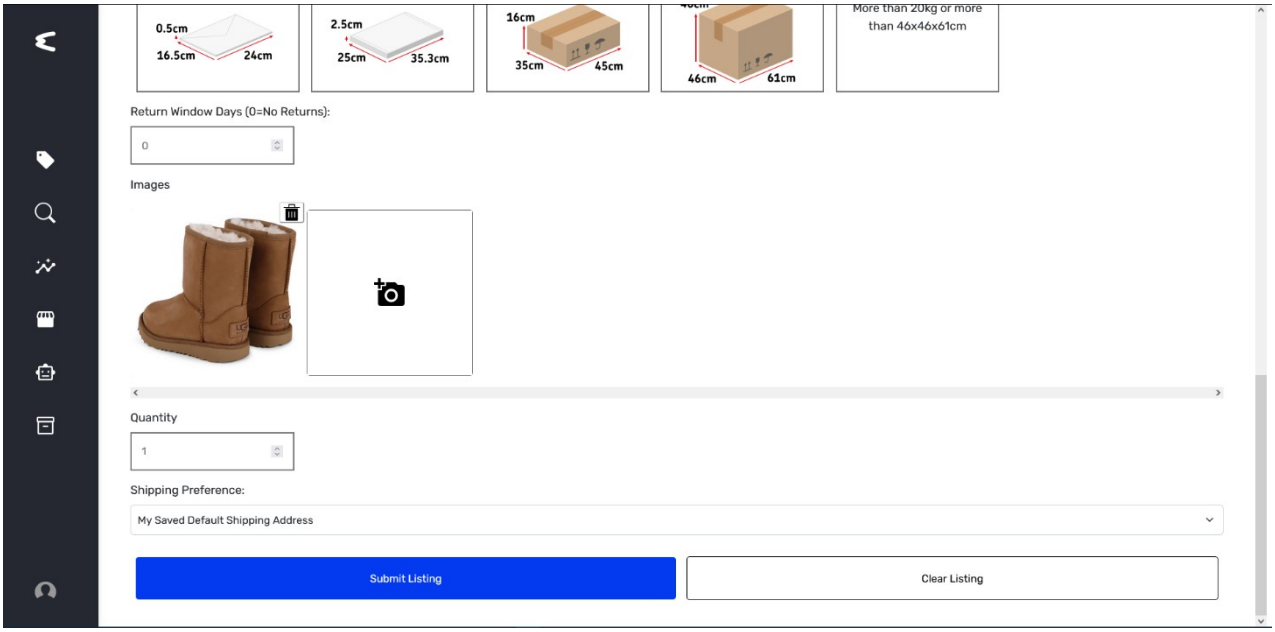


Figure 29 Remaining information required from user.

Figure 30 shows an example of data returned by the suggestion API while the user is inputting data in add product form..

```
{
  "title": "Suggested Product Title",
  "gtin": "1234567890",
  "init_price": "12.09",
  "min_price": "11.05",
  "description": "Obtained from EAN product data or ChatGPT if not available.",
  "product_data": {}
}
```

Figure 30 Example Suggestion JSON data output.

4.4.3 Delete Listing

When users want to delete a specific listing, they can do so in the listing dashboard, there are 2 different operations for deleting listings depending on the state it is in.

Draft listing deletion can be done by pressing the icon as shown below in Figure 31. As this means the listing is not linked to any marketplace, this will just delete the listing from the systems database and references to the user.

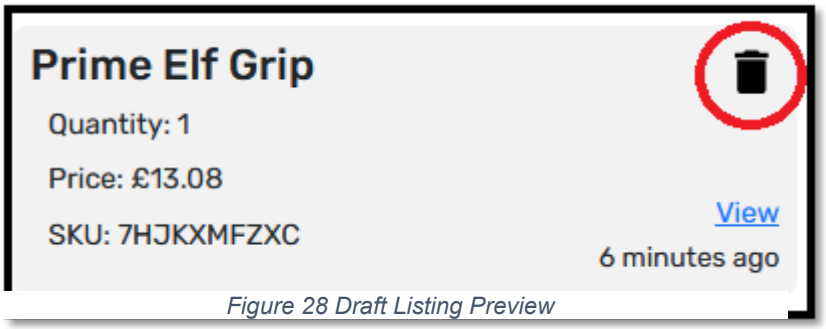


Figure 28 Draft Listing Preview

Figure 31 Draft Listing Deletion button.

Active listing is shown in Figure 32, deletion requires an additional step in the backend, which is dealing with integrations and handling errors. If deletion of a listing fails in one of the marketplace integrations, the system will stop the deletion process. This is because the users may face a situation where a product, they thought was deleted on the system sells on a marketplace.

Figure 32 Active Deletion button.

This app makes sure that deletion of a product only occurs if all marketplaces have successfully removed the product as shown in Figure 33.

```
if user == logged_in:
    if listing.user == user:
        del listing.integrations
        del listing
```

Figure 33 Pseudocode for deleting listing process.

4.4.4 Edit Listing Product Information

To edit a listing, a user can click on the edit icon within the listing dashboard. In the edit listing page, users can edit product information such as price, title, description, quantity, and images. If listing being edited is currently active on specific marketplaces, links of current locations the products are listed on will be highlighted like shown in Figure 34.

Figure 34 External Links in Edit Product page.

Figure 35 and 36 shows the layout of the edit interface and popup for editing the marketplaces the listing is on.

Figure 35 Form of Edit Listing Page.

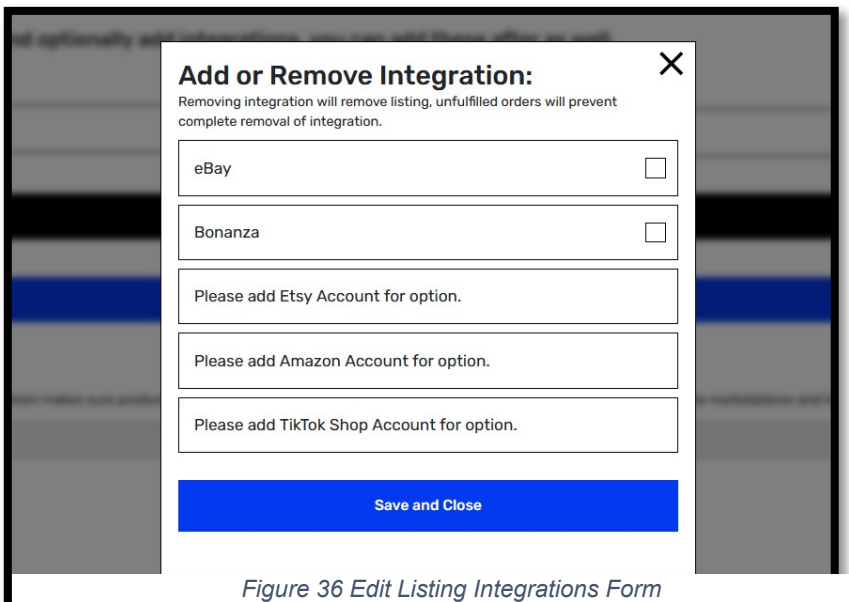


Figure 36 Edit Listing Integrations Form

4.5 Taxonomy, Category and Task Handling Functionality

Product Taxonomy refers to the categorizing of products based on types; most marketplaces store this information as an implementation of a tree structure. As this is the case, merging of category trees can occur algorithmically.

4.5.1 Maki Taxonomy and Algorithm (Own Implementation)

The main issue with automatically listing on multiple platforms is the handling of categories, marketplaces heavily rely on the categories in order to best serve content to buyers looking for it, especially if this is a new type of product. Within our system, adding a product is done through a selection input named product category in add listing interface. However, there may be hundreds of different ids/names for the same categories on different marketplaces. For example, eBay may have ID of 1 for Toys and Hobbies but Bonanza has the ID of 5 for the exact same category. A major job this system had to get right in order to work with multiple marketplaces is merging of different category trees together.

Issues arises when categories on these different platforms have slightly different names such as, “Films and TV” and “Movies and TV”. To solve this, 2 problems need to be overcome, these include:

- Recursive Tree Traversal, to find all categories.
- Mapping to other marketplace categories using Levenshtein distance to match categories.

Mapping Categories using Levenshtein Distance:

Levenshtein distance allows us to measure the definition differences between 2 sentences. For example, the word **Phone** and **Mobile** would have a very low score/distance as they are very similar in meaning compared to, **Phone** and **Car**. The project uses a library called Jellyfish that contains functions allowing it to calculate distance between words.

```
# Find distance score.
score = jellyfish.levenshtein_distance(ebay_name.lower(), bonanza_name.lower())

# We reward similar matches in literal string.
ebay_words = set([word.strip(string.punctuation) for word in ebay_name.lower().split() if
                          len(word.strip(string.punctuation)) > 1])
bonanza_words = set([word.strip(string.punctuation) for word in bonanza_name.lower().split(
                            len(word.strip(string.punctuation)) > 1])
identical_words = set(ebay_words).intersection(bonanza_words)

# If identical words then reward by decreasing score.
if identical_words:
    score *= 0.25

# Store the current best score as the ID of category.
if best_match_score is None or score < best_match_score:
    best_match_score = score
    print("Best match score: " + str(best_match_score) + " for " + str(ebay_name) + " and '


# Check if score is less than 5 then set as ID of category on system.
if best_match_score <= 5 :
    best_match_id = bonanza_id
```

Figure 37 Pseudocode for taxonomy merged based on Levenshtein distance.

This code also rewards inputs with identical words by multiplying the score by 0.25, as it is most likely to have the same meaning in category sense. If a score of less than 5 is present and the lowest, we can be certain it has the same meaning and is selected to be associated with a specific category ID of both marketplaces together. In Figure 38, an example is shown traversing each child category and obtaining the different scores possible.

```
Best match score: 24 for Mobile Phones & Communication and Collectibles
Best match score: 22 for Mobile Phones & Communication and Toys & Hobbies
Best match score: 21 for Mobile Phones & Communication and Books & Magazines
Best match score: 19 for Mobile Phones & Communication and Video Games & Consoles
Best match score: 4.25 for Mobile Phones & Communication and Cell Phones & Accessories
```

Figure 38 Output of Levenshtein distance at work.

The output of this function then produces 2 JSON files that we call Maki Taxonomy which is used within the metrics and listing creation functionality, an example is shown in Figure 39, where both IDs are obtained for category **Flag.**

Figure 39 Output of taxonomy merging.

4.5.2 Task Management and Scheduling

In this system, multiple users can be using the system at the same time, where each listing request could mean 10s of requests to marketplace API endpoints, although our Redis caching system tries to limit this, it is not always possible. If multiple users start sending these requests, it is possible for the server to easily be overwhelmed, for that reason, the webserver has a implemented task manager that runs in a background thread, that when receives tasks from users, such as list product on marketplace A, or edit listing, or delete listing, it is sent to the global task manager that takes tasks and run then asynchronously when possible, in essence putting them in a queue, this can control the number of processes running and how many requests are being send out at any given time.

4.6 Metric and Search Based Functionality

4.6.1 Search for Product Metrics based on user query.

Users can search products and obtain key metrics of the product, including,

- Most used colours in listing images.
- Price Ranges, Shipping Cost Range

- Number of sales within 7 days, month.
- Top Keywords used.
- Product History Line Graph
- Recent Sold Listings.

When a queried, these are all send through a WebSocket, this is because different pieces of data take longer to arrive than others.

Figure 40 Metrics Page Loading

Figure 41 Metrics Page Loaded.

We also can see in Figure 42, there is an upward trend in sales of iPhone 10 devices which can suggest that a major event occurred during 23rd April.

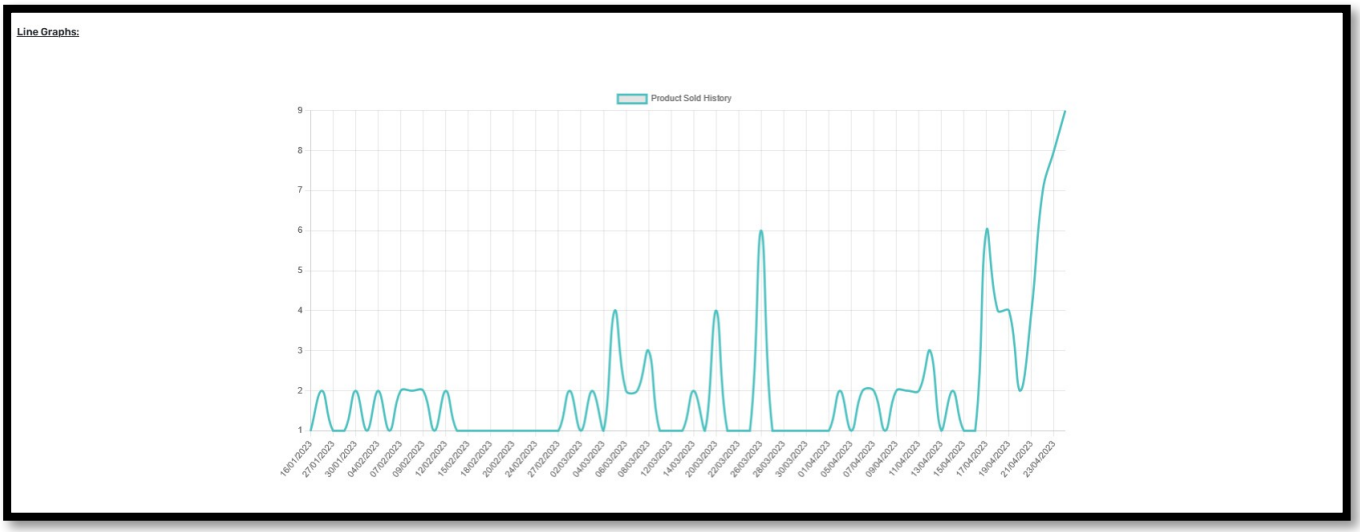


Figure 42 Sold Product History Graph and List View in Metrics Interface.

This is the network's calls in the background being made, so the webpage does not freeze while obtaining data. Implementing using WebSockets on both the backend and frontend.

4.6.2 Filtering Product Metrics, Outlier Removal.

Users can filter product search of metrics based on a range of filters, including item aspects, price, and condition, it is an optional form that appears when searching as a way of getting more accurate metric results.

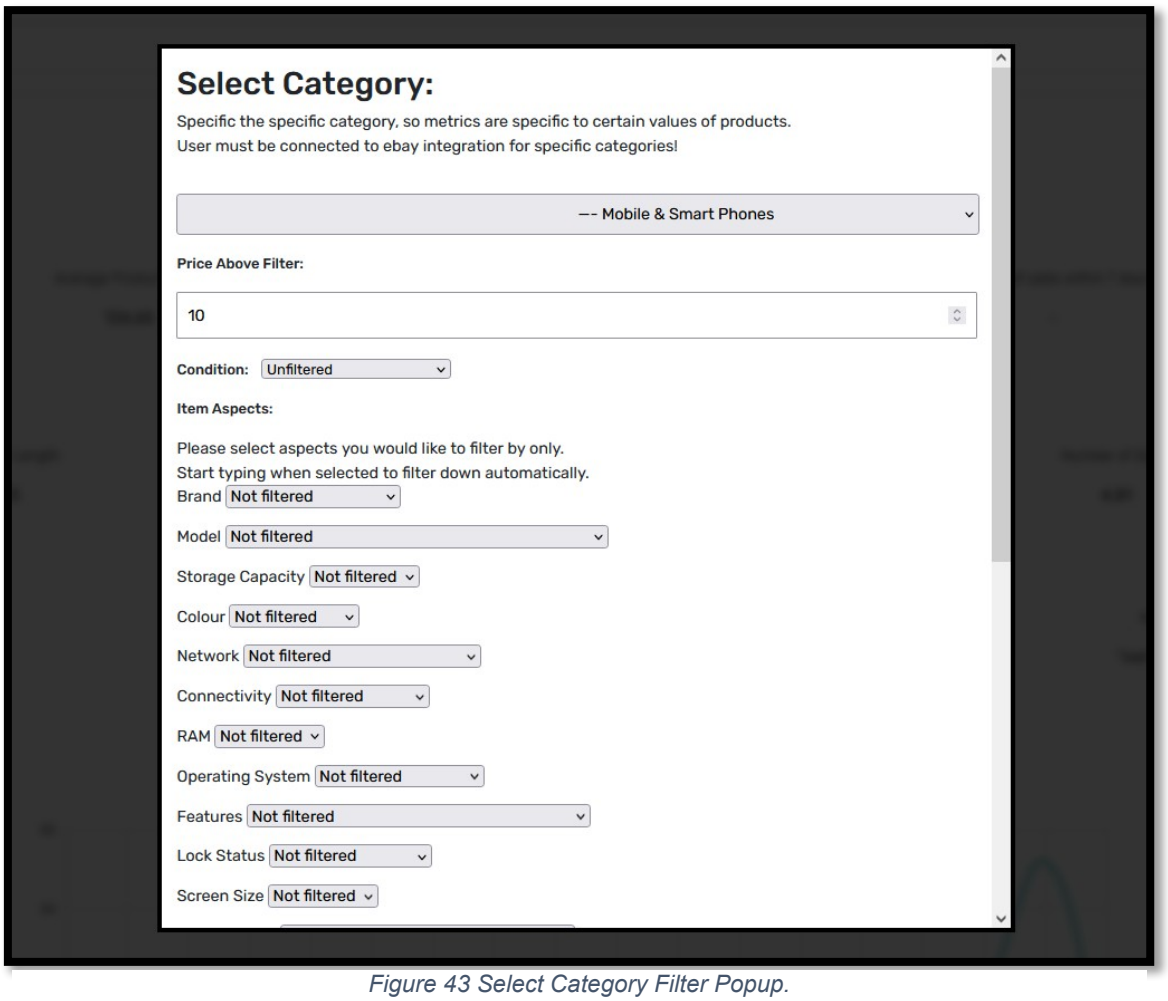


Figure 43 Select Category Filter Popup.

For metrics to be useful, we cannot allow outliers to distort overall range of a product, for example, an iPhone X should not have a product price range of £10 - £350, as we do not want to include the iPhone cases. We can remove this by using a bit of statistics and only collecting data, a max of two standard deviations away, this proved to work, and the second image shows the reduced-price range.

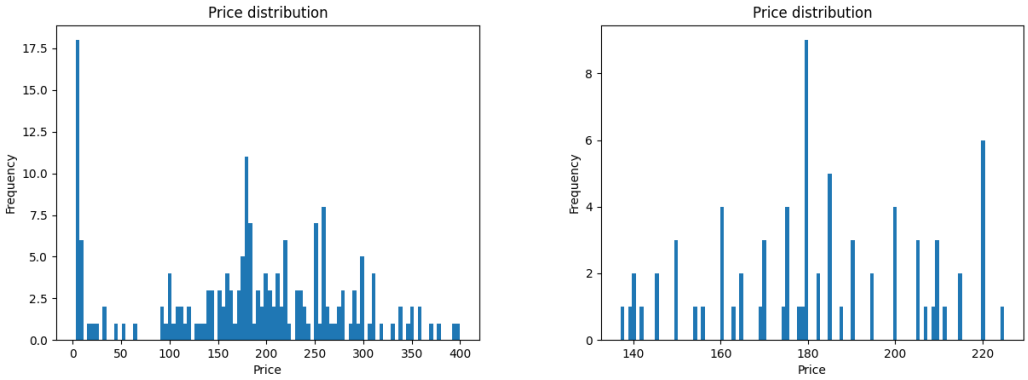


Figure 44 iPhone X price distributions before and after filtering outliers.

4.6.3 Metric Scraping Bots

The main data required from metrics features, is data that is not readily available, instead this data has been processed from aggregation of data collecting.

The project makes use of listings on eBay for data collection. To obtain completed/sold listings of a specific category and query, appending of specific URL arguments allow us to do this.

This project scrapes listing from the following URL:

Figure 45 Base URL of Scraped Webpage.

Here we receive a set of product listings, and we can extract all this information

Figure 46 Additional Parameters of Scraped Webpage.

and applying transformations to the data such as:

- Images processing, applying k clustering to obtain the five common colours and usage percentage, like the example output shown in Figure 47.

Figure 47 Example Average Colour pallet for iPhone 11

- Analysis of max and min product and shipping prices.

Other related metrics have been produced and is derived from data scraped using URLs in this format.

5.6.4 Product Search on all Marketplaces

Users can search for products more efficiently, by searching multiple marketplaces at the same time, and obtaining combined results from multiple streams of marketplaces. This is done by querying all marketplace class's function **get Listings(query)**

Users enter their product query into a form with selected filters to obtain listings, shown below.

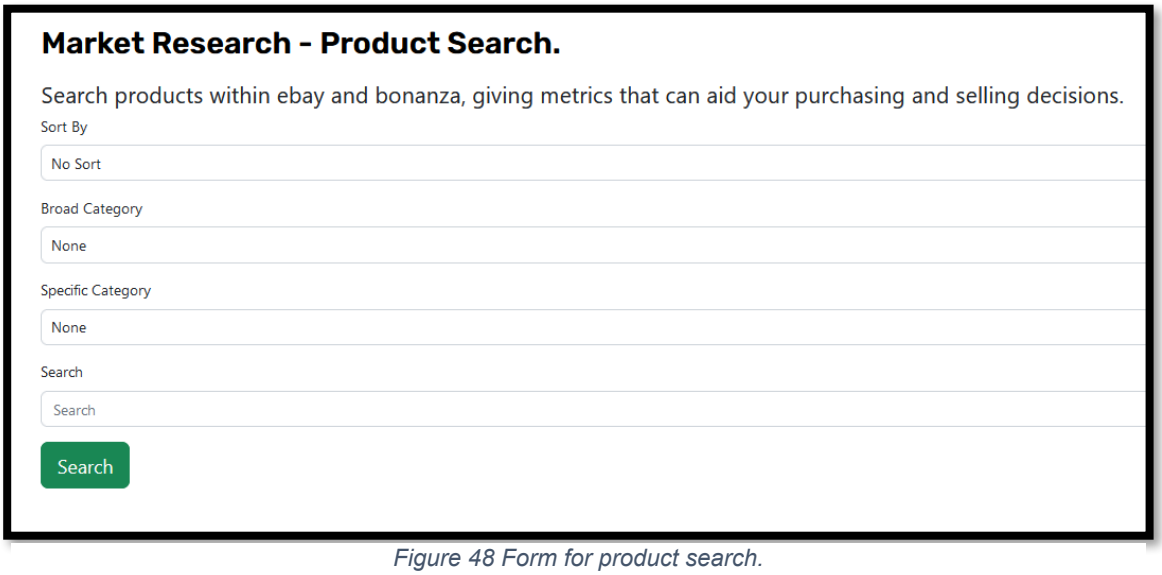


Figure 48 Form for product search.

The results for a search of "*Casio fx-991ex*" yields the following output.

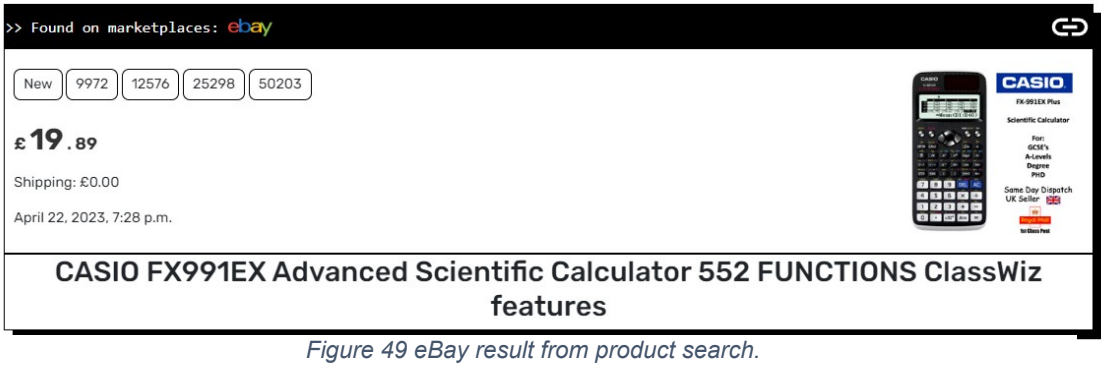


Figure 49 eBay result from product search.

4.7 Automatic Pricing (Theoretical)

Figure 29 Bonanza result from product search.

4.7.1 MLP Approach

If a system can scrape enough data from a marketplace like eBay, collecting specifically the average time of sale for a product based on its category and price, we can determine the price required to sell an item of the same specific category at any specific time using a multilayer perceptron.

Dataset Collection:

In order to train and test a model, we do need a dataset that can predict product price and time to sell of a product. Based on dependant variable of specific category type.

We use the eBay Buy APIs for obtaining data of listing creation. Using the same method shown in Section 5.6.3, we can scrape listings, obtaining listing URLs, shown in Figure 51.

https://www.ebay.co.uk/itm/354419422844

Figure 51 eBay example listing URL

Using eBay's **getItem** function call, we can get crucial information needed for the dataset, shown in Figure 52.


```
{
  "itemId": "v1|354536210098|0",
  "title": "Mielle Organics Rosemary Mint Scalp and Hair Strengthening Oil - 59ml",
  "shortDescription": "Prevents dryness of the scalp. Promotes hair growth with biotin in
  "price": {
    "value": "17.65",
    "currency": "GBP"
  },
  "categoryPath": "Health & Beauty|Hair Care & Styling|Treatments, Oils & Protectors",
  "categoryIdPath": "26395|11854|177660",
  "condition": "New",
  "conditionId": "1000",
  "itemLocation": {...},
  "image": {...},
  "size": "Standard Size",
  "brand": "Mielle Organics",
  "itemCreationDate": "2023-01-23T00:03:12.000Z",
  "seller": {...},
  "mpn": "30679",
  "epid": "21052490994",
```


Figure 52 JSON response of getItem API call

We now have a data source for training and testing the model. A dataset is created using all products currently found in a specific category. With table illustrated in Figure 53, number of hours to sell and the listing price.

Hours until sale time.	Sale Price

Figure 53 Example table layout for dataset.

After collecting this data, and presenting on a graph, the result would visually look like Figure 54.

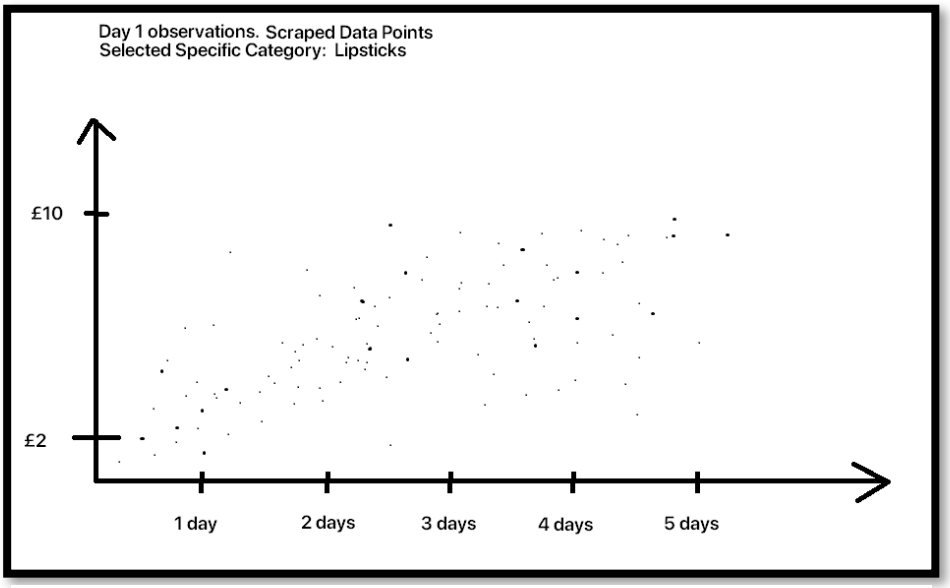


Figure 54 Collection of data within 2 dimensions, time to sell and price of product.

Defining and training the model:

To train the model the use of this architecture in PyTorch is created.

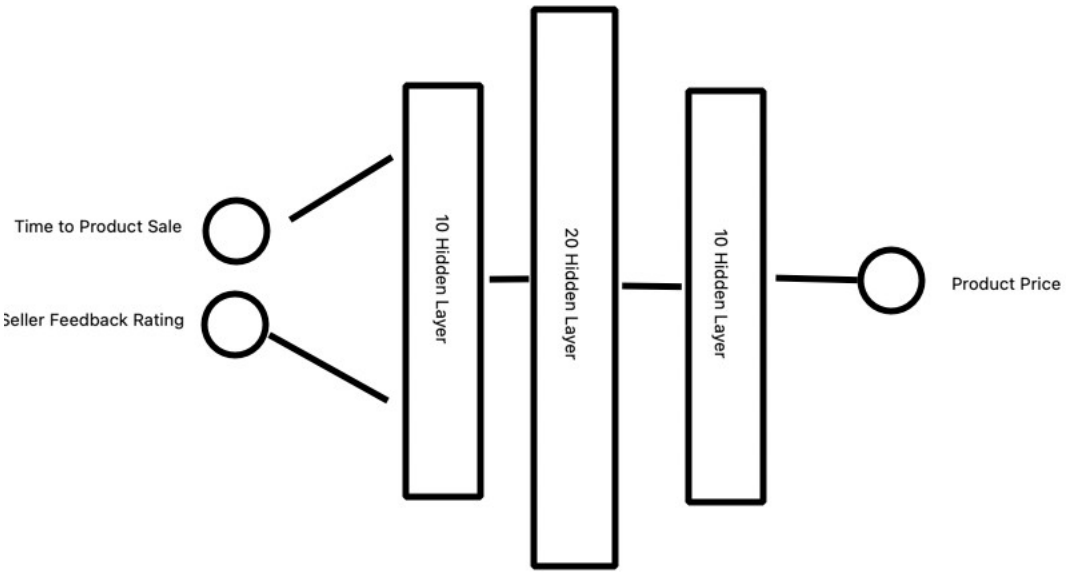


Figure 55 Model Architecture

Each layer uses ReLU activation function and a dropout before going to the next hidden layer, loss function used is Mean Square Error. The model will be trained to obtain the best price based on the time to product sale and seller feedback rating, this is illustrated with model architecture in Figure 55. With a code implementation in PyTorch shown for clarity in Figure 56.

```
class MakiModel(nn.Module):
    def __init__(self, dropout_rate=0.1):
        super(MakiModel, self).__init__()

        self.linear1 = nn.Linear(2, 10)
        self.linear2 = nn.Linear(10, 20)
        self.linear3 = nn.Linear(20, 10)
        self.linear4 = nn.Linear(10, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=dropout_rate)

    def forward(self,x):
        # Layer 1
        out1 = self.linear1(x)
        re_out1 = self.relu(out1)
        dre_out1 = self.dropout(re_out1)

        # Layer 2
        out2 = self.linear2(dre_out1)
        re_out2 = self.relu(out2)
        dre_out2 = self.dropout(re_out2)

        # Layer 3
        out3 = self.linear3(dre_out2)
        re_out3 = self.relu(out3)
        dre_out3 = self.dropoout(re_out3)

        # Layer 4
        out4 = self.linear4(dre_out3)
        return out4


optimizer = torch.optim.SGD(net.parameters(), lr=0.001)
loss = torch.nn.MSELoss()
```

Figure 56 Snippet of Model Implementation in PyTorch

Usage of Model for Repricing:

After applying MLP to train using data, we can try to fit a line that overly represents the model for this category, relationship between product price and the time to sell.

We can then determine that if we have been listing our product costing us 5£ it should take us 3 days to sell, however if it takes longer to sell, then decrease the price by the time delta that we were expected to sell at.

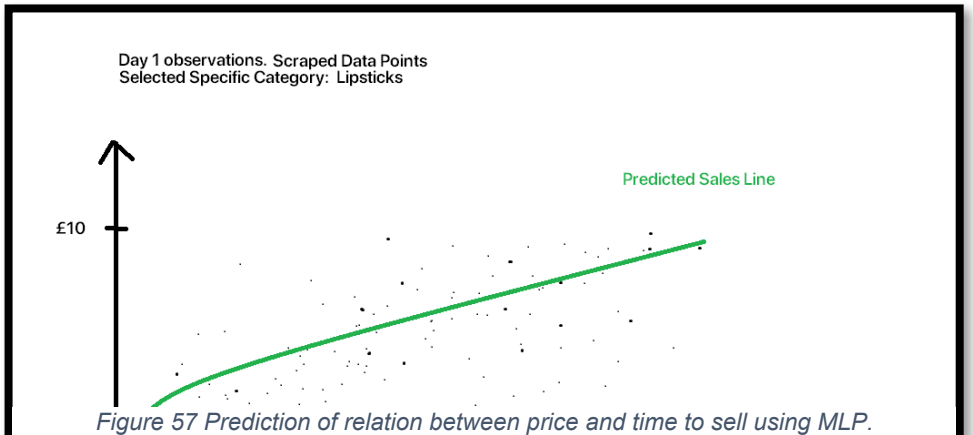


Figure 57 Prediction of relation between price and time to sell using MLP.

Here if we can predict the time it would take to sell an item, based on a price. Then given this predicted sales line, we could build a naïve auto-price updater that bases the specific time we would like to sell an item with different associated prices.

This approach is less prone to error, when products scraped are of the same type, and under the assumption the only major factor for selling products is the time it takes to find a specific buyer, which is not the only factor.

4.8 External Libraries

Third-party libraries used for the development of this project shown in **Appendix A.**

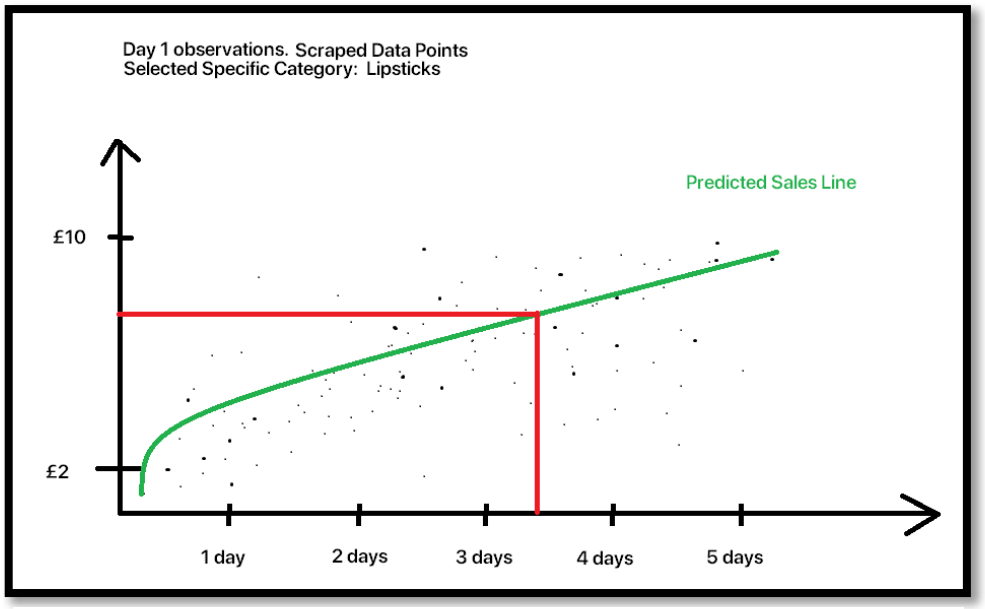


Figure 58 Using MLP predictions for selling products.

Chapter 5: **Testing & Evaluation**

5.1 Testing

5.1.1 Unit Testing

This tests that specific functions work as intended, there are 3 main sections that are tested, including tests within the groups of:

- Account Based Testing
- Listing Based Testing
- Metric/Search-Based Testing.

The tests can be found in **Appendix B**.

5.1.2 Security Testing

This web app is tested for security flaws using the guidance of the OWASP testing guide (OWASP, 2020), for most effectiveness. We will demonstrate major security attacks for testing the system.

- **SQL Injection Attempts**
  "Django query sets are protected from SQL injection since queries are constructed using query parameterization"[5] and since we did not use any raw SQL queries and only ORM provided, this passes the tests. Fortunately for us, login fails and notifies the user the password is incorrect with inputs shown in Figure 59.

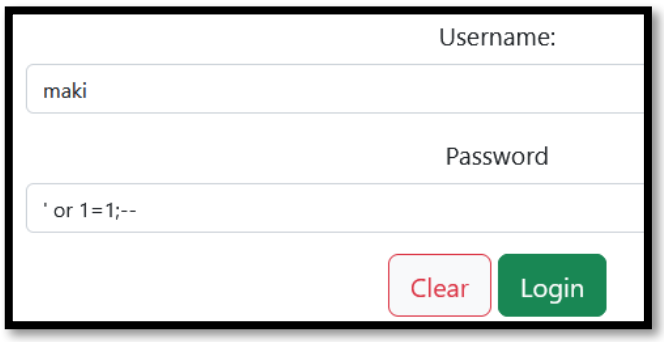


Figure 59 SQL Attempt on Login.

- **Hashed User Passwords**

  All passwords created and associated with a user is hashed with SHA256. So even if the system is compromised, resulting users' passwords be harder to crack compared to plain text.

- **Cross-Site Scripting (XSS)**
  By default, all pages and content created is only viewable by the user that creates it, the system follows the Discretionary Access Control model, where subjects can determine who is able to see their own objects. However, we will test this anyway, an input to form is shown in Figure 60.

[5] Databases in Django, htt bases/

Figure 61 Output of XSS attempt on HTML content.

The result of such input is shown in Figure 61, XSS failed and thus successfully passes the XSS test.

- **CSRF**

  CSRF tokens are used within forms, such as login and register so that we can confirm requests are coming from our system, the system also has set up Allowed Origins of hosts that can connect to itself. Figure 62

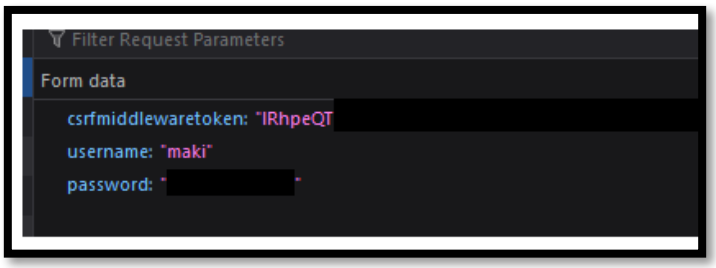


Figure 62 Login Request sends a CSRF token.

  shows an example request from a user with the **csrfmiddletoken** passed.

- **Hcaptcha Input**

  To create an account, users must pass a hCaptcha, which confirms they are not a bot account creating an account and stops abuse on the site. This works by obtaining a success token from the user and comparing it with hCaptcha authentication server.

5.2 Evaluation against project objectives and requirements.

To determine whether the project has been successful, we will analyse whether objectives mentioned in section 1.5 of the report has been completed. Based on Figure 63, the project achieves all objectives, and solves the problem outlined in Chapter 1.

Objectives	Outcome	Report Section
1. To analyse and contrast current approaches to identify key characteristics and limitations of these platforms.	Achieved	Chapter 2 – 2.2 Existing Applications
2. Create a platform that allows users to oversee their listings on multiple marketplaces.	Achieved	Chapter 5 – 5.4 Listing Based, 5.3.6 Account Marketplace Integrations
3. Create functionality that grants users CRUD operations on their listings across these marketplaces.	Achieved	Chapter 5 – 5.4.1, 5.4.2, 5.4.3 5.4.4
4. Allow users more insights into product performance on different marketplaces with ability to investigate key metrics and marketplace insights.	Achieved	Chapter 5 – 5.6 Metric and Search Based
4. Creation of a selling agent that deals with the automatic repricing of products in relation to market conditions and sellers.	Theoretically Achieved	Chapter 5 – 5.7 Automatic Pricing

Figure 63 Objectives Achieved Table

5.2.1 Functional Testing

Requirement ID	Outcome	What section of Chapter 5 shows the evidence?
F1	Achieved	5.3.2
F2	Achieved	5.3.1
F3	Achieved	5.3.5
F4	Achieved	5.3.6
F5	Achieved	5.3.6

F6	Achieved	5.4.2, 5.4.4
F7	Achieved	5.4.2, 5.4.4
F8	Achieved	5.4.2
F9	Achieved	5.3.3
F10	Achieved	5.4.2, 5.4.4
F11	Achieved	5.4.2
F12	Achieved	5.4.4
F13	Achieved	5.4.3
F14	Achieved	5.4.1
F15	Achieved	5.4.1
F16	Achieved	5.4.4
F17	Achieved	5.4.3
F18	Achieved	5.4.4
F19	Achieved	5.6.1, 5.6.2
F20	Achieved	5.3.2, 5.4.2, 5.4.4
F21	Achieved	5.6.3
F22	Partially Achieved	5.3.4, 5.3.3
F23	Achieved	5.4.2
F24	Not Achieved	-
F25	Achieved	5.3.2
F26	Not Achieved	-

Most of the requirements were completed, and the corresponding evidence is illustrated in the third column. Not Achieved were non-core functionalities.

5.3 Legal, Social, Ethical and Commercial Issues

A discussion of current issues with the project in the following regions and what would need to be changed in the system to be allowed to be commercially deployed is mentioned here.

5.3.1 Legal and Policies

With the involvement of many marketplaces in this project, it is no surprise lots of legal factors come into play.

Marketplace Logos:

All marketplaces specifically request the use of logos that are provided from their website only, however, these are not suitable, as the project requires only small icons.

Trademark Policies:

The marketplaces that this project integrates with require a disclaimer that informs the user that this project is not endorsed or certified by a marketplace. This is to avoid confusion; however, this hasn't been made clear in any of the documents within the project.

Lack of Infrastructure:

Marketplaces require this project to not display listing content more than 6 hours old than the information on respective marketplace sites. The project does not depend on marketplaces as their source of data, and so the project infrastructure needs to be altered.

Security Policies:

All marketplaces currently working with the application require (2 Factor Authentication) 2FA for account security and an incident response plan. This project does not contain these requirements but is necessary if the project is used in the public domain.

5.3.2 Ethical and Privacy

Usage of Bots/Scrapers:

The project makes use a multitude of web scrapers that use marketplace websites in order to obtain metrics and suggestion data for specific products. This may be against marketplace terms and usage policies and may cause the site to handle requests of bots instead of genuine human users. This is unethical as it is restricting access to the site using bots.

Privacy Issues:

Marketplaces like eBay require personal information that's present in accounts to be deleted from the system when it's deleted-on eBay.

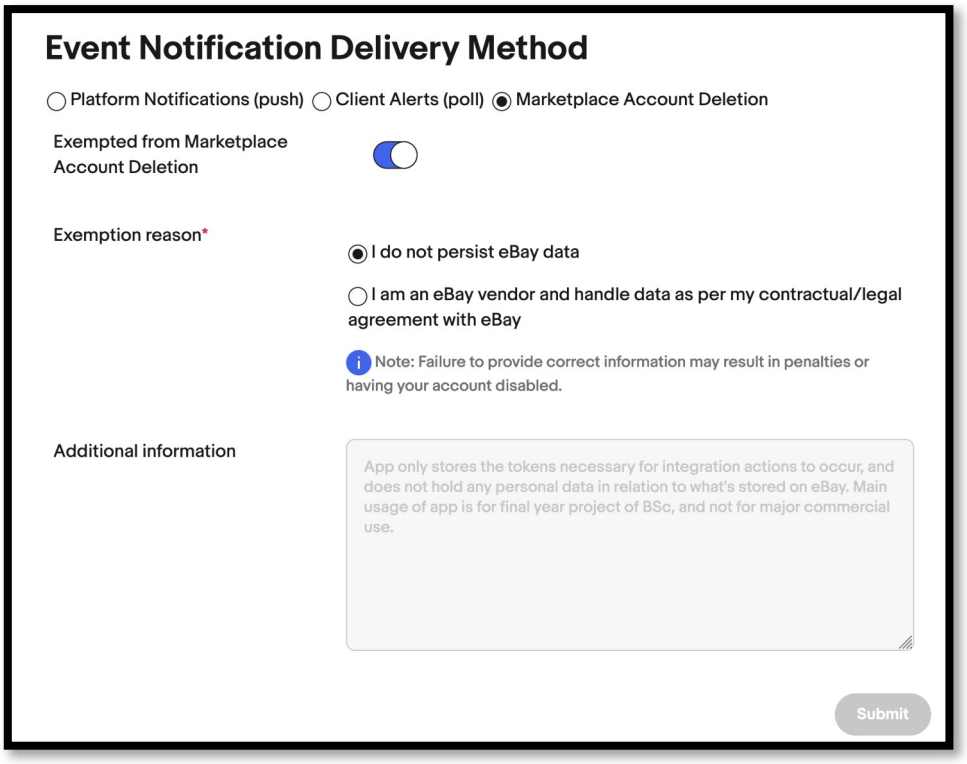


Figure 64 eBay requiring marketplace account deletion.

This policy is mandatory if the project requires usage of personal information from the user, such as automatically obtaining the users addresses. This is not handled as this app is in development and has been exempt as shown in Figure 64.

General Data Protection Regulation (GDPR) Requirements:

GDPR requires the project to only keep data for no longer than necessary, and include protection measures against unauthorised processing, access, loss, destruction, or damage (GOV.UK, 2018). Currently the project, deletes all data associated with a user such as shipping addresses and listings when account deletion occurs. Adequate security testing has been performed for preventing unauthorised access to the system. However, the database consists of one file that can be obtained by anyone that has physical access to the local machine, to solve this, an encrypted database file, in a cloud setting may allow for better security than a university student PC.

5.3.3 Environmental and Commercial Factors

Commercial Fees:

Charging of fees for specific parts of the app are dependent on the policies set by marketplaces, marketplaces like Etsy, require no fee is charged for accessing part of the application "that integrates with the API and that Etsy provides members free of charge.

Chapter 6: **Conclusion**

This project has ignited a new interest in ecommerce and operations, and I will carry on improving this platform over the summer starting with the future implementation section. Based on my objectives, I am amazed of how much work I managed to do, and thankful for my supervisor Mustafa's aid.

6.1 Achievements

This project was an extremely exciting and successful journey. I fulfilled the objectives by:

- Creating a platform allowing connectivity between marketplaces.
- Users could check insightful metrics not found on any marketplace to aid their decisions.
- A theoretical protocol for price changing and monitoring was mentioned.

I learnt a lot about different marketplaces, the systems they use and put myself into the world of automated e-commerce. This project has also introduced me to OAuth flows and the importance of good software architecture in developing maintainable and understandable code throughout a three-month period.

6.2 Limitations

Due to my initial choice of functionality over aesthetics, a frontend library like React or Next JS was not used, this made it much harder to create functionality in the frontend to support backend capabilities. This choice did lead to limitations in the work made on the frontend as functions already handled by React and Next had to be created in Vanilla JS.

Secondly, my initial lack of knowledge in neural networks before starting the project made it hard to initially think of achievable or implement solutions to the repricing problem which later uses an MLP as part of the solution.

6.3 Future Work

The implementation currently created for this solution is far from complete, with a lot more work needed to make it fully functional, including:

1) Marketplace Integrations: To increase the visibility of products, addition of more marketplace integrations like TikTok shop, and Amazon will be supported.

2) Integration Model Layout: Currently Integrations is an abstract class, and so user object holds direct one to one mapping with all integration classes, I would like to change Integration to a concrete base model, so that all integrations are referenced by it, and users do not require the direct mappings to integration classes.

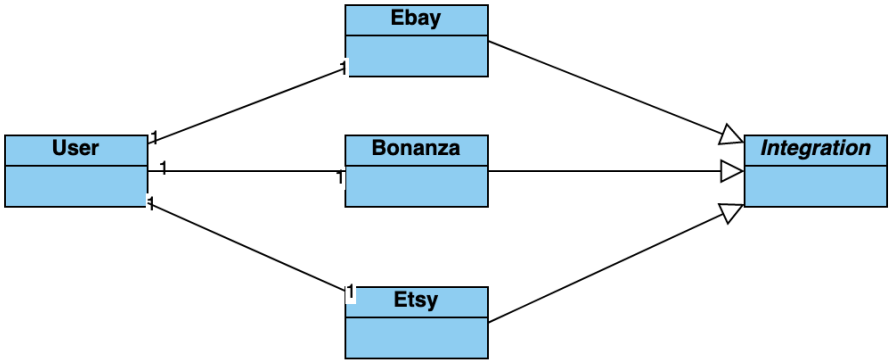


Figure 65 Current Integration Model Solution

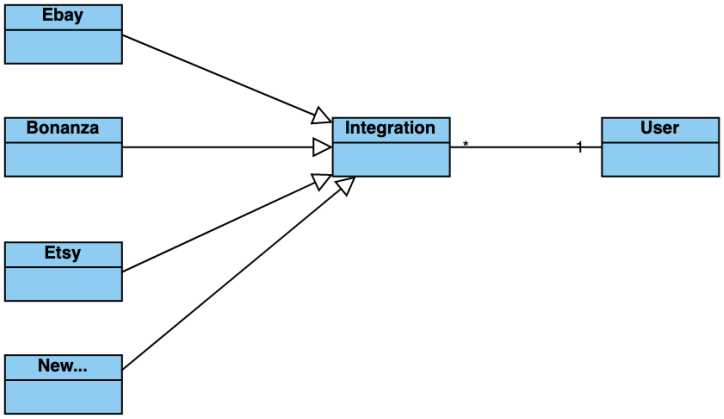


Figure 66 Proposed Integration Model Solution

3) Automatic Category Selection: Usage of user data to train a machine learning algorithm to determine categories based of product titles, and other information like product description. Allowing for automatic category selection without reliance on any marketplace API.

4) Abstraction of Listing and Product models: Currently, if a user wants to input a new listing, regardless of if the product is already known, they need to fill in all this information, if the system had a abstraction, so stored product data from sources like EAN, and can reference it when listing products, we can effectively limited required data for listing.

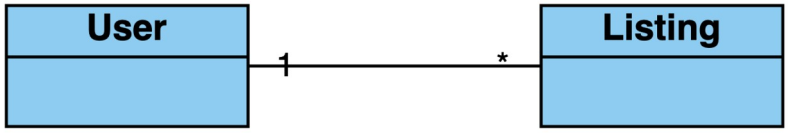


Figure 67 Current Model to Listing Model.

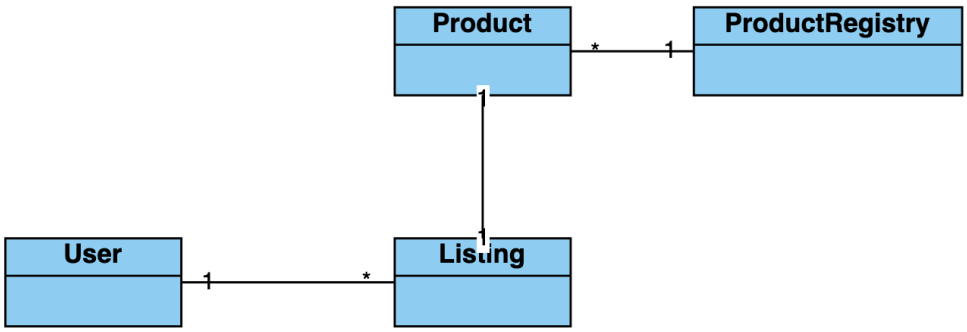


Figure 68 Proposed Solution to Listing Model.

5) Order and Returns Handling: If listing is ordered, then the system does not notify the user, and there is not currently any state for the listing.

Appendix

Appendix A: Third Party Library Imports

```
aiohttp==3.8.4
aiosignal==1.3.1
anyio==3.6.2
appdirs==1.4.4
asgiref==3.6.0
async-timeout==4.0.2
asyncer==0.0.2
attrs==22.2.0
autobahn==22.12.1
Automat==22.10.0
beautifulsoup4==4.11.2
bleach==6.0.0
boto3==1.26.89
botocore==1.29.89
bs4==0.0.1
certifi==2022.12.7
cffi==1.15.1
channels==4.0.0
channels-redis==4.0.0
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.6
coloredlogs==15.0.1
constantly==15.1.0
contourpy==1.0.7
cryptography==38.0.4
cssselect==1.2.0
cycler==0.11.0
daphne==4.0.0
Django==4.1.4
django-bootstrap5==22.2
django-cors-headers==3.13.0
django-crontab==0.7.1
django-hcaptcha-field==1.4.0
django-ipware==4.0.2
django-multiupload==0.6.1
django-ratelimit==4.0.0
django-simple-history==3.2.0
django-storages==1.13.2
docker==6.0.1
fake-useragent==1.1.1
fastapi==0.87.0
filelock==3.9.0
filetype==1.2.0
flatbuffers==23.1.21
fonttools==4.39.3
forex-python==1.8
frozenlist==1.3.3
h11==0.14.0
h2==4.1.0
hpack==4.0.0
huggingface-hub==0.12.1
humanfriendly==10.0
hyperframe==6.0.1
hyperlink==21.0.0
```

```
idna==3.4
ImageHash==4.3.1
imageio==2.26.0
importlib-metadata==6.0.0
incremental==22.10.0
itsdangerous==2.1.2
jellyfish==0.11.2
Jinja2==3.1.2
jmespath==1.0.1
joblib==1.2.0
keybert==0.7.0
kiwisolver==1.4.4
llvmlite==0.39.1
lxml==4.9.2
markdown-it-py==2.2.0
MarkupSafe==2.1.2
matplotlib==3.7.1
mdurl==0.1.2
mpmath==1.2.1
msgpack==1.0.4
multidict==6.0.4
networkx==3.0
nltk==3.8.1
numba==0.56.4
numpy==1.23.5
onnxruntime==1.13.1
openai==0.27.4
opencv-python==4.7.0.72
opencv-python-headless==4.6.0.66
packaging==23.0
parse==1.19.0
Pillow==9.3.0
pipdeptree==2.5.2
pooch==1.6.0
priority==1.3.0
protobuf==4.22.0
psycopg2==2.9.6
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.21
pydantic==1.10.5
pyee==8.2.2
Pygments==2.14.0
PyMatting==1.1.8
pyngrok==5.2.1
pyOpenSSL==22.1.0
pyparsing==3.0.9
pyppeteer==1.0.2
pyquery==2.0.0
pyreadline3==3.4.1
python-dateutil==2.8.2
python-dotenv==0.21.0
python-multipart==0.0.6
PyWavelets==1.4.1
pywin32==305
PyYAML==6.0
redis==4.4.0
regex==2022.10.31
rembg==2.0.30
requests==2.28.1
requests-html==0.10.0
rich==13.3.1
```

```
s3transfer==0.6.0
scikit-image==0.19.3
scikit-learn==1.2.1
scipy==1.9.3
sentence-transformers==2.2.2
sentencepiece==0.1.97
service-identity==21.1.0
simplejson==3.18.0
six==1.16.0
sniffio==1.3.0
soupsieve==2.3.2.post1
sqlparse==0.4.3
starlette==0.21.0
sympy==1.11.1
threadpoolctl==3.1.0
tifffile==2023.2.28
tokenizers==0.13.2
torch==1.13.1
torchvision==0.14.1
tqdm==4.64.1
transformers==4.26.1
Twisted==22.10.0
twisted-iocpsupport==1.0.2
txaio==22.2.1
typing_extensions==4.4.0
tzdata==2022.7
urllib3==1.26.13
uvicorn==0.20.0
w3lib==2.1.1
watchdog==2.1.9
webencodings==0.5.1
websocket-client==1.5.1
websockets==10.4
weight-converter==0.0.3
Werkzeug==2.2.3
whitenoise==6.4.0
wincertstore==0.2
xmltodict==0.13.0
yarl==1.8.2
zipp==3.15.0
zope.interface==5.5.2
```

Appendix B: Unit Testing

Account Based Testing:

Screen	Test	Expected Output	Result
Sign Up	Password and confirm password are not the same.	User will be redirected to sign up page with error notifying passwords do not match.	Success
	Username is already taken.	User will be redirected to sign up page with error notifying username is already taken.	Success
	Username or Password or First/Last name inputs are less than 2 characters in length.	Validation error notifying users' input must be longer than at least 2 characters long.	Success
	User enters valid information into the system.	User is redirected to home page and logged in.	Success
Login	Username and password must be at least 2 characters long.	Validation error notifying users' input must be longer than at least 2 characters long.	Success
	Username does not exist.	Validation error notifying username does not exist.	Success
	Incorrect password given	Validation error notifying password is incorrect for username.	Success
	Correct credentials given and login pressed.	User is redirected to home page and logged in.	Success
Shipping Preferences	Input fields for shipping preferences.	Shows newly created shipping preferences in list.	Success
	Delete a selected shipping preference.	Deletes the shipping preference from the user.	Success
Show Account Page	Select account page while logged in.	Presents the account page.	Success
	Select account page while not logged in.	Presents account login page, to continue	Success
Delete Account	Delete Account	Will delete user and redirect to login page.	Success

Account Marketplace Integrations	Show All Integrations	Will show list of integrations of status	Success
	Add Integrations and connect to marketplaces.	Will update integration and allow user to remove integrations.	Success
	Remove Integrations	This will refresh website and remove integration.	Success

Notes: All unit tests were successful.

Listing Based Testing:

Screen	Test	Expected Output	Result
View Listing Dashboard	User clicks on listing dashboard with account logged in.	Listing dashboard is presented with draft and active listings.	Success
	User clicks on listing dashboard with not account.	User is redirected to account login page.	Success
Add Listing	User fills in product with no integrations selected.	Listing is presented in draft listing section.	Success
	User fills in product with integrations selected.	Listing is presented in active listing section.	Success
	User tries to submit product with no product tile	Form will state missing product title and reset form.	Success
Delete Listing	User acts to delete a draft listing.	Listing is deleted from system and dashboard is refreshed.	Success
	User acts to delete an active listing.	Listing is deleted from system and all integrations.	Success
	User tries to delete an active listing with expired integration.	Listing will not be deleted, with alert presented that integration reconnection needed.	Success
Edit Listing	User inputs latest information	Listing is updated on system	Success

Information	in the editing listing page for active listings.	and all marketplaces associated with it.	
	User inputs latest information in the editing listing page for draft listings.	Listing is updated on system and refreshes page.	Success

Notes: All unit tests were successful.

Metric / Search Based Testing:

Screen	Test	Expected Output	Result
Search Query from Product Metrics	User search metrics based on query	Returns a range of metrics for that specific product.	Success
	User search metrics based on query is empty	Return validation error that query must not be empty.	Success
Filtering by Item Aspects	User filters based on specific metrics.	Returns list of items based on specific metrics.	Success
	User filters without specific metrics.	Returns list of items based on query.	Success
Product Search	Users' inputs query to search products	Returns a list of products based on query.	Success
	Users' input is empty to search products	Gives users a validation error of empty query.	Success
	User adds filtering for search products	Returns a list of products based on query & filtering.	Success

Notes: All unit tests were successful.

Appendix C: Risk Register

Description of Risk	Description of Impact	Likelihood Rating	Impact Rating	Preventative Actions
Developer account access is revoked by marketplace.	In the case that my application acts against their terms and usage policies, my application may be revoked access to user accounts through my app id.	Low/ Medium	High	Read each integrations policies of usage and terms so that the app is not randomly revoked access to a specific marketplace removing full functionality.

Not enough time is allocated to each phase of the project.	Due to insufficient time being allocated this may derail the project plan and may lead to the project not being completed before the deadline.	Medium	High	Create a Gantt Chart outlining each phase of the project and allowing for enough time for each phase, as well as implementing a small buffer to ensure each phase can be completed on time. Ensure weekly check-ups are made to the Gantt Chart to make sure the project is going to plan.
User actions exploiting listing capability.	Spam creation of listings may be a denial-of-service attack or be exploiting unknown use cases that may make the software/ company liable for these actions.	Low	Medium	Make assumptions of user actions, add limits to the number of listings that can be added within the system that reflects that off the integration/ marketplace. Also implement logs to log user activity to determine the cause of issues, exploits, and remove accounts that perform badly. Implement a terms of use policy to instruct users what they can and cannot do with system.
Student becomes unwell	If the student becomes unwell it may derail the project phases causing the student to get behind on the work.	Low	Medium	If the risk occurs the student must contact their supervisor to ensure they are made aware of the situation. Afterwards they must amend their Gantt Chart accordingly to ensure each phase can still be completed on time.
Unclear Objectives	The project solution may not address the problem described in the problem statement.	Low	Medium	Ensure sufficient time is spent understanding the problem and the necessary end user needs required to solve the problem outlined.
Scraper fails.	Scrapers defined to search for specific web elements by return error when the website layout changes or latest updates to id and classes are made meaning data cannot be obtained for these results.	High	Low	Try to use CSS selectors such as id and class instead of x-path selectors for more robust scrapers. Use of Crawlee (Crawlee, n.d.) framework to help deal with changes to website. Attempt to use web requests to private APIs

				to obtain data instead of using HTML or JS generated content.
Storage device running database has failed and now is corrupted.	Databases running on hard drives or other data storage may fail due to hardware failure outside of the developers' hands. In cases like this data storing listings and user integration tokens are lost and not possible to recover from without new data entry.	Low	High	Make backups of database to go back on in case of failure. Use of distributed databases such as Apache Cassandra (Apache, n.d.), to limit loss of data and ability to recover from it. Using key indicators for storage device performance to determine risk of failure.
Computer Crash	A computer crash can lead to the loss of software.	Low	Medium	Ensure that software is uploaded to GitHub regularly (every two days or when considerable progress is made).

References

Alaei, A.M., Taleizadeh, A.A. and Rabbani, M. (2020). Marketplace, reseller, or web-store channel: The impact of return policy and cross-channel spillover from marketplace to webstore. *Journal of Retailing and Consumer Services*, 65(102271), p.102271. doi:https://doi.org/10.1016/j.jretconser.2020.102271.

Apache (n.d.). *Apache Cassandra | Apache Cassandra Documentation*. [online] cassandra.apache.org. Available at: https://cassandra.apache.org/_/index.html.

Bajari, P., Hortaçsu, A. and Hortacsu, A. (2003). The Winner's Curse, Reserve Prices, and Endogenous Entry: Empirical Insights from eBay Auctions. *The RAND Journal of Economics*, 34(2), p.329. doi:https://doi.org/10.2307/1593721.

Bakos, J.Y. (1997). Reducing Buyer Search Costs: Implications for Electronic Marketplaces. *Management Science*, 43(12), pp.1676–1692. doi:https://doi.org/10.1287/mnsc.43.12.1676.

Boyle, M.J. (2021). *What Is Search Theory?* [online] Investopedia. Available at: https://www.investopedia.com/terms/s/search-theory.asp.

Calvano, E., Calzolari, G., Denicolò, V. and Pastorello, S. (2019). Algorithmic Pricing What Implications for Competition Policy? *Review of Industrial Organization*, 55(1), pp.155–171. doi:https://doi.org/10.1007/s11151-019-09689-3.

Chen, L., Mislove, A. and Wilson, C. (2016). An Empirical Analysis of Algorithmic Pricing on Amazon Marketplace. *Proceedings of the 25th International Conference on World Wide Web - WWW '16*. [online] doi:https://doi.org/10.1145/2872427.2883089.

Crawlee (n.d.). *Crawlee · Build reliable crawlers. Fast. | Crawlee*. [online] crawlee.dev. Available at: https://crawlee.dev/ [Accessed 28 Apr. 2023].

Dixon, S. (2023). *Average reach per post on Instagram 2022*. [online] Statista. Available at: https://www.statista.com/statistics/1353279/average-reach-per-post-instagram/.

GOV.UK (2018). *Data Protection Act*. [online] Gov.uk. Available at: https://www.gov.uk/data-protection.

Hwang, S.B. and Kim, S. (2006). Dynamic Pricing Algorithm for E-Commerce. *Advances in Systems, Computing Sciences and Software Engineering*, pp.149–155. doi:https://doi.org/10.1007/1-4020-5263-4_24.

Kuruzovich, J. and Etzion, H. (2018). Online Auctions and Multichannel Retailing. *Management Science*, 64(6), pp.2734–2753. doi:https://doi.org/10.1287/mnsc.2017.2732.

Lewis, R. (2022). *Retail sales, Great Britain - Office for National Statistics*. [online] www.ons.gov.uk. Available at: https://www.ons.gov.uk/businessindustryandtrade/retailindustry/bulletins/retailsales/december2021.

LUCKING-REILEY, D., BRYAN, D., PRASAD, N. and REEVES, D. (2007). PENNIES FROM EBAY: THE DETERMINANTS OF PRICE IN ONLINE AUCTIONS. *Journal of Industrial Economics*, 55(2), pp.223–233. doi:https://doi.org/10.1111/j.1467-6451.2007.00309.x.

OWASP (2020). *OWASP Web Security Testing Guide*. [online] owasp.org. Available at: https://owasp.org/www-project-web-security-testing-guide/.

P, Maes. (1996). Kasbah : An Agent Marketplace for Buying and Selling Goods. *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 1996*. [online] Available at: https://cir.nii.ac.jp/crid/1570291225342887040 [Accessed 28 Apr. 2023].

Pentina, I. and Hasty, R.W. (2009). Effects of Multichannel Coordination and E-Commerce Outsourcing on Online Retail Performance. *Journal of Marketing Channels*, 16(4), pp.359–374. doi:https://doi.org/10.1080/10466690903188021.

Smith, M.D. (2002). The Impact of Shopbots on Electronic Markets. *Journal of the Academy of Marketing Science*, 30(4), pp.446–454. doi:https://doi.org/10.1177/009207002236916.

Zabidi, A., Ali, M.S.A.M., Yassin, I.M., Tahir, N.M. and Rizman, Z.I. (2022). Analysis of Web Marketplace Integration for E-Suripreneur Multi-Channel Listing Software. *Mathematical Statistician and Engineering Applications*, [online] 71(3s2), pp.905–914. Available at: https://www.philstat.org/special_issue/index.php/MSEA/article/view/322 [Accessed 28 Apr. 2023].

Zhou, L., Zhang, P. and Zimmermann, H.-D. (2013). Social commerce research: An integrated view. *Electronic Commerce Research and Applications*, 12(2), pp.61–68. doi:https://doi.org/10.1016/j.elerap.2013.02.003.

END OF FINAL REPORT [20th April 2023]

X

Michael Peres