



FINAL REPORT

UNIVERSITY OF SUSSEX

DEPARTMENT OF ENGINEERING & INFORMATICS

SHOPPING BOT PROOF OF CONCEPT

Author:
David Ade-Odunlade

Supervisor:
Dr Natalia Beloff

Candidate Number:
234591

Degree:
BSc Computer Science with Artificial
Intelligence

List of Figures

1	Wget command to download a website	7
2	Cybersole Logo	9
3	Cybersole Graphical User Interface	9
4	Wrath Logo	10
5	Wrath Graphical User Interface	10
6	Prism Logo	11
7	Prism Graphical User Interface	11
8	Model View Controller Architecture Diagram	16
9	Use Case Diagram	17
10	Create Task - Sequence Diagram	17
11	Create Profile - Sequence Diagram	18
12	Gantt chart planning creation of this project	18
13	Initial Home screen design	19
14	Initial Tasks screen design	19
15	Initial Profile screen design	20
16	Initial Analytics screen design	21
17	Loading Screen on start up	24
18	Initialising providers	25
19	User Data class handles all the user data	26
20	Final Home Screen Layout	26
21	The map for creating the tab	27
22	The class outside of the widget tree that manages the state of the index across the interface	27
23	Task Instance class	27
24	Task Group Class	28
25	Task List class (manager)creating the tasks and serving out profiles	29
26	Final Tasks Screen Layout	29
27	Profile Instance Class	30
28	Profile Group Class	30
29	Profile Provider (Manager) Class	31
30	Final Profiles Screen Layout	31
31	Analytics class	32
32	Final Analytics Screen Layout	32
33	Sending user data to port 679 through dart	33
34	Javascript function listening for a connection	34
35	Calling the thread function	34
36	Starts the bot-web interaction script on a new thread	34
37	Implementation of the plugins	36
38	The Tor proxy server is added to the browser arguments	37
39	User Agent spoofing implementation	37
40	All types of delays	38
41	Code Snippet showing the retrieval of the products.json page	38
42	Code Snippet showing the construction of the search query	38
43	Function that calculates cosine similarity	40
44	Function that calculates semantic similarity	40
45	Function that gets N tweets of a topic	41
46	Pre processing and sentiment analysis	42
47	API request	43
48	Release data class that manages all the data flow	43

1 Statement Of Originality

This final year project report is submitted as a requirement for the BSc Computer Science w/ Artificial intelligence degree at the University of Sussex. Everything contained in this report contains work that is completed by me with exception of any background research performed which is cited appropriately. I give permission for this final year project report to be freely utilised providing it is referenced to this work.

2 Acknowledgments

I would like to acknowledge my project supervisor Natalia Beloff for her assistance throughout this whole project as well as the few individuals that provided feedback on my user interface prototype design.

Contents

1	Statement Of Originality	2
2	Acknowledgments	3
3	Introduction	6
3.1	Problem Area	6
3.1.1	Challenges in Creating Bots for Limited-Edition Product Acquisition	6
3.2	Motivations	6
3.3	Project Objectives	7
3.3.1	Bot-Web Interaction	7
3.3.2	Graphical User Interface	7
3.4	Ethics	7
4	Professional Considerations	8
4.1	Ethics	8
4.2	BCS Code Of Conduct	8
5	Background Research	9
5.1	Evaluation of Existing Shopping Bots - User Interface	9
5.2	Botting Methods	12
6	Requirement Specification	14
6.1	Fundamental Requirements	14
6.2	Non-Functional Requirements	15
6.3	Requirements conclusion	15
7	Software Design	16
7.1	Model View Controller Architectural Design Pattern	16
7.2	Interaction Design	16
7.3	Behavioural Design	17
7.4	Project Plan	18
8	Prototype Design	19
8.1	Home Screen	19
8.2	Tasks Screen	19
8.3	Profile Screen	20
8.4	Analytics Screen	21
8.5	Prototype Design Feedback	21
9	Implementation	22
9.0.1	Flutter	22
9.1	Changes	23
9.1.1	Sentiment Analysis	23
9.1.2	Tor Implementation	23
9.1.3	User Agent Spoofing & Browser Args	23
9.1.4	New Requirements	23
9.2	User Interface	24
9.2.1	Loading Screen	24
9.2.2	Providers	25
9.2.3	Home Screen	25
9.2.4	Tab Bar	26
9.2.5	Tasks Screen	27
9.2.6	Profiles Screen	29
9.2.7	Analytics Screen	32
9.3	Bot-Web Interaction	32
9.3.1	Web socket connection	32
9.3.2	Multi-threading	34
9.3.3	Bypassing security	35
9.3.4	Choosing the right product	38
9.4	Sentiment Analysis	41

9.4.1 DistilBERT & SST-2	42
9.5 Releases Data	43
10 Testing	44
10.1 Test Results	44
11 Conclusion	45
11.1 Evaluation of Process and Software Objectives	45
11.2 How Websites Can Avoid Bots	45
11.2.1 Remove the products.json from public access	45
11.2.2 Block any browser requests from Tor	45
11.2.3 Randomise element names or change them after a period of time	45
11.2.4 Defend against User agent Spoofing	45
11.3 Skills Demonstrated and Acquired	46
11.4 Project Improvements	46
11.4.1 Expanding the Scope of Supported Websites	46
11.4.2 Improving the User Interface and User Experience	46
11.4.3 Optimizing the Bot's Performance	46
12 Appendix A - Project Logs	47
13 Appendix B - Questionnaire Results	50
14 Appendix C - Test Plan Results	53
14.1 Bot-Web Interaction Functional Requirements	53
14.2 User Interface Functional Requirements	53
14.3 Sentiment Analysis Functional Requirements	53
14.4 Non-functional requirements	53
14.5 Results	53

3 Introduction

In this project, a proof of concept shopping bot is developed for two distinct websites to raise awareness about the techniques bots employ to access and purchase products. The project's objectives include creating algorithms that bypass anti-bot measures on both websites, developing an intuitive graphical user interface for users, and establishing a data analytics system to evaluate bot effectiveness and recommend purchasable products. The goal is to research and devise solutions for circumventing complex anti-bot systems, enabling companies to adapt their back-end defenses against bots.

This project aligns with my Computer Science with Artificial Intelligence degree, as it involves adhering to user-centric software development practices like Agile, and applying machine learning and artificial intelligence knowledge from my coursework. The project will test my programming skills and require learning new abilities, such as GUI creation for user-friendly bot interaction and Human-Computer Interaction for optimized interface design.

3.1 Problem Area

The internet has drastically altered the retail marketplace, little by little the percentage of sales made on the internet has increased to the point that as of September 2022, 25.3% [12] of all retail sales are made through online marketplaces. This provides many advantages for a consumer that are not located near a specific store or do not have the necessary time to be able to go into a physical store as the internet is a flexible medium through which businesses can sell their products. Comparably, the internet also provides many benefits to businesses that wish to expand their customer reach or reduce the extra costs of in-store maintenance and employees. Despite this, just like all things in the cosmos, there is a downside. One significant drawback that restricts many businesses from launching an online counterpart to their physical store is the increasing presence of bots used to purchase products online. This all came to a breaking point when in 2019 the 'Bodega New Balance 997S' sold out in under 10 minutes, however, 60% of those sneakers were sold to users that used bots. Since then bots have been on the rise and have become increasingly popular around a community of business-minded individuals that purchase these products, with bots, intending to sell them on for a profit, the aforementioned 'reseller' community has forced major companies like Nike, Adidas, AMD, and NVIDIA to program anti-bot protection and alternate methods of commerce on their websites to in an attempt to keep the bots at bay. Consequently, a major problem emerges for users without a bot. Initially created to help their operators purchase large quantities of limited-edition products, bots are now that they have become so prevalent, regular users struggle to buy any rare items.[1]

3.1.1 Challenges in Creating Bots for Limited-Edition Product Acquisition

Bots, or automated software programs, are designed to enhance an individual's chances of quickly purchasing limited-edition products. These products can range from rare sneakers and Non-Fungible Tokens (NFTs) to high-demand graphic cards and collectible playing cards. The type of bot employed is contingent on the e-commerce platform used by the website selling these exclusive items.

Creating bots for this purpose is challenging due to the difficulty in using a website's HTML or JSON code to accurately determine whether the bot is being directed to the correct product. This issue arises because the bot possesses limited knowledge of the product's exact location on the website. To circumvent this obstacle, many bot developers implement a text similarity score system, which allows them to identify the product with the highest similarity to the keywords provided by bot users.

However, even with this approach, the manner in which the bot accesses the website depends on the sales method employed by the online retailer [13]. This means that developers must constantly adapt their bots to the ever-changing landscape of e-commerce platforms and sales techniques, making bot creation a complex and demanding task.

3.2 Motivations

Being an avid sneaker fan since childhood, I have attempted many times to purchase sneakers on the internet but since the uprising of bots, getting a successful purchase is rare. Since understanding the world of sneaker botting and high-end products, I have been fascinated with the complexity of the software used to bypass the security measures of websites as they have managed to bypass security measures for so long. As a Computer Scientist myself, I have looked into the methods that these software engineers use to create these bots and I believed that I could produce a better bot. Additionally, I believe that this undertaking can be useful for many online retail marketplaces as, in this report, I relay information about botting and how these methods get past anti-bot security measures. On top of this, as an AI student, I wanted to take a shot at implementing NLE

and Machine Learning classification into this project as an expression of what I have learnt throughout my Computer Science with Artificial Intelligence degree.

3.3 Project Objectives

3.3.1 Bot-Web Interaction

Building a system that interacts with a website that does not alert any bot protection systems is a complex matter. As mentioned before, the system has to be able to navigate to the product using only keywords given from the user. It is possible to search through the products.json file of a website a perform a linear search to find the website handle that contains the desired product but some website handles are not correlated to the product. Puppeteer is the main too that I will use to simulate user clicks. Web-scraping and browser interaction are two processes that are needed enable a bot to simulate website interaction as if they were a human. Web-scraping is the process of using bots to extract information and data from the website using the underlying HTML code or HTTP requests to direct the bot towards the desired information. Subsequently, this information is used by the way browser interaction processes to know where the bot will click on the web-page. There are various ways to perform data extraction and browser interaction on website however, in this project I will use two open-source web frameworks that permits you to execute cross-browser tests named *Puppeteer* Puppeteer will be used in this project due its ability to do cross-browser testing and to simulate user clicks.[3]

3.3.2 Graphical User Interface

Graphical User Interfaces are useful to present the user with points of interest that abstract away from the command line interface that underlie that. The intention of having a graphical user interface is to make it easier for users to interact with the software without any prior knowledge. We will make a graphical user interface using using the flutter and dart. Flutter is an open source UI software development kit that can be used to develop cross platform applications from a single code-base and is written in Dart, a programming language designed for client development. We chose flutter because the of the vast amount of libraries it has that make it easier to code GUIs and due to the benefit of fast GUI testing with hot reloading.

3.4 Ethics

To demonstrate that this project does not intend to be malicious I made steps to ensure that this project was made ethically and I limited the bot used in this project's functionality. As I created the bot to only demonstrate the methods used to bypass anti-bot measures, the bot is not actually able to purchase products but navigates towards the checkout checkout page of a website which is past the anti-bot measures. Moreover, despite the fact that the websites that I have chosen to bot do not have any bot restrictions in the terms of use, I have downloaded the websites and contained it in the folder so that I can switch over to the downloaded file when I want to test it out with many requests. I downloaded the files with this line: [21][20]

```
wget --recursive --no-clobber --page-requisites --html-extension --convert-links --restrict-file-names=windows --domains uk.trapstarlondon.com --no-parent https://uk.trapstarlondon.com
```

Figure 1: Wget command to download a website

By addressing these security and ethical considerations, the project will not only increase awareness about the methods used by shopping bots but also contribute to the development of more effective anti-bot measures that protect both businesses and consumers in the online marketplace.

4 Professional Considerations

4.1 Ethics

In this project, due to the lack of participants needed for software testing, there was no need to gain any ethical clearance however, there are some issues commonly associated with shopping bots that we will avoid in this project for the sake of ethics.

1. ***We will ensure that any testing performed does not occur on the certified and live website.***

Due to the fact that using bots on live websites break the terms & conditions of some websites we will test the bot-web interaction system on dummy websites that contain the same HTML code and anti-bot measures but will not damage the reputation of the company. As a proof of concept project we are solely trying to demonstrate how the anti-bot protection measures are not enough to protect purchases from bots. We will generate this dummy websites using the chrome extension Clone Page.

2. ***We will ensure that no one gets a hold of the software and attempts to use this software on any certified websites.***

We will make sure that the software does not get released and result in any targeted websites reputation's being damaged.

4.2 BCS Code Of Conduct

The BCS Code Of Conduct sets the professional standard required by the BCS as a condition of membership. The BCS Code Of Conduct is important as it sets professional standards of competence, conduct and ethical practice for computing in the United Kingdom.

Professional Competence and Integrity

1. ***Develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field.***

I will ensure that I continuously develop my professional knowledge and skill as well as staying aware of technological developments in relevant industries.

2. ***Ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.***

I will ensure that I have an understanding of any Legislation that is relevant to bots and make sure I comply with said Legislation.

3. ***avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.***

I will ensure that this project does not damage the property, reputation or employment of any company that the shopping bot works on and I will ensure that this project will comply with all the terms and conditions of each website that the shopping bot gets tested on.

Duty to Relevant Authority

4. ***NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information), or take advantage of the lack of relevant knowledge or inexperience of others.***

I will make sure to be accurate and not withhold any information on the performance of software systems and services or take advantage of any individuals.

Duty to the Profession

5. ***accept your personal duty to uphold the reputation of the profession and not take any action which could bring the profession into disrepute.***

I accept my personal duty as an prospective BCS member to uphold the reputation of the profession and not perform any actions which could bring the profession into disrepute.

5 Background Research

5.1 Evaluation of Existing Shopping Bots - User Interface

To be able to develop a novel shopping bot I will need to evaluate existing shopping bot applications to get an understanding of the human computer interaction design principles of each application. In determining the key strengths and weakness we can improve on the weakness and ensure the strengths are present in our projects. We we also have to try and determine the types of bot-web interaction methods that used by other bots.

Cybersole



Figure 2: Cybersole Logo

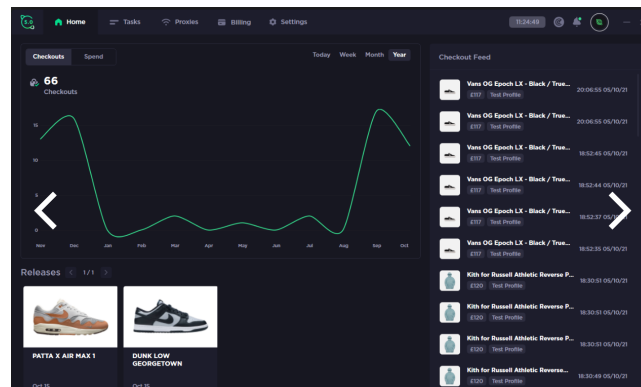


Figure 3: Cybersole Graphical User Interface

HCI Design Principle

Visibility

- Dark theme appeals to target audience.
- Intuitive icons.
- Consistent font size and colour scheme.
- UI is aesthetically pleasing due to contrasting and futuristic colors.

Consistency

- Highly consistent color scheme, font size and style.
- Icons are consistent and correlate with their function.
- Uniform design and structure.

Affordance

- Tabs and buttons are easy to understand.
- Use of color changes clearly show what the user has and has not interacted with.
- Every button and tab has a clear action.

Wrath



Figure 4: Wrath Logo

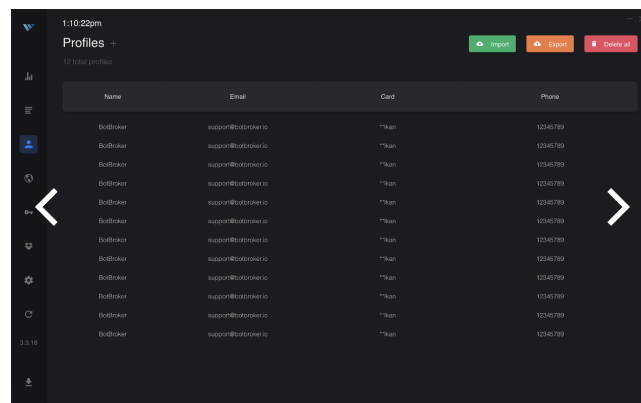


Figure 5: Wrath Graphical User Interface

HCI Design Principle

Visibility

- Content is separated by tabs and has clear color changes to show what the user has and hasn't interacted with.
- Dark and bland theme may come across as boring for some users.
- Consistent color scheme.
- Some text is hard to see due to the similarity in color between the background and the text.

Consistency

- Highly consistent color scheme, font and layout.
- Consistent icons for buttons with the same function.

Affordance

- Tabs has distinct coloring when interacted with.
- Arrows and icons are easily identifiable.
- Lots of space make the user interface easy to look at
- Every button and tab is easy to understand and have a specific function

Prism

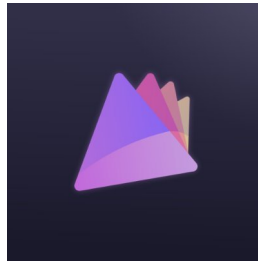


Figure 6: Prism Logo



Figure 7: Prism Graphical User Interface

HCI Design Principle

Visibility

- Tabs are a distinct colour from the side bar background and when clicked
- The Stats buttons are bright and centered to make sure the user sees it.
- Consistent color scheme.
- All text is readable as the text is white and has an underlayer of another lighter blue.

Consistency

- Highly consistent color scheme, font and layout.
- Font is the same for all buttons and Icons.

Affordance

- Areas that can be clicked on the GUI is clear as it is shown to be a different colour
- User interface is quite open to give a spacious feel when using the software
- Every widget has a specific function and is clearly separable from other widgets.

Evaluation Conclusion

The HCI design principles are important as they inform how a user interface should be created. In conclusion, it is clear that the colour scheme of the software should reflect the target audience in a way that does not lose the consistency between font sizes, font styles, images and icons. An ideal user interface also interacts with the user in some way to show that the user has interacted with that particular widget for example, the way Cybersole's user interface has icons change to the color green when pressed which makes the user interface fun to interact with. [24]

5.2 Botting Methods

To effectively bypass websites' security measures, bots employ a multitude of sophisticated techniques aimed at enhancing their chances of being perceived as genuine users, rather than as automated programs. This section will explore the various methods that bots implement to achieve their goal of evading detection and discuss the implications of such practices in the broader context of online security and user experience. Because of the word limit, this section is not exhaustive and other methods implemented by this project will be discussed in the Implementation section

Headless Browsers

Bots may use headless browsers like Puppeteer or Selenium, which can interact with websites like a real user, executing JavaScript and rendering pages, making them harder to detect. Headless browsers are web browsers with a User interface and are designed to run in the background, typically on a server or as part of automated scripts to interact with web pages problematically. There are two popular frameworks commonly used to run headless browsers: Headless Browsers offer several advantages:

- **Speed** - Without the need to render web pages on a screen, headless browsers can be significantly faster than traditional browsers, making them ideal for automation and testing.
- **Resource Efficiency** - Running a browser without a GUI typically requires fewer system resources, allowing developers to run multiple instances simultaneously or execute tasks on low-resource devices or servers.
- **Consistency** - Headless browsers provide a consistent environment for running tests and automation scripts, reducing the risk of unexpected behavior due to browser or operating system differences.
- **Flexibility** - They can be integrated with various tools and platforms, allowing developers to create custom solutions for their specific needs.

[4] As a result Headless Browsers can be used to bypass certain security measures and allows users to run multiple instances of a bot at the same time. By controlling the elements to replicate those made by a user the bot can be used to purchase products on a website by navigating to checkout.

IP Rotation

Bots often use proxy servers or VPNs to rotate their IP addresses, making it difficult to identify and block them based on IP alone. IP Rotation is a technique used by bots to avoid detection and blocking by changing their IP addresses frequently. This is done to evade IP-based security measures. IP rotation works by distributing grequests over a range of IP addresses, making it harder for a website or server to identify and block the source of the unwanted traffic. IP rotation can be implemented through:

- Proxy Server
- VPNs (Virtual Private Networks)
- IP Rotation Services
- Residential IP Networks
- Tor Network

[5]

User Agent Spoofing

User Agent Spoofing is a technique in which a cline , typically a bot or a web scraper, alters its user-agent string to masquerade as a different browser or device. The user-agent string is a text identifier that web browsers and HTTP clients send to web servers as part of the request headers. By spoofing the user-agent string, bots can evade detection bypassing anti-bot measures that rely on user-agent analysis. A user-agent string typically follows this format: "iProduct¿ / iProductVersion¿ (iComment¿)"

For example, a Google Chrome browser on a Windows 10 system might have a user-agent string like this:

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"

This string contains information about the operating system (Windows NT 10.0), the browser (Chrome), and the rendering engine (AppleWebKit/537.36).

To perform user-agent spoofing, Bots can use a list of predefined user-agent strings or generate random ones, changing the user-agent string for each request. It is also popular practice to imitate browsers are widely used like Chrome, Firefox or Safari to blend in with the genuine traffic flow. Furthermore, Bots can alter their user-agent string to act as if they are mobile devices, tablets or even gaming consoles to access platform specific content. [23]

6 Requirement Specification

6.1 Fundamental Requirements

Bot-Web Interaction

- The bot will be able to bypass the anti-bot security of 5 websites
- The user will be able to purchase for products through the bot without alerting any bot protection system
- The user will be able to customise tasks that affect how the bot interacts with the website
- The user will be able to apply proxies that the bot will use to bypass the websites security
- The user will be able to start, pause, edit and delete tasks
- The user will be able to store their card details on to the bot with a password for security
- The bot will be able solve any Captcha
- The user will be able to save tasks and run them as groups
- The user will be able to input any details that are needed when making an order on a website e.g. delivery address, postcode, full name, e-mail address, phone number and chosen shipping method
- When creating a task, the user will be able to choose:
 - The website the product is on
 - Type of bot method the user wants the bot to use (HTTP Client or Selenium)
 - The number of bots active on this task
 - Any keywords that will aid the bot find the product of choice
 - The type of product the bot is after
 - The size of the product(e.g. XS,S,M,L,XL,38,42 etc)
- Web-scraping and sentiment analysis will be used to take data from the internet with the intention of passing this data to the data analytics system

Graphical User Interface

- The GUI will provide an interface for a user to create tasks
- The GUI will display status information about each task
- The GUI will display information about recent purchases the bot has made
- The GUI will display information relating to any upcoming exclusive drops.
- The GUI will prompt the user for any information needed to run a task
- The GUI will display the information returned from the data analytics systems

Data analytics

- This system will be able to classify product soon to be released into one of two classifications: wanted by the user or not wanted by the user.
- The classification made will be constantly reevaluated to always try improve the classifications.
- This system will allow the user to judge the classifications on whether they are accurate or not.
- This system will store all the purchase information made by the user.
- Analyse bot behaviour

6.2 Non-Functional Requirements

Software Usability

- The desktop app should have an magnetic user interface design
- The desktop app should be easy to use and straight-forward
- There should be a manual for how to use the software (video or text)
- All prompts should be easy for the user to understand

Software Efficiency

- The desktop app should be as responsive as possible to all user actions
- The task status updates should update instantly and be displayed to the user through the GUI

Software Dependability

- Everything on the software should work and be dependable
- Any user data stored on the system should be protected and not lost
- If anything fails on the desktop app, the software should be able to reboot

Software Interoperability

- The desktop app should be available on Linux and Windows
- The bot-web interaction should be able to run on the latest versions of Google Chrome and Microsoft Edge

6.3 Requirements conclusion

All the bots on the market today reach of all the fundamental requirements for bot-web interaction but some shopping bots use command line user-interfaces which is often too little abstraction for the regular user. This is where many bots under perform as an ideal system will include a graphical user interface that does not require any prior knowledge. Despite this, Many of these bots use both HTTP Client and Selenium to provide a range of options for the users. A system that uses both HTTP Client and Selenium is ideal so as a result we aim to incorporate this into our project to replicate how advanced bots enter a website. In this project, we are circumventing anti-bot protection measures to show faults in said systems and due to this we are incorporating advanced shopping bot practices to exemplify how bots can and do enter websites

7 Software Design

7.1 Model View Controller Architectural Design Pattern

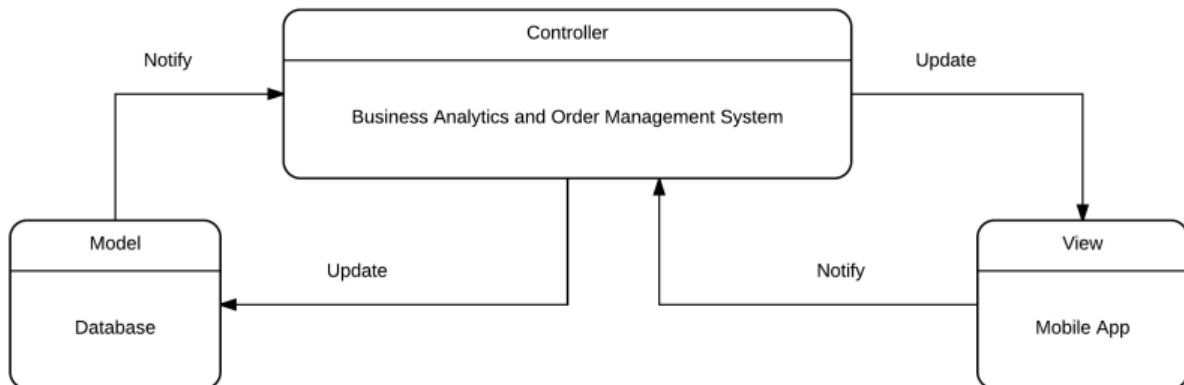


Figure 8: Model View Controller Architecture Diagram

We will use the model-view-controller architectural design pattern to separate the software application into three components, each of which with their own specific task. The model, the central component of the pattern, is responsible for an application's dynamic data structure. It manages the rules of the application. The view is the visual representation of the data and is what the user sees. Finally, the controller is the component that manages the exchange of information between the model and the view.

Model

The model deals with the database the holds the user's personal data, purchase history, upcoming product drops, the bot-web interaction and the data analytics AI. The bot-web interaction tasks created by the user will be automatically update the model using the controller.

View

The view deals with the visual representation of the model data and alerts the controller of any new upcoming products, news, checkout status and any updates that are coming to the software and subsequently, any new updated data will then be shown on the view.

Controller

The controller deals with taking the user's personal data and purchase history and storing it inside the model. The controller will also extract product purchase details in order to provide the business analytics to the AI in order for it to form recommendations of which products the users likes.

7.2 Interaction Design

This Use Case Diagram is a visual representation of all the possible choices a user has when interacting with a the desktop application and subsequently, how the application interacts with the bot-web interaction system and the data analytics system

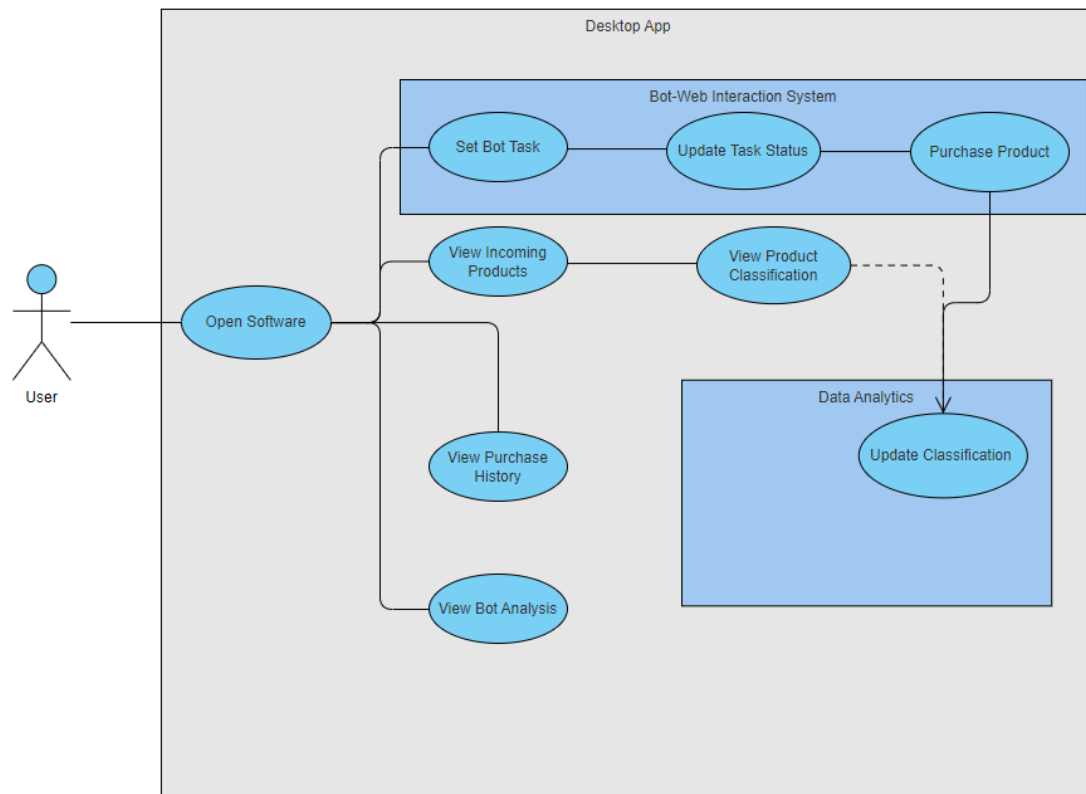


Figure 9: Use Case Diagram

7.3 Behavioural Design

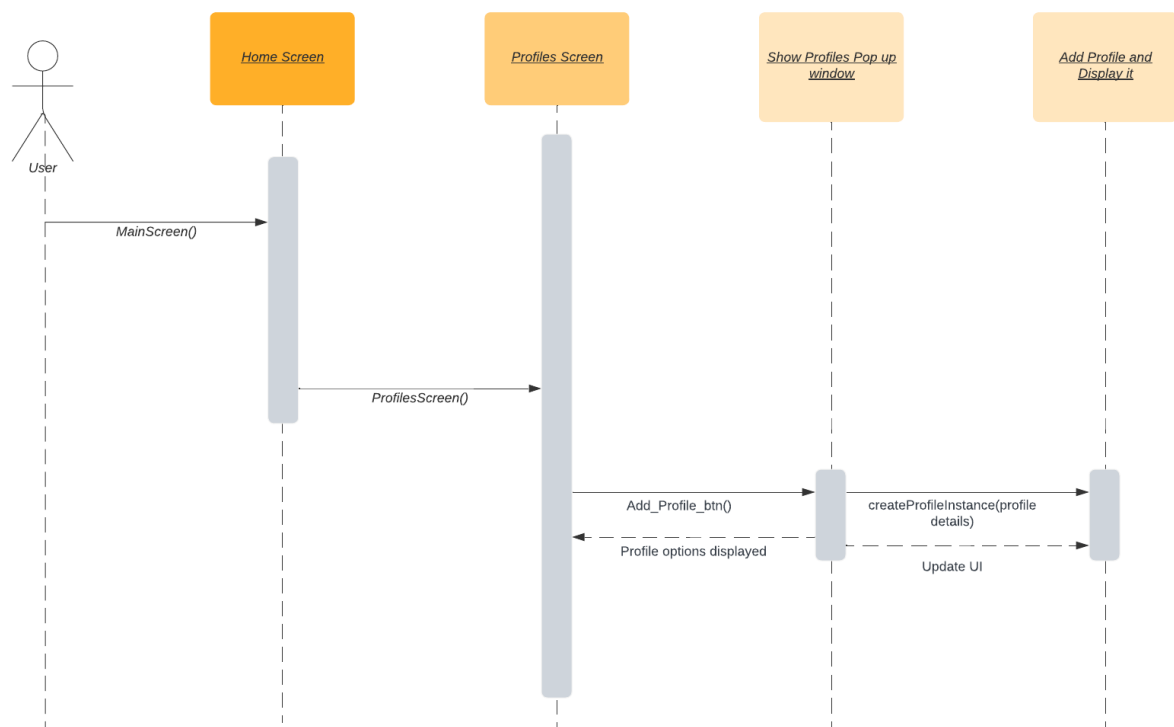


Figure 10: Create Task - Sequence Diagram

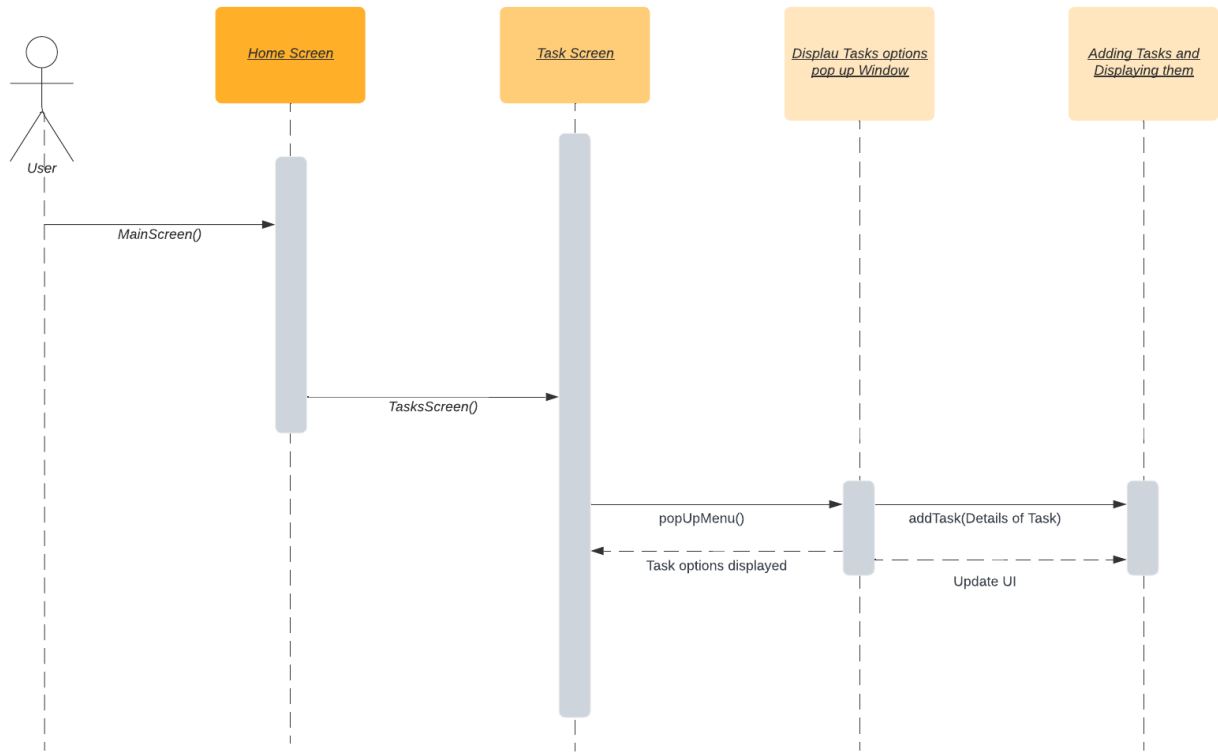


Figure 11: Create Profile - Sequence Diagram

7.4 Project Plan

This is a Gantt chart representation of how I expect to spend my time creating this software:

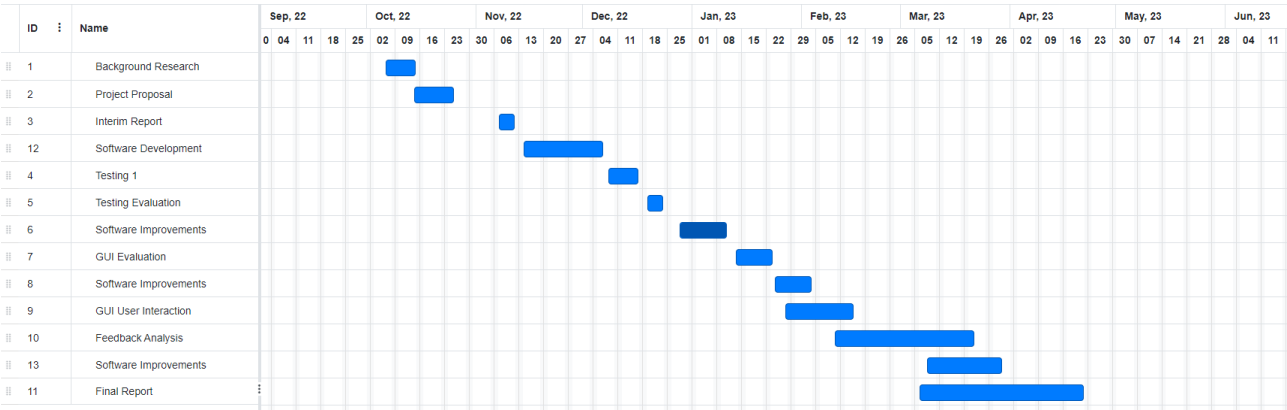


Figure 12: Gantt chart planning creation of this project

8 Prototype Design

Once I finished designing the software architecture of the system, I needed to create a graphical user interface design prototype to layout my ideas and begin to create a user interface that can align with the analysis performed on other shopping bots. I created this prototype using Figma, a web application for interface design which makes it easy for developers to use vector graphic editor and prototyping tools to design Mobile and desktop apps.

8.1 Home Screen

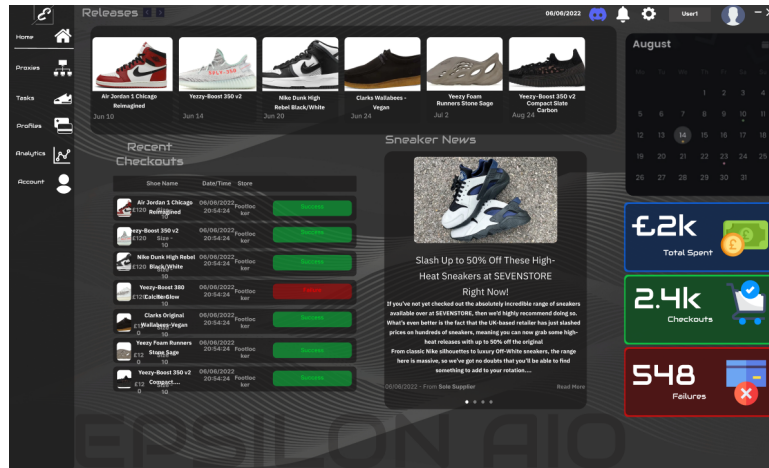


Figure 13: Initial Home screen design

The home screen is an introduction screen that gives the user various pieces of information that is relevant to the user that influence how the user might enter some details on the task screen. Due to the fact that I aim to make software that replicates how current sneaker bot software are used, I thought it was necessary to create a GUI that showed this. As a result, In the home screen I have a created different sections to display different pieces of information:

- sneaker releases that are coming out on the day of use
- the recent checkouts of the user and details about the purchase including time, price and website
- news about the current sneaker market or upcoming sneakers.
- a calendar
- stats on the amount spent by the user, number of checkouts and number of failures.

Modern sneaker software have all this information in order for a user to be able to tweak their software tasks in the right direction to get in fashion products from the right websites.

8.2 Tasks Screen

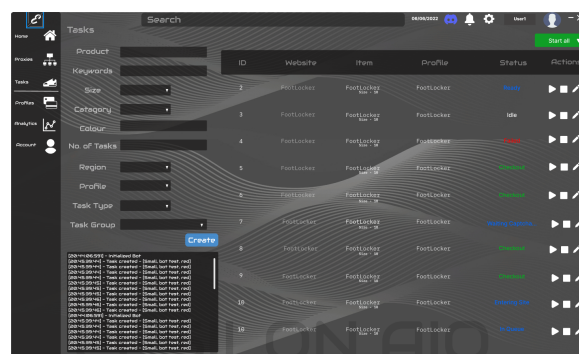


Figure 14: Initial Tasks screen design

On the task screen the user can create tasks/bots that will enter a website and purchase products for the users when all the necessary information is entered. Using Figma, once again, I created a layout for the task screen which allowed the user to put in information that is needed for the software to scrape websites:

- the product the user wants the software to target.
- keywords that will help the software find the product on the website as the input might not be exactly the same as the product on the website.
- the Size of the product the user wants to purchase.
- Which clothing category the product comes under.
- The number of bots the user wants to send to the site.
- The region the website is in.
- The group of bank cards the user wants to use.
- Whether the user wants to use a proxy server or not.
- The group of tasks the user wants to use.

By task I mean the operation that a bot is performing so multiple tasks are contained in a task group which can then all be launched at once.

8.3 Profile Screen

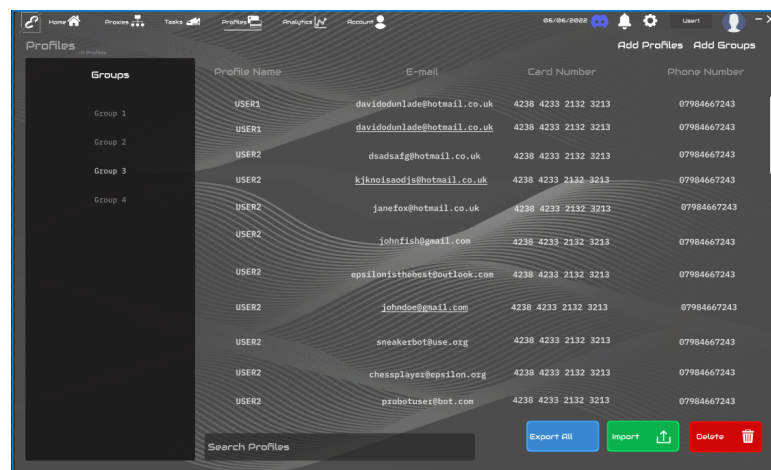


Figure 15: Initial Profile screen design

On the profile screen the user can create profiles and enter their credit card details, email, phone number and address onto said profile which will be used anytime the user specifies that they want to use this profile to purchase a product. The user can also create groups of these profiles which is used when a user wants to send out multiple bots to a site. In order to avoid the anti-bot protection it is crucial that a user would not use the same card repeatedly as the website might realise that multiple IP addresses are using the same card - this would get that bank card banned from the website.

8.4 Analytics Screen

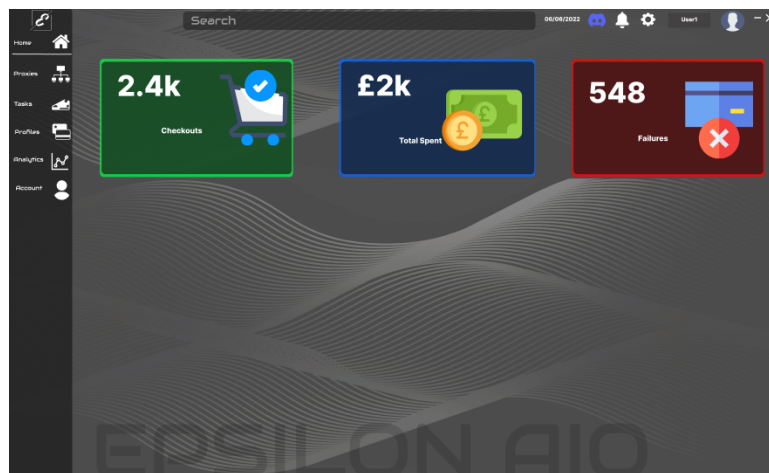


Figure 16: Initial Analytics screen design

On the analytics screen, I found it hard to map out a concrete look for the Data analytics artificial intelligence but I wanted to put the stats, that are also present on the home screen, on this screen and then I was going to put the data analytics screen below this.

8.5 Prototype Design Feedback

The next phase in this project involved obtaining feedback from potential users through a survey. This allows users to examine the user interface, providing insight into their preferences, areas for improvement, and their overall enjoyment of all interface screens. A total of 6 users, aged 18-26, responded to the app survey, while 1 individual, aged 32-59, responded to the user interface survey. As a result, the sample size and diversity were not notably extensive, which will be considered during feedback analysis. A comprehensive evaluation of the survey outcomes is available in Appendix B.

Changes to Design

As a result of the feedback, I decided to change the colour scheme of the application and simplify the user interface to reflect the way that bot software's would make their user interface the accessible to customers without any technical skill. This resulted in a dark blue/light blue colour scheme, a simplification of the Home screen and buttons that stand out from the background. Furthermore, I decided to change the font of most of the application as one of the participants stated that the font would be an improvement to the application.

9 Implementation

I believe it is important to discuss the technologies needed to create the desktop app and achieve the required specifications stated in the aforementioned section. After I finish describing the key technologies used I will discuss how these technologies were implemented in detail. I made sure to use the feedback to inform my decisions on the method of implementation.

Model

This represents the data and the business logic of the application. For this project the model is written mainly in dart and some parts are in JavaScript. I chose to use dart as part of the model because it is the easiest way to manage states and store back end data in a flutter application. Additionally, I used javascript as part of this model which manages the bot-web interaction side.

In the model for my application I use providers in dart which is a popular state management package. Providers simplify the process of managing and distributing state across different parts of the application. Providers make it easy to access and share data across multiple widgets in the widget tree. In my application I use providers for much of the backend code to handle the backend logic for storing and sharing data like profiles and tasks across the app.

The javascript used in this project manages the bot side which is hosted on the users localhost on port 679. Once the GUI connects to the port by starting a bot/task, a dictionary of information regarding the task details are sent through to javascript side which then uses these details to start the task. I decided to use Web sockets because this allows there to be an open communication from the GUI to the bot side allowing the controller to update the GUI about the current state of the task. This section of the model also contains the instructions for the bot which is ran on its own thread allowing multiple bots to perform at the same time.

View

The View represents the user interface and displays the data from the Model. Flutter is a cross-platform software development kit (SDK) app development that was created by Google. Flutter utilizes a reactive programming model which offers a rich set of pre-built widgets which are available in the form of libraries on <https://pub.dev>. flexible design options and extensive libraries making it ideal for developing high-quality and dynamic mobile applications. Features like hot-reload, which allows users to reload the view after a change is made allowing for rapid debugging, are a major reason why there is such a large and active community for flutter resulting in numerous third-party libraries of easily install-able widgets. The view will show the available products, user profiles, and tasks. It will also provide user input elements, such as buttons, text fields, and check boxes, for interacting with the application. To understand some of the coding explanations I think it is worth giving a description of flutter and how it works

9.0.1 Flutter

At its core of Flutter is the concept of widgets, which are the basic building blocks of the UI. Widgets can be as simple as a text element or as complex as a layout container. They are organized in a hierarchy known as the widget tree. The widget tree represents the structure of your app, with the root widget at the top and the leaf widgets at the bottom. In Flutter, everything is a widget, including layout elements like rows, columns, and grids, as well as interactive elements like buttons, text fields, and sliders. Widgets can be combined to create custom UI elements and are organized using composition rather than inheritance. State management is a crucial aspect of building applications in Flutter. State refers to the data or information that is used to build the UI and can change over time. Managing state effectively is essential for creating responsive and maintainable applications.[8]

In Flutter, widgets can be classified into two main categories based on how they manage state:

- **Stateless Widgets:** These widgets do not store any mutable state. They only depend on their configuration, which is passed to them through their constructor. Examples of stateless widgets are Text, Icon, and RaisedButton widgets.
- **Stateful Widgets:** These widgets can store mutable state and are used when the UI can change dynamically due to user interaction or external events. Examples of stateful widgets include checkboxes, radio buttons, and text fields widgets.

To manage state in a more complex application, Providers, BLoC, or Redux can be used. These solutions help you organize and manage state in a more scalable way that allows for mass data flow.

Controller

The Controller handles the user input from the View and updates the Model and View accordingly. the Controller is written in javascript and dart in order to update the user interface accordingly. The javascript side takes data from the user interface in the form of inputs and uses this data to run the bot-web interaction part of the backend and subsequently updates the model and view accordingly. The dart controller code does the same role in the project but with the user interface only backed data e.g. moving from screen to screen, clicking buttons etc.

9.1 Changes

Throughout the development of my project I had to pivot and change some features to be able to align the project with the requirements I stated earlier.

9.1.1 Sentiment Analysis

Initially, my objective was to develop a data analytics system that would integrate artificial intelligence to categorize emerging products and, subsequently, recommend them to users. However, to ensure the efficacy of this system, it would be necessary for the software to function across a larger number of websites, thereby providing a more comprehensive data set for the analytics system. As it stands, the current software's limitation to only two websites restricts the variety of available products, which, in turn, hampers the data analytics system's ability to generate an accurate classification. As a result, I decided to incorporate a sentiment analysis algorithm that takes a word, keywords and N and returns the number of positive and negative sentiments about that word by analysing N sentiments. This algorithm also returns the top 15 most used words in those sentiments. This feature would be useful for individuals that use shopping bots because it would be important to know how the general public feels about a specific topic or product. I will go into further detail about how this system works in the bot-web interaction section.

9.1.2 Tor Implementation

Proxies are used by shopping bots to achieve a strategic advantage in the competitive landscape of online purchasing. These intermediaries serve as a means to obfuscate the bots' activities and evade detection, allowing them to circumvent restrictions imposed by websites. By utilizing multiple proxy servers, shopping bots can distribute requests across a diverse range of IP addresses, thus reducing the likelihood of being identified and blocked. As part of my initial design of the software, I wanted to allow users to input custom proxies to be used on the bot however, I could not test this as this would require the purchase of proxies which I was not willing to do. As an alternative pathway, I decided to give the script access to the Tor network. Tor is a decentralized, open-source system designed to facilitate anonymous communication and enhance privacy on the internet by routing encrypted data through a series of volunteer-operated servers. This additional feature allows the users to gain access to a range of IP addresses across the globe without having to pay for them. Additionally, due to the way Tor operates, the IP-address used cannot be linked back to the user.[22]

9.1.3 User Agent Spoofing & Browser Args

Due to the way I was testing my software and the websites that I decided to implement a solution for, It was not necessary for me to create a method to implement a Caphca solver for. Furthermore, websites that do not suspect any bot activity tend to not require users to solve any Captcha's as this would devalue user experience. To ensure that this software was not cheapened by the lack of this feature, I made sure to implement strong anti-bot bypass measures like user agent Spoofing and additional browser arguments. User agent spoofing is a technique in which a script masquerades as a different device by altering the user agent string to avoid linking the bot back to the user. On the other hand, browser args are parameters for a web browser than can be adjusted to alter its behaviour and functionality. By altering browser arguments one can circumvent automatic measures some websites use to detect automated scripts.[23]

9.1.4 New Requirements

As a result of the changes made in this project, I thought it necessary to list the functional requirements for each.

Sentiment Analysis

- Create an Artificial Intelligence that takes a Sentiment and returns the number of positive and negative sentiments that show through twitter.
- Allow the user to chose the number of sentiments analysed.
- Train the Model using a comprehensive Sentiment analysis model.
- Create a method through which the user can input this data in the graphical user interface.
- return the top 15 keywords of that occur in the sentiments analysed.
- Implement the best practice is NLE to date.

Tor Implementation

- Implement a method for the bot script to work on a tor browser
- Create a way for the user to decided whether they want a task to use Tor or not.

User Agent Spoofing & Browser Args

- Implement the most recent and common User Agents.
- Implement all the necessary browser arguments.
- Implement this on both websites.

9.2 User Interface

After incorporating the feedback for the prototype design redesigned the user interface. I changed the colour scheme and added a new logo. I also made the design more simple to align it with other shopping bot software user interfaces. For the sake of space, I will not go into the GUI Code aspect but it is described well in the view of the MVC priciple.

9.2.1 Loading Screen

I created a loading screen in order for the flutter project to initialise all the data and providers. The loading screen is created on start up using the main.dart file

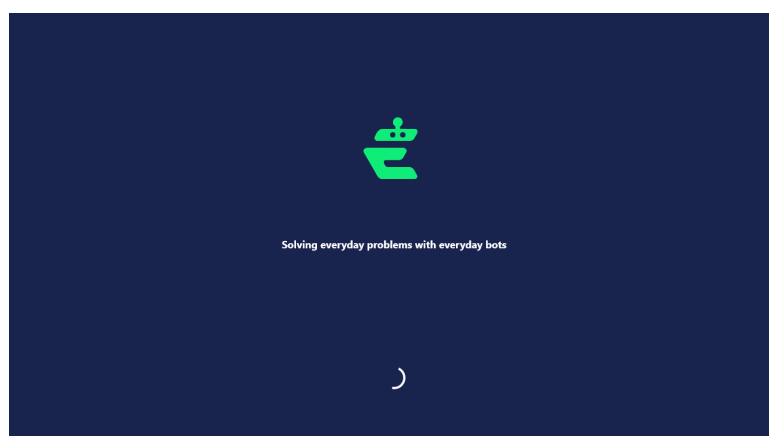


Figure 17: Loading Screen on start up

```
runApp(  
  MultiProvider(  
    providers: [  
      ChangeNotifierProvider(create: (_) => TasksLists()),  
      ChangeNotifierProvider(create: (_) => ConsoleLogger()),  
      ChangeNotifierProvider(  
        create: (_) => Taskinstance(  
          0,  
          "",  
          "",  
          Profile(  
            "",  
            "",  
            "",  
            "",  
            "",  
            "",  
            "",  
            "",  
            "",  
            "",  
            Profile_card(  
              profile_name: '',  
              address: '',  
              card_name: '',  
              card_no: ''), // Profile_card  
            ), // Profile  
            "",  
          ],  
          false)), // Taskinstance // ChangeNotifierProvider
```

Figure 18: Initialising providers

9.2.2 Providers

Initializing providers in Flutter is crucial for managing state and sharing data across the widget tree. It ensures proper data source setup, connects dependent widgets, and manages state changes using classes like `ChangeNotifier` or `ValueNotifier`. Providers also enable lazy resource initialization and scoping of data to specific parts of the widget tree, optimizing app performance and providing flexibility in data access and visibility.

9.2.3 Home Screen

User Data Statistics

The statistics on the side monitors the amount spent, number of checkouts and number of failures of the user. To apply this functionality I used the 'Shared Preferences' plugin in Flutter which allows users to store simple key-value pairs of data persistently on the native platform. Shared preferences in Flutter offers a convenient method for storing and retrieving modest amounts of data, including user preferences, application settings, or the state of the app. This functionality remains effective even when the app is closed or the device undergoes a restart.[18]

```

class UserData with ChangeNotifier {
  static final UserData _instance = UserData._internal();
  UserData._internal();

  static UserData get instance => _instance;

  int totalSpent = 0;
  int checkouts = 0;
  int failures = 0;
  double totalSpentDouble = 0.0;
  List<ProfileGroup> profileGroups = [];
  List<TaskGroup> taskGroups = [];

  int get totalSpent_ => totalSpent;
  double get spentDouble => totalSpentDouble;
  int get checkouts_ => checkouts;
  int get failures_ => failures;

  // Load data from SharedPreferences
  Future<void> loadData() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    totalSpentDouble = prefs.getDouble('totalSpentDouble') ?? 0;
    checkouts = prefs.getInt('checkouts') ?? 0;
    failures = prefs.getInt('failures') ?? 0;
  }

  // Save data to SharedPreferences
  Future<void> _saveData() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.setDouble('totalSpentDouble', totalSpentDouble);
    await prefs.setInt('checkouts', checkouts);
    await prefs.setInt('failures', failures);
  }
}

```

Figure 19: User Data class handles all the user data

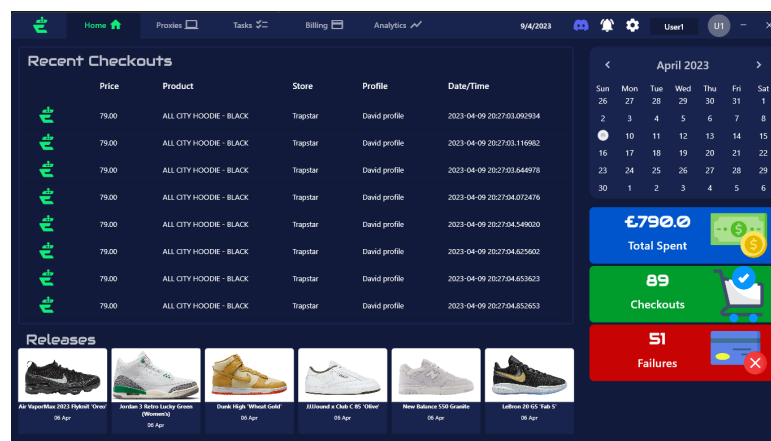


Figure 20: Final Home Screen Layout

In order for the functionality of the home screen and the top bar to operate I had to create separate classes outside of the main interface code.

9.2.4 Tab Bar

A class was created to manage the states of the screen and update the widget tree according to what Tab was clicked. Each screen was mapped to an index and when a tab was pressed, the index would show which screen the user intends the interface to navigate to.

```
Map<int, Widget> page_map = {
  0: mainScreen(),
  //1: ProxiesScreen(),
  1: tasks_screen(),
  2: ProfilesScreen(),
  3: AnalyticsScreen()
};
```

Figure 21: The map for creating the tab

```
import 'package:epsilon_gui/screens/components/TopBar_.dart';
import 'package:flutter/material.dart';

You, 3 months ago | 1 author (You)
class TabbarIndex with ChangeNotifier{
  int index = 0;
  late TabController tabController;
  TopBar topbar = new TopBar();

  int get current_index => index;
  TabController get controller => tabController;
  TopBar get this_TopBar => topbar;

  void setIndex(int newIndex){index = newIndex;}
  void setTopBar(TopBar newtopbar){topbar = newtopbar;}
}
```

Figure 22: The class outside of the widget tree that manages the state of the index across the interface

9.2.5 Tasks Screen

Tasks

In our system, we have designed a task instance class to handle individual tasks. This class stores all the data and actions for each task, such as product name, profile, size, and more. Additionally, it includes unique methods for starting tasks, ending tasks, creating a visual representation of the task, and any getters and setters for the variables.

```
class TaskInstance with ChangeNotifier {
  int taskID = 0;
  String taskproduct = "";
  String taskstore = "";
  String tasksize = "";
  String taskcategory = "";
  List<String> taskkeywords = [];
  String taskregion = "";
  Profile taskprofile;
  String taskPrice = '';
  String tasktype = "";
  String taskstatus = "Ready";
  bool check = false;
  bool isactive = false;
  DataRow taskRow = DataRow(cells: []);
  TasksLists taskList = TasksLists();
  int completed_int = 0;
  int failed_int = 0;
  bool tor = false;

  UserData data = UserData.instance;
  RecentCheckoutProvider recent_checkouts = RecentCheckoutProvider.instance;
}
```

Figure 23: Task Instance class

Tasks Groups

We have also implemented a task group feature that allows users to create a collection of tasks. The task group class stores all the data for the tasks, enabling them to be recreated when the task group button is pressed. This functionality helps users avoid having to recreate the same set of tasks if needed later, as all tasks within the group are displayed and ready to begin when selected.

```
class TaskGroup {
    int tasknum;
    String store;
    String product;
    ProfileGroup profile;
    String size;
    String groupName;
    List<String> taskkeywords;
    TaskGroupWidget widget = TaskGroupWidget(
        groupName: "",
        tasknum: 0,
        uniqueKey: UniqueKey(),
        taskstore: "",
        taskproduct: "",
        taskprofile: ProfileGroup(UniqueKey(), "", []),
        tasksize: "",
        keywords: [],
        taskOption: "",
    ); // TaskGroupWidget
    UniqueKey key;
    int checkouts = 0;
    int fails = 0;
    String option = "";

    TaskGroup(this.groupName, this.product, this.profile, this.store, this.size,
        this.tasknum, this.key, this.taskkeywords, this.option) {
        widget = TaskGroupWidget(
            groupName: groupName,
            tasknum: tasknum,
            uniqueKey: key,
            taskstore: store,
            taskproduct: product,
            taskprofile: profile,
            tasksize: size,
            keywords: [],
            taskOption: option,
        );
    }
}
```

Figure 24: Task Group Class

Task manager

Lastly, a manager class called 'TasksLists' is responsible for managing both task instances and task groups. It is invoked when a user creates any task and assigns a profile from a profile group to each task, based on the profile group chosen by the user to be part of a task. This class ensures that tasks and task groups are organized and properly managed within the system.

```

List<Profile> getTaskProfiles(int taskNumber, ProfileGroup profileGroup) {
    List<Profile> profileList = [];
    bool satisfied = false;
    if (taskNumber == profileGroup.profiles.length ||
        taskNumber < profileGroup.profiles.length) {
        for (var i = 0; i < taskNumber; i++) {
            profileList.add(profileGroup.profiles[i]);
        }
        return profileList;
    } else {
        for (var i = 0; i < taskNumber; i++) {
            profileList
                .add(profileGroup.profiles[i % profileGroup.profiles.length]);
        }
        return profileList;
    }
}

/ ADDS TASKS TO THE TABLE
void addTask(
    int numOfTasks,
    String storeData,
    String productData,
    ProfileGroup profileData,
    String sizeData,
    List<String> keywords,
    bool tor) {
    List<Profile> profile_Li = getTaskProfiles(numOfTasks, profileData);
    for (var i = 0; i < numOfTasks; i++) {
        if (id_list.isEmpty == true) {
            id_list.add(1);
            Taskinstance task = Taskinstance(
                1, productData, storeData, profile_Li[i], sizeData, keywords, tor);
            tasks_instances.add(task);
            current_tasks_list.add(task.taskRow);
        } else {
            id_list.add(id_list.last + 1);
            Taskinstance task = Taskinstance(id_list.last, productData, storeData,
                profile_Li[i], sizeData, keywords, tor);
            tasks_instances.add(task);
            current_tasks_list.add(task.taskRow);
        }
        tasks_instances.forEach((task) {
            task.setParent(getSelf());
        });
    }
    notifyListeners();
}

```

Figure 25: Task List class (manager)creating the tasks and serving out profiles



Figure 26: Final Tasks Screen Layout

9.2.6 Profiles Screen

Profile Instance

The system also includes a profile instance that stores all the data related to a user's profile, such as card number, card name, address, and so on. Each profile instance is equipped with methods required for operation, including getters and setters for each value, as well as a method to create a visual representation of the card when displayed on the interface.

```

class Profile with ChangeNotifier {
  String first_name = "";
  String last_name = "";
  String profile_name = "";
  String card_name = "";
  String card_number = "";
  String address = "";
  String city = "";
  String postcode = "";
  String phone = "";
  bool checked = false;
  UniqueKey key = UniqueKey();
  Profile_card profileCard = Profile_card(
    profile_name: "",
    card_name: "",
    address: "",
    card no: "",
  );
  int ProfileID = 0;
  ProfileGroupProvider GroupProvider = ProfileGroupProvider.instance;
  //BuildContext build_context;
  DataRow dataRow = DataRow(cells: []);
}

```

Figure 27: Profile Instance Class

Profile Groups

In addition to individual profiles, we have implemented a profile group feature, which allows users to combine multiple cards into a single profile group. This functionality enables users to create multiple tasks that will be purchased using various cards available within the selected profile group. The profile group stores a list of profiles under a unique name and key, ensuring that each group is easily distinguishable.

```

class ProfileGroup {
  UniqueKey uniqueKey;
  List<Profile> profiles;
  ProfileGroupWidget widget = ProfileGroupWidget(
    groupName: "", profileLength: 0, uniqueKey: UniqueKey()); // ProfileGroupWidget
  String name;

  int get profileLength => profiles.length;

  ProfileGroup(this.uniqueKey, this.name, this.profiles) {
    widget = ProfileGroupWidget(
      groupName: name, profileLength: profiles.length, uniqueKey: uniqueKey);
  }

  void setName(String newName) {
    name = newName;
  }

  void updateWidget() {
    widget = ProfileGroupWidget(
      groupName: name, profileLength: profiles.length, uniqueKey: uniqueKey);
  }

  void setProfiles(List<Profile> newProfiles) {
    profiles = newProfiles;
  }

  void removeProfile(Profile profile) {
    profiles.remove(profile);
  }

  bool contains(Profile profile) {
    bool check = false;
    for (Profile profile_ in profiles) {
      if (profile == profile_) {
        check = true;
        return check;
      } else {
        check = false;
      }
    }
    return check;
  }
}

```

Figure 28: Profile Group Class

Profile Manager

Finally, a ProfileProvider class manages all the profiles in a manner similar to the TaskList class. This class includes helpful methods such as removeAllProfiles, createProfileInstance, and removeProfile, which enable efficient management of profiles and profile groups within the system.

```

class ProfileProvider with ChangeNotifier {
  static final ProfileProvider _instance = ProfileProvider._internal();
  ProfileProvider._internal();
  List<Profile> all_profile_instances = [];
  List<Profile_card> all_profile_cards = [];

  static ProfileProvider get instance => _instance;

  void removeProfile(Profile_card profileCard) {
    all_profile_cards.remove(profileCard);
    var removeIns;
    for (Profile instance in all_profile_instances) {
      if (instance.profileCard == profileCard) {
        //all_profile_instances.remove(instance);
        removeIns = instance;
      }
    }
    if (removeIns != null) {
      all_profile_instances.remove(removeIns);
      ProfileGroupProvider.instance.removeFromProfileGroups(removeIns);
    }

    notifyListeners();
  }

  void removeAllProfiles() {
    all_profile_cards.clear();
    all_profile_instances.clear();
    notifyListeners();
  }

  void createProfileInstance(
    String fname,
    String lname,
    String pname,
    String cname,
    String cnum,
    String address_,
    String city_,
    String postcode_,
    String phone_,
    Profile_card cardWidget,
    ProfileGroupProvider groupProvider) {
    Profile profileInstance = Profile(fname, lname, pname, cname, cnum,
      address_, city_, postcode_, phone_, cardWidget);
  }
}

```

Figure 29: Profile Provider (Manager) Class

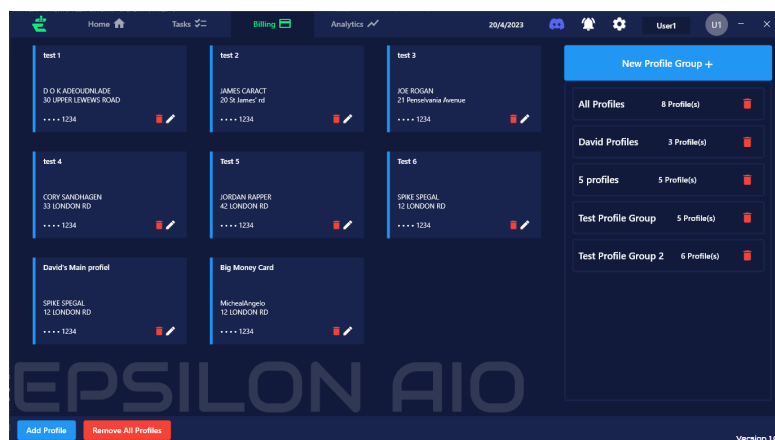


Figure 30: Final Profiles Screen Layout

9.2.7 Analytics Screen

The Analytics Screen required an analytics provider that stored the data given to it by the user and sent it across the websocket. I will delve into how it was sent across the websocket in the 'Websocket connection' section but it is worth mentioning the class that backed that process. The data that is sent activates a Future, which is a class that waits for data. While the data is being calculated, an animation is played. The class has the usual methods to set and get the sentiment number, the sentiments analysed and any keywords.

```
class Analytics with ChangeNotifier {
  static final Analytics _instance = Analytics._internal();
  Analytics._internal();
  Map<String, dynamic> recieved_data = {};
  String product = '';
  List<String> keywords = [];
  int sentimentNumber = 0;

  static Analytics get instance => _instance;

  void setProduct(String newProduct) {
    product = newProduct;
  }

  void setKeywords(List<String> newKeywords) {
    keywords = newKeywords;
  }

  void setSentimentNumber(int newSentimentNumber) {
    sentimentNumber = newSentimentNumber;
  }

  void setParams(String product, List<String> keywords, int sentimentNumber) {
    setProduct(product);
    setKeywords(keywords);
    setSentimentNumber(sentimentNumber);
  }

  Future<Map<String, dynamic>> getSemanticAnalysisResult() async {
    Completer<Map<String, dynamic>> completer = Completer();
    try {
      IWebSocketChannel? channel;

      try {
        channel = IWebSocketChannel.connect('ws://localhost:679');
        if (channel == null) {
          throw const SocketException("");
        }
      } catch (e) {
        print("Error on Connecting to server");
      }

      var map = jsonEncode(dataSentMap(product, keywords, sentimentNumber));
      channel?.sink.add(map);
      channel?.stream.listen((event) {
        recieved_data = jsonDecode(event);
        completer.complete(recieved_data);
      });
    } on SocketException catch (e) {
    }
  }
}
```

Figure 31: Analytics class

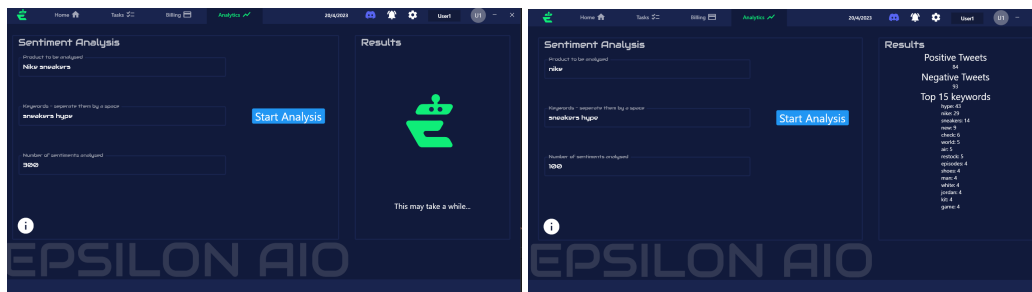


Figure 32: Final Analytics Screen Layout

9.3 Bot-Web Interaction

In order for the software to interact with websites that the user wants to purchase items off, the GUI (dart application) has to interact with the JavaScript file. From there we had to use the inputs from the user to decide which was the right product on the website and then lastly, we had to interact with the website using the puppeteer to click on the right elements and input the right data all while bypassing security. Each section below details the key implementations to achieve this goal.

9.3.1 Web socket connection

Web Sockets is a communication protocol that enables bidirectional, real-time communication between a client and a server over a single connection. Web Sockets offer several advantages over HTTP, long polling or SSE that influence my decision when choosing to use Web Sockets.

The most influential factor in my decision to use web sockets was because it allows for Bidirectional communication. Web sockets enable full-duplex communication, meaning that both the client and the server can

send and receive data simultaneously without having to establish multiple connections. This was essential for this project as each task needs to update the GUI about the current stage of purchase for each task so bidirectional communication simplifies this problem. Besides this, Web Sockets uses a binary framing protocol that is more efficient than the text-based protocols used by HTTP or SSE. For this project, I created a function to listen for a connection and another to send data when that connection is available.[7]

```
void startTask() async {
  try {
    IOWebSocketChannel? channel;

    try {
      channel = IOWebSocketChannel.connect('ws://localhost:679');
      if (channel == null) {
        updateTask("Error");
        throw const SocketException("");
      }
    } catch (e) {
      print("Error on Connecting to server$e");
    }
    var map = jsonEncode(createTaskMap());
    channel?.sink.add(map);
    //channel?.sink.add(taskkeywords);

    isactive = true;
    channel?.stream.listen((event) {
      print(event);

      if (checkIfInt(event)) {
        taskPrice = event;
      } else {
        updateTask(event);
      }
    });
  } on SocketException catch (e) {
    print("Error on Connecting to server$e");
  } on PlatformException catch (e) {
    print('error: ${e.details}');
  }
}
```

Figure 33: Sending user data to port 679 through dart

This function waits for the connection to be stable and send the task data. This data came in the format of a map which contained all the details needed to process the purchase and what type of event this was:

- Event Type - whether this was for task creation, a shoe data request or an analysis request
- Product Name
- Product Size
- Website
- Task ID - needed so the right task status is updated
- User Name - first name and last name
- Bank Card Name
- Bank Card Number
- User Address
- City Of Address
- Postcode
- Phone number
- Keywords - used to help the machine learning algorithm to find the correct product if the name isn't sufficient
- TOR functionality - whether the user wants tor to be used or not

```

async function connection_websockets() {
  const wss = new WebSocketServer({ port: 679 });

  Complexity is 10 It's time to do something...
  wss.on('connection', async (ws) => {
    console.log('New WebSocket connection');

    Complexity is 9 It's time to do something...
    ws.on('message', async (data) => {
      console.log('data: %s', data);
      const requestData = JSON.parse(data);

      switch (requestData.eventType) {
        case 'Task Creation':
          await handleTaskCreation(ws, requestData);
          break;
        case 'Release Data Request':
          await handleReleaseDataRequest(ws);
          break;
        case 'Semantic Analysis Request':
          await handlesSemanticAnalysisRequest(ws,requestData);
          break;
        default:
          console.log('Unsupported event type');
      }
    });
  });

  wss.on('close', () => {
    console.log('Connection closed');
  });
}

```

Figure 34: Javascript function listening for a connection

This function listens for a connection and waits for data to be sent and uses the data sent to handle what function the script should perform next. In this case, as a result of the event type being 'Task Creation', the handleTaskCreation() function will be called to start each task on it's own thread.

9.3.2 Multi-threading

Multi threading is beneficial in order for the bots to access the website quickly and work in parallel so that if one thread encounters an issue, the others can continue working which increases the overall reliability of the software.[2]

```

async function handleTaskCreation(ws, task_data) {
  const website = task_data.store;
  switch (website) {
    case 'Trapstar':
      threadManager.runTrapstarBotInNewThread(task_data, ws);
      break;
    case 'End-Clothing':
      break;
    case 'Palace-Clothing':
      threadManager.runPalaceBotInNewThread(task_data,ws);
      break;
    default:
      console.log('Website not supported');
  }
}

```

Figure 35: Calling the thread function

```

import { Worker } from 'worker_threads';
import { parentPort, workerData } from 'worker_threads';
import trapstarBot from './trapstar.js';

const { task_data } = workerData;
const ws = {
  send: (data) => {
    parentPort.postMessage({ type: 'ws', data });
  },
};

trapstarBot(task_data, ws);

```

Figure 36: Starts the bot-web interaction script on a new thread

The multithreading aspect of this project consists of two parts: the main code and the worker code. The main code uses a function called `runTrapstarBotInNewThread` which takes `task_data` and `ws` as arguments which is the data needed for the script to run and the websocket that the script can report to about the stage of the script. The `runTrapstarBotInNewThread` function creates a new worker thread to run the code in the `trapstarWorker.js` file by initializing a new Worker Instance. In this function there are also three event listeners for the worker:

- **Message** - When the worker posts a message, the event is triggered. If the message type is `ws`, it sends the message data to the Websocket (`ws`) otherwise it logs the message to the console
- **Error** - If an error occurs in the worker, this event is triggered and the error is logged in the console.
- **Exit** - When the worker exits the thread, this event is triggered. If the worker exits with non-zero code, an error is logged to the console.

The worker code creates a custom WebSocket-like object `'ws'` with a `send` method that uses `'parent-Port.postMessage'` to send the message to the main thread with the `'ws'` type and data. Lastly, the `trapstarBot` function is called with `'task_data'` and the custom WebSocket object. This function then processes the purchase.

9.3.3 Bypassing security

The main feature of this software is that it is able to circumnavigate anti-bot protection measures of `trapstar.com` and `palaceskateboards.com`. Like many websites on the internet today, these websites use Shopify as a platform to host their website and sell their products. As a result, many Shopify websites use similar methods to protect against bots, some of these include:

- **CAPTCHAS** - Shopify uses CAPTCHA challenges to verify whether a user is a human or a bot when it is difficult to differentiate already. CAPTCHAs usually require users to complete tasks that are difficult for bots, such as identifying objects in images or solving puzzles.
- **Rate limiting** - Shopify can limit the number of requests that can be sent from an IP address or user account per session. This prevents bots from overwhelming the website with excessive requests, which can lead to service disruptions, DOS attacks and other forms of abuse.
- **User-Agent analysis** - Shopify examines the User-Agent string in HTTP requests to identify patterns associated with bots. By detecting non-human User-Agent strings, Shopify can block, require a CAPTCHA from or limit requests from suspected bots.
- **Behaviour analysis** - Shopify monitors user behaviour on their platform, looking for patterns that are typical of bots. Examples of this are keystroke timing, mouse movements and scrolling behaviour.
- **Header Information** - HTTP headers play an essential role in securing Shopify sites against bots by blocking suspicious requests before they even reach the site. Some important headers are cookies and session headers, User-Agent header and the referrer header.

[11]

In order to avoid to bypass these methods I used a Node.js Library developed by Google, named `puppeteer`, that provides a high-level API to control headless or full browsers programmatically. `Puppeteer` enables developers to automate browser actions like clicking buttons, filling out forms, retrieving data, and navigating between pages. I used the `Stealth` and `Evasion` `puppeteer` plugins which enhance bypassing detection by modifying the browser to avoid mechanisms commonly used by websites to detect bots. In addition to this, I changed the arguments of the browser, implemented the Tor network proxy, and modified the code in a way to account for the behaviour analysis[16]

Stealth Plugin The `Stealth` plugin is designed to mimic human browsing behaviour and evade detection by websites that use fingerprinting techniques or other measures to identify and block any bot traffic. The `Stealth` plugin works by modifying the browser's User-Agent string, disabling features that are commonly blocked on websites, like `WebGL` and `WebRTC`, and simulating human-like mouse and keyboard events. `WebRTC` can give away your true location through your IP address and `WebGL` can leak information about your real device.

[16]

Evasion Plugin The Evasion plugin is designed to help puppeteer bypass anti-bot mechanisms that use JavaScript-based detection methods. Evasion works by modifying the browsers JavaScript environment, such as the navigator object or the window object to emulate a more realistic browsing context. Techniques used by this plugin include:

- Language and Timezone Spoofing: Mimicking the user's language and timezone settings to make the browser appear more like a real user.
- Navigator Plugins and MIME Types: Modifying the list of browser plugins and MIME types to resemble those of a legitimate browser.
- Media Devices Spoofing: Spoofing the list of media devices (e.g., cameras and microphones) to make the browser appear more like a real user.
- Permissions and Notifications: Faking the behavior of browser permissions and notifications to appear like a real user.
- Uses an Headless browser.

[16]

These plugins were implemented by importing them and applying them to the browser the bot is using:

```
import StealthPlugin from 'puppeteer-extra-plugin-stealth';
import EvasionsPlugin from 'puppeteer-extra-plugin-stealth';

puppeteer.use(StealthPlugin());
puppeteer.use(EvasionsPlugin());
```

Figure 37: Implementation of the plugins

Browser Arguments I used a list of browser arguments that add to the circumnavigation:

- `--no-sandbox` and `--disable-setuid-sandbox`: These arguments disable the sandboxing feature, which isolates the browser's processes from the rest of the system. Disabling the sandbox can be a security risk, but it may be necessary in certain environments where the sandbox cannot be enabled.
- `--disable-infobars`: This argument disables the yellow infobar that appears at the top of the browser when an extension is installed.
- `--window-position=0,0`: This argument sets the initial position of the browser window to (0,0), the top left corner of the screen.
- `--ignore-certificate-errors` and `--ignore-certificate-errors-spki-list`: These arguments disable certificate error checking, which can be useful for testing or accessing sites with invalid or self-signed certificates.
- `--disable-dev-shm-usage`: This argument disables the use of `/dev/shm` shared memory in the browser, which can cause issues in some environments.
- `--disable-extensions`: This argument disables all extensions in the browser.
- `--disable-features=site-per-process`: This argument disables the "site isolation" feature, which isolates each website's rendering process for security reasons.
- `--disable-hang-monitor`: This argument disables the browser's hang monitor, which can cause false positives in some cases.
- `--disable-ipc-flooding-protection`: This argument disables the IPC (Inter-Process Communication) flooding protection feature, which can cause performance issues in some cases.
- `--disable-renderer-backgrounding`: This argument disables the browser's background rendering feature, which can cause performance issues in some cases.
- `--disable-breakpad`: This argument disables the "crashpad" feature, which generates crash reports for the browser.
- `--disable-notifications`, `--disable-translate`, and `--disable-save-password-bubble`: These arguments disable various browser features, such as notifications, translation, and password saving prompts.

- `--disable-background-timer-throttling` and `--disable-backgrounding-occluded-windows`: These arguments disable browser features that limit resource usage for inactive tabs or background processes.
- `--disable-client-side-phishing-detection` and `--disable-popup-blocking`: These arguments disable browser features that detect phishing attempts or block popups.
- `--disable-features=IsolateOrigins,site-per-process`: This argument disables the "isolate origins" feature, which isolates resources from different origins in the browser.
- `--disable-bundled-ppapi-flash`: This argument disables the bundled Pepper API Flash plugin.
- `--disable-web-security` and `--allow-running-insecure-content`: These arguments disable browser security features that prevent running insecure content or scripts.
- `--disable-features=Translate,InfiniteSessionRestore,SimpleCacheBackend`: This argument disables various browser features, such as translation, infinite session restore, and simple cache backend.
- `--disable-sync`: This argument disables synchronization with Google services.

[15]

Tor As mentioned before, the Tor network is a decentralized, open-source system to facilitate anonymous communication. It works by routing users' internet traffic through a series of nodes distributed across the globe.

When a user connects to the Tor network, their data is encrypted and sent through a series of randomly selected relays before reaching its intended destination. Along the way each relay adds a layer of encryption to the data. At each relay, one layer of encryption is removed, revealing only the address of the next relay in the chain. This process ensures no single relay has access to both the source and destination of the data which preserves anonymity.

The benefits of incorporating Tor into this project is that by routing requests through multiple relays using different IP addresses, Tor helps the software avoid IP-based blocking mechanisms employed by websites and if one worker in a thread does fall victim to IP blocking, then the worker can be restarted and a new IP will be given to the worker. Overall the effectiveness and the resilience of the bot is enhanced as a result of the Tor implementation. It is important to mention that in order for a user to use the TOR feature, they will have to download Tor proxy on their computer.[22]

Tor is implemented in the code like this:

```
'--proxy-server=socks5://127.0.0.1:9150'
```

Figure 38: The Tor proxy server is added to the browser arguments

User Agent Spoofing User agent spoofing is a technique used by bots to evade detection by mimicking the user agent string of a legitimate web browser. The user agent string is a piece of metadata that is included in HTTP request headers and identifies the browser and operating system used to access the website. By altering the user agent string the bot can masquerade as a legitimate user and bypass any bot detection measure that rely on this metadata. The user agent string is commonly modified to mimic that of popular web browsers like Chrome, Firefox or Safari. User agent spoofing can bypass rate limiting detection, a method used by websites to impose rate limits on requests from certain user agents to prevent excessive use of resources.[23]

```
export async function newPage(browser){
  const page = await browser.newPage();
  const userAgent = ['Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36',
  ];
  await page.setUserAgent(userAgent[Math.floor(Math.random() * 3)]);
  return page;
}
```

Figure 39: User Agent spoofing implementation

There is a selection of the three most common user agent strings, as of April 2023, and through the use of a pseudo-random number generator, a 'random' user agent is chosen and applied that an instance of a bot.

Bot Behaviour Shopify uses behavioural analysis to detect and block bot activity on their platforms. It involves monitoring user behaviour and interactions with the website to identify patterns that are typical of bots. The specific things they monitor are Mouse movements, keystroke timings and scrolling behaviour. Throughout my script I mimicked the website movements and interactions of a human by adding in delays and pauses throughout the code so that the websites sees a regular human instead of a bot. [3]

```
await botPack.sleep(5000);
await page.keyboard.type("davidodunlade@hotmail.co.uk",{delay: 10});
await page.waitForSelector('input[value="Check out"]');
```

Figure 40: All types of delays

9.3.4 Choosing the right product

For this project to achieve it's goal of getting to checkout and bypassing security, we had to develop a way to find the right product from a users input and keywords. We added keywords to help the similarity functions find the right product if the user knew some details about the product but not the exact name. I will expand on the methods for retrieving the products data then how we calculated the right product.

Many Shopify websites use JSON to store user data and this data is accessible by adding '/products.json' to the URL. On top of this if we add '?limit=1000000' we set the limit of the query to be 1000000 which is a large enough limit for the websites we were using. All the details about the products are stored here including the ID, product name, handle and sizes available. We parsed this JSON data and created a list of product names that are used when calculating the Cosine and Semantic Similarity. After using the similarity functions to get the right product, we used the size data given by the user to get the correct handle string that would bring the bot to the add to cart page of the product with the right size already selected.

```
async function processCheckout(socket,data,page){
  await page.setDefaultNavigationTimeout(60000);
  await page.goto("https://uk.trapstarlondon.com/products.json?limit=1000");
  var innerText = await botPack.getInnerText(page);
  const result = await botPack.getProductDetails (innerText,data.product,data.size,data.keywords);
  var handle = result.handle;
  var size_id = result.sizeId;
  var price = result.price;
  socket.send(price);

  await page.goto("https://uk.trapstarlondon.com/products/"+handle+"?variant="+size_id );
  //await sleep(2000);
  await page.waitForSelector("#addToCart-product-template");
}
```

Figure 41: Code Snippet showing the retrieval of the products.json page

This method was solid for Trapstar.com, however, palaceskateboards.com seemed to hide some product data in their products.json page which meant that some products would not be found. As a solution to this, we used the '/search?q=' url string to construct a search query on the websites product page to return a list of products that were similar to the users input. From here we calculated the Cosine and Semantic similarity of the products returned to find the correct products.

```
function createQuery(product){
  return product.toLowerCase().split(' ').join('+');
}
Complexity is 7 It's time to do something..
async function selectSize(data,socket,page){
  //await page.click("#product-select");
  const size = getCorrectSizeFormat(data.size);
  try {
    Complexity is 4 Everything is cool!
    await page.evaluate((size) => {
      const selectElement = document.querySelector("#product-select");
      const options = selectElement.options;

      for (let i = 0; i < options.length; i++) {
        if (options[i].textContent === size) {
          selectElement.selectedIndex = i;
          break;
        }
      }
    }, size);
    const str = "Add to Cart";
    await page.click("input[value="+str+"]");
  } catch (error) {
    console.log("Failed");
    socket.send("Failed");
    console.log(error.message);
  }
}
```

Figure 42: Code Snippet showing the construction of the search query

During the process of coding the process of finding a product on a website, I ran into a problem that may occur. Initially, if a user did not input the exact same name as the product then the script would fail and send an error explaining that the product could not be found. At first, I decided to code a solution for this problem that calculated the JaroWinkler Distance between two sets of strings made a selection based on the value that was the highest. The JaroWinkler distance is a string similarity measure used to compare and calculate the similarity between two text strings. The Jaro-winkler is calculate in two main steps:

Finding the Jaro Distance The Jaro distance is a measure of similarity by the way of the number of matching characters and number of transpositions.[9]

1. let $s1L$ = Length of string 1
2. let $s2L$ = Length of string 2
3. let m = the number if matching characters
4. let t = half the number of transpositions

$$\text{JaroDistance} = (1/3) * (m/|s1L| + m/|s2L| + (m-t)/m)$$

Winkler Adjustment The Winkler's adjustment improves the Jaro Distance by giving more weight to common prefixes.

1. let l = length of the common prefix (up to a maximum of 4)
2. let p = the scaling factor (typically set to 0.1)
3. let dj = JaroWinkler Distance

$$\text{Winkler Adjustment} = dj + (l * p * (1 - dj))$$

However, in application the Jaro-Winkler distance was not the best choice for matching these two texts because the Jaro-Winkler distance emphasizes character-level similarity and common prefixes which is not sufficient to capture the true similarity of two strings in this context. Furthermore, the Jaro-Winkler distance does not take semantic similarity in consideration so if a user types 'P90 Black Jacket' and the product is 'New P90 Unreal Jacket Black' the Jaro-Winkler distance could not draw them together.

After much research I came across the use of Cosine Similarity and Semantic Similarity.

Cosine Similarity Cosine similarity is a measure of similarity between two non-zero vectors in a mulit-dimension space. It calculates the cosine of the angle between the two vectors, which indicates how close they are in terms of direction regardless of their magnitude. The cosine similarity is often used to compare the similarity between documents or a search query and a document. The use of cosine similarity means that users do not input a product name that is near to the real product name on the website, as long as it alludes to the same thing then it will match them up. This is due to cosine similarities focus on the angle between vectors instead of the presence of common words or word length. Despite this, only using cosine similarity would be a problem as this measure does not recognize related concepts or synonyms. For this I implemented a measure of Semantic Similarity.[14]

1. let A & B = two vectors being compared
2. let $||...||$ = the norm of the vector

$$\text{Cosine Similarity (A,B)} = (A \bullet B) / (||A|| ||B||)$$


```

async function calculateCosineSimilarity(searchTerm, itemTitle) {
  const tokenizer = new natural.WordTokenizer();
  const searchTermTokens = tokenizer.tokenize(searchTerm);
  const itemTitleTokens = tokenizer.tokenize(itemTitle);

  const termFrequency = new natural.TfIdf();
  termFrequency.addDocument(searchTermTokens);
  termFrequency.addDocument(itemTitleTokens);

  const searchTermTfidf = termFrequency.listTerms(0).map((term) => term.tfidf);
  const itemTitleTfidf = termFrequency.listTerms(1).map((term) => term.tfidf);

  // Pad the shorter vector with zeros
  while (searchTermTfidf.length < itemTitleTfidf.length) {
    searchTermTfidf.push(0);
  }

  while (itemTitleTfidf.length < searchTermTfidf.length) {
    itemTitleTfidf.push(0);
  }

  const searchTermVector = tf.tensor1d(searchTermTfidf);
  const itemTitleVector = tf.tensor1d(itemTitleTfidf);

  const dotProduct = tf.sum(tf.mul(searchTermVector, itemTitleVector));
  const normA = tf.norm(searchTermVector);
  const normB = tf.norm(itemTitleVector);
  const cosineSimilarity = dotProduct.div(normA.mul(normB));

  return cosineSimilarity.dataSync()[0];
}

```

Figure 43: Function that calculates cosine similarity

This asynchronous function takes two strings, the search string and the product string from the website. It begins by preprocessing the input strings by tokenizing them using the WordTokenizer function from the Natural Language Toolkit library. We then compute the Term Frequency-Inverse Document Frequency (TF-IDF) for each token in each of the strings with the TfIdf class. We need this to be able to create tensors, using Tensorflow.js [19] for both vectors that can be used to calculate the dot product and norms of the vectors. Once we calculate the dot product and the norms we compute the cosine similarity by dividing the dot product by the norms of the vectors. We then return the value as a single-element array with dataSync().

Semantic Similarity Semantic similarity is a measure of how closely the meaning of two pieces of text, phrases or words relate to each other. There are various methods to perform but the method I decided to use was the word embedding approach. This method captures semantic relationships between words.

1. let A & B = two vectors being compared
2. let $||...||$ = the norm of the vector

$$\text{Semantic Similarity (A,B)} = (A \bullet B) / (||A|| ||B||)$$

**It is important to note these two functions differ due to Utilizes pre-trained models like the Universal Sentence Encoder (USE) to generate embeddings for the input text.*

```

async function calculateSemanticSimilarity(useModel, searchTerm, itemTitle) {
  const embeddings = await useModel.embed([searchTerm, itemTitle]);
  const similarity = await tf.tidy(() => {
    const e1 = embeddings.slice([0, 0], [1, -1]);
    const e2 = embeddings.slice([1, 0], [1, -1]);
    const dotProduct = tf.sum(tf.mul(e1, e2));
    const normProduct = tf.norm(e1).mul(tf.norm(e2));
    return tf.div(dotProduct, normProduct);
  });
  return (await similarity.data())[0];
}

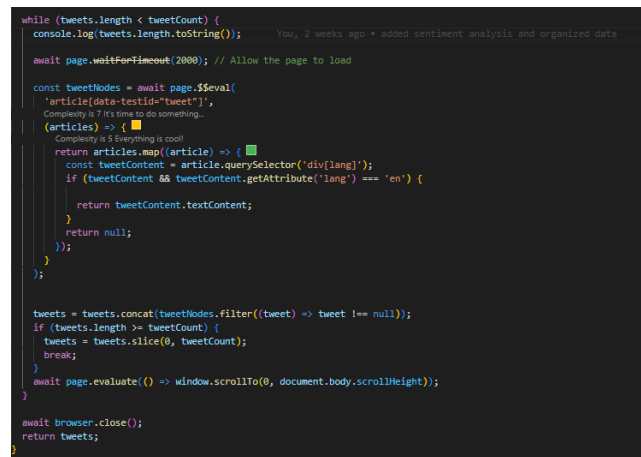
```

Figure 44: Function that calculates semantic similarity

The calculateSemanticSimilarity function asynchronously calculates the semantic similarity between two parameters, the searchTerm and the itemTitle. We use two pre-trained models to perform this. The function first generates word embeddings for both input texts using the provided model's embed() method. After we used Tensorflow.js to perform tensor operations to calculate the dot product and their euclidean norms. Lastly, we compute the semantic similarity by dividing the dot product by the Euclidean norms.

9.4 Sentiment Analysis

Sentiment analysis, is a sub field of natural language processing (NLP) that focuses on identifying and extracting subjective information from text. It involves determine the sentiment (emotional response) expressed in the piece of text given. Businesses and organizations use sentiment analysis to gauge public opinion and track social media trends in order to adjust marketing strategies among other applications. I wrote my sentiment analysis script in python, employing spaCy and the Transformers libraries to analyze the sentiment of tweets stored in a file name 'tweets.txt'. This file was initially created by using an custom on API which returns a certain number of tweets on a topic which is then saved into a txt file however this became unavailable to me as I would have to pay to continue the API calls. As a solution to this, I created a web scraper that performs the same task using puppeteer on JavaScript and saved this in tweets.txt. [10]



```
while (tweets.length < tweetCount) {
  console.log(tweets.length.toString()); // You, 2 weeks ago • added sentiment analysis and organized data

  await page.waitForTimeout(2000); // Allow the page to load

  const tweetNodes = await page.$$eval(
    'article[data-testid="tweet"]',
    (articles) => {
      // Comments is something is cool!
      return articles.map((article) => {
        const tweetContent = article.querySelector('div[lang]');
        if (tweetContent && tweetContent.getAttribute('lang') === 'en') {
          return tweetContent.textContent;
        }
        return null;
      });
    }
  );

  tweets = tweets.concat(tweetNodes.filter((tweet) => tweet !== null));
  if (tweets.length >= tweetCount) {
    tweets = tweets.slice(0, tweetCount);
    break;
  }
  await page.evaluate(() => window.scrollTo(0, document.body.scrollHeight));
}

await browser.close();
return tweets;
}
```

Figure 45: Function that gets N tweets of a topic

Algorithm:

- Import necessary libraries: Tweepy for Twitter API, asyncio for asynchronous operations, Counter for counting occurrences, re for regular expressions, json for JSON operations, Spacy for natural language processing, contractions for expanding contractions, and Hugging Face transformers for sentiment analysis.
- Load the Spacy English model
- Define expand_contractions(text) function to expand contractions in a given text.
- Define preprocess_text(text) function to preprocess the text by expanding contractions, removing URLs, mentions, and non-alphabet characters.
- Define analyze_sentiment_async(tweets) asynchronous function to analyze the sentiment of a list of tweets using the Hugging Face Transformers library, which includes loading the tokenizer and model, and preprocessing the tweets.
- Define analyze_sentiment(tweets) function to run the asynchronous analyze_sentiment_async(tweets) function using an event loop.
- Define get_tweets() function to read tweets from the 'tweets.txt' file.
- Define extract_keywords(tweets, top_n=15) function to extract the top_n keywords from a list of tweets, excluding stop words and specific terms like "tweets", "positive", "negative", and "keywords".
- Define main() function that:
 - Retrieves the list of tweets from the file using get_tweets().
 - Performs sentiment analysis on the tweets using analyze_sentiment(tweets).
 - Counts the number of positive and negative tweets.
 - Extracts the top keywords using extract_keywords(tweets).
 - Creates a dictionary containing the required data (counts of positive and negative tweets, top keywords) and prints it in JSON format.

This txt file was then used in my python sentiment analysis code. In this script, the text is initially processed through contraction expansion, the removal of URLs, twitter handles and non-alphabetic characters. Contraction expansion is a pre-processing step in natural language processing that involves converting contractions back to their original expanded form. Contractions are shortened forms of words or phrases, created by combining two words to create a shorter word with the insertion of an apostrophe. For example, "don't" is a contraction of "do not". This pre-processing is a crucial step in this analysis because it helps prepare the raw data for the algorithm. Properly pre-processed text data can lead to more accurate and reliable results from the analysis. This is because the standardized text data is more representative of the sentiments expressed. After the pre-processing, the sentiment of each tweet is asynchronously analysed using the pre-trained DistilBERT model which is fine-tuned for sentiment analysis. The number of positive and negative tweets is then calculated as well as the top N most common words after filtering out stop words, single letter words and other specific terms. Finally, this data is then printed in JSON format and sent to the user interface.[6]

```
nlp = spacy.load("en_core_web_sm")

def expand_contractions(text):
    return contractions.fix(text)

def preprocess_text(text):
    text = expand_contractions(text)
    text = re.sub(r'http|', '', text)
    text = re.sub(r'@we', '', text)
    text = re.sub(r'["A-Za-z\s]', '', text)
    return text

async def analyze_sentiment_async(tweets):
    results = []
    model_name = "distilbert-base-uncased-finetuned-sst-2-english"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(model_name)
    sentiment_classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

    for tweet in tweets:
        try:
            cleaned_tweet = preprocess_text(tweet)
            sentiment = sentiment_classifier(cleaned_tweet)[0]
            sentiment_type = sentiment["label"]
            results.append((tweet, sentiment_type))
            await asyncio.sleep(0)
        except Exception as e:
            print(f"Error: {e}")
    return results

def analyze_sentiment(tweets):
    loop = asyncio.get_event_loop()
    return loop.run_until_complete(analyze_sentiment_async(tweets))

def get_tweets():
    filename = 'tweets.txt'
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            tweets = f.read().splitlines()
            return tweets
    except FileNotFoundError:
        print(f"Error: {filename} not found.")

def extract_keywords(tweets, top_n=15):
    stop_words = nlp.Defaults.stop_words

    words = []
    for tweet in tweets:
        cleaned_tweet = preprocess_text(tweet)
        words.extend(re.findall(r'\w+', cleaned_tweet.lower()))

    # Filter out stop words and single-letter words
    filtered_words = [word for word in words if word not in stop_words and len(word) > 1 and word not in ["tweets", "positive", "negative", "keywords"]]

    word_counts = Counter(filtered_words)
    return word_counts.most_common(top_n)
```

Figure 46: Pre processing and sentiment analysis

9.4.1 DistilBERT & SST-2

In this code, the "distilbert-base-uncased-finetuned-sst-2-english" model is employed, which is a DistilBERT model specifically fine-tuned for sentiment analysis using the SST-2 (Stanford Sentiment Treebank) dataset. This dataset offers fine-grained sentiment labels, hierarchical structure, preprocessing, benchmarking capabilities, and transfer learning opportunities, making it an excellent choice for training and evaluating sentiment analysis models. DistilBERT, a smaller, faster, and lighter variant of the popular transformer-based BERT language model, offers several advantages over other models like LSTM or GRU. Firstly, its performance is commendable as it retains a significant portion of BERT's accuracy with only a small drop, ensuring high-quality results in various natural language processing tasks, such as sentiment analysis. Secondly, DistilBERT's smaller size and speed, achieved by having about 40% fewer parameters than BERT, results in faster training and execution, as well as lower memory consumption, making it ideal for resource-constrained environments. Lastly, the model's fine-tuning on the SST-2 dataset enables easy adaptation for sentiment analysis tasks, like analyzing tweets in this code, without requiring additional training. [17]

9.5 Releases Data

```

import axios from "axios"; // 53k (gzipped: 19.9k)

// Complexity is 5 Everything is cool!
export default async function request_data() {
  let date = get_current_date();
  const options = {
    method: 'GET',
    url: 'https://the-sneaker-database.p.rapidapi.com/sneakers',
    params: { limit: '100', releaseDate: 'gt:' + date, sort: 'releaseDate:asc' },
    headers: {
      'X-RapidAPI-Key': '6f3e91ea75msh7fe7a628ac745d6p1fd6f6fjsn22fd7b9b9ac4',
      'X-RapidAPI-Host': 'the-sneaker-database.p.rapidapi.com'
    }
  };

  return axios.request(options)
    .then(function (response) {
      console.log("SAVED");
      return response.data.results;
    })
    .catch(function (error) {
      console.error(error);
    });
}

function get_current_date() {
  const currentDate = new Date();
  const year = currentDate.getFullYear();
  const month = String(currentDate.getMonth() + 1).padStart(2, '0');
  const day = String(currentDate.getDate()).padStart(2, '0');
  return `${year}-${month}-${day}`;
}

```

Figure 47: API request

The releases data section is a useful segment of the GUI that gives users the names, dates and the image of a shoe that is dropping on the day of using the software. In order to create this feature I used a similar method as the sending and receiving data for the creation of tasks. I implemented a web socket connection to port 679 and sent a request for the releases data by changing the event type on the map sent across the connection to be a 'Release Data Request' as demonstrated in **Figure 16**. I then used 'The Sneaker Database' API from rapidapi.com and created a function that sends an API request once a day and stores that data in JSON which is sent across the connection and handled on the GUI side. So that this is shown to the GUI on start up, the websocket connection is created on start up. This data is then displayed on the user interface with a provider that waits for the data to be recieved, a Future.

```

class ReleasesData with ChangeNotifier {
  List<dynamic> recieved_data = [];
  List<Map<String, String>> releases_data = [];

  String data = "";

  List<Map<String, String>> get releasesData => releases_data;

  Future<List<Map<String, String>>> getData() async {
    bool is_data = false;
    Completer<List<Map<String, String>>> completer = Completer();
    try {
      IOWebSocketChannel? channel;

      try {
        channel = IOWebSocketChannel.connect('ws://localhost:679');
        if (channel == null) {
          throw const SocketException("");
        }
      } catch (e) {
        print("Error on Connecting to server$e");
      }

      var map = jsonEncode(dataSentMap());
      channel?.sink.add(map);
      channel?.stream.listen((event) {
        print('Data Recieved');
        recieved_data = jsonDecode(event);
        List<Map<String, String>> data = CreateMap(recieved_data);
        completer.complete(data);
      });
    } on SocketException catch (e) {
      print("Error on Connecting to server$e");
      completer.completeError(e);
    } on PlatformException catch (e) {
      print('error: ${e.details}');
      completer.completeError(e);
    }

    return completer.future;
  }
}

```

Figure 48: Release data class that manages all the data flow

10 Testing

Testing Process

The complete software will now be tested after implementation to ensure that it meets the functional and non-functional requirements specified in the requirements section. The testing process consisted of:

- System Testing - involves testing the entire software application as a whole and ensures that the software meets its functional and non-functional requirements.
- Acceptance Testing - involved asking the participant whether they believe that the software they tested met with out non-functional requirements

User Testing

We gave access to every user that agreed to test our application and sent them a list of activities with a list a products they could 'purchase'. The Testing requirements we sent are shown in Appendix C. After the results were given back by the user we asked the participant a few questions to see if they believe that we passed our non-functional requirements.

10.1 Test Results

Overall every participant was successful and passed 21/21 test instructions. All 3 participants had no issue except one user whose laptop froze when using the sentiment analysis part of the application which suggests a large amount of processing power being used when this process is run. Every user agreed that the User interface was magnetic, was well designed and was very responsive however, one user did not think that the application was straight forward as there is some knowledge that is required to understand botting and the stages of using the application.

11 Conclusion

11.1 Evaluation of Process and Software Objectives

The project process was personally rewarding and successful, as it involved the development of a windows software application from scratch and the application of Artificial Intelligence throughout the project. This experience allowed me to develop vital skills for my career in software engineering including UI/UX design, software development and artificial Intelligence. I am especially thankful for my developed understanding of flutter which is highly valuable in the mobile and desktop app sector. The final product was a success as I hit my project objectives by bypassing the security of two shopping websites and creating a GUI to interface with this. Out of the 36 relevant requirements this project satisfied 34 of them with the two unsatisfied requirements being due to a lack of a user manual and the software only providing an interface to buy clothes on two websites.

11.2 How Websites Can Avoid Bots

As stated before, the intention of this project was to create a shopping bot, similar to popular bots on the market today, with a user interface and botting applications in order for readers to understand:

- The ways some bots may purchase products on websites so that these websites could avoid these bots by implementing the necessary anti-bot protection measures.
- The way that users of these bots use the application and how this GUI application links to the bot scripts.

Websites that have to protect against bots are in a tough predicament as they have to defend against bots that are pretending to be users all while real users are trying to buy the same products. In essence, this is a problem of differentiation between real users and bot users. By reflecting on the bot measures I used to bypass anti-bot protection measures, I believe these are the key measure that need to be implemented.

11.2.1 Remove the products.json from public access

In this project we use the products.json file to construct the correct URL for a product as the products.json contains all the data about all the products on the website. If we did not have the data in this project we would not be able to navigate to a product URL quickly. By letting any user have access to the products.json file, any user can construct a URL based upon the data that is given in this file. In this project, we used the products.json file to get all the user data which was then used to construct a URL based on the user input. By closing off the products.json file less users will have information about the website and products on said website to be able to create web-scraping algorithms that can access sites quickly.

11.2.2 Block any browser requests from Tor

Tor is an open-source software that enables anonymous communication over the internet. It is designed to protect users' privacy and anonymity by routing internet traffic through a series of volunteer-operated servers named relays. Blocking requests from Tor can help reduce the risk of fraudulent transactions since it is more challenging to trace users' real identity and location while using Tor. Fraudulent users may use Tor to hide their identity and location while using Tor. In addition to this, Tor can be used to bypass geo-restrictions and by blocking Tor it can help the company maintain compliance with their legal and regulatory obligations.

11.2.3 Randomise element names or change them after a period of time

In this project we puppeteer to navigate to elements by element name or id. However, all of the names and ids related to what the element actually is on the page. Although this sounds intuitive, it is actually detrimental as bidders can predict what the element name could be based upon what the bot script is attempting to click at that time. Furthermore, websites should also change the names as all the elements on the website periodically because this would make all bot scripts temporary and inefficient. The bot scripts created in this project work by puppeteer interacting with the site through id and names however, if the website changed the names of the elements then the script will only work from the previous iteration of the website.

11.2.4 Defend against User agent Spoofing

To defend against user agent spoofing, a company can employ the following methods:

- **User-Agent Analysis:** Look for inconsistencies in user-agent strings or patterns that suggest a spoofed user-agent (e.g., outdated browser versions, unusual combinations of browser and operating system, etc.).
- **Behavioral Analysis:** Monitor user behavior on the website, such as click patterns, navigation, and dwell time, to identify any anomalies that may indicate a bot.
- **Combine Multiple Detection Techniques:** Relying solely on user-agent analysis is not sufficient. Combine it with other techniques like rate limiting, CAPTCHAs, and browser fingerprinting to improve bot detection accuracy.

11.3 Skills Demonstrated and Acquired

Throughout the project, I demonstrated project management skills, such as time management, using software management software (GitHub) to keep track of progress. I also developed in depth research skills when having to delve into sentiment analysis AI and different areas of botting. I learned Dart during the implementation phase, which was initially challenging due to the new syntax however, was worth the effort and proved to be an appropriate choice as it provided access to flutter and a wide range of user interface design tools to build the graphical user interface. I also demonstrated technical skills acquired from my university course throughout the three years such as Natural Language Engineering, Further Programming and Introduction to Computer Security. As a software professional, I successfully developed a complex and extensible piece of software, gaining experience in integrating and testing its components. I worked with the agile software engineering model and employed the Model View Controller design pattern. As mentioned in the introduction, the purpose of this project was to learn how bots affect websites by implementing bot practices and creating a user interface around this to be able demonstrate how shopping bots may work. This information is valuable to companies that sell rare products online and anyone that may want to buy these rare products. I wish to emphasise that bots go against the terms and conditions of many sites so I took the right procedures so that this software did not affect any real website, business or corporation. In addition to this, the creation of this software as outlined in this paper, does not encourage the use of shopping bots to buy limited-edition products.

11.4 Project Improvements

11.4.1 Expanding the Scope of Supported Websites

One area of improvement for this project is expanding the scope of supported websites. Currently, the bot only supports two websites. To make the bot more versatile and useful, additional websites could be integrated. This would involve researching and understanding the structure and anti-bot mechanisms of each new website, and implementing specific strategies for bypassing their security measures. Additionally, seeing as the websites we worked on did not have a significant problem with botting it is possible to assume that the bot protection on these websites is not as in-depth as other websites and that the bot made for this project was 'overkill' in a sense.

11.4.2 Improving the User Interface and User Experience

Although the project successfully implemented a graphical user interface for users to interact with the bot, further refinements could be made to improve the overall user experience. This may include refining the UI design, simplifying user interactions, or adding additional features to enhance the user's control over the bot's behavior. Additionally, developing a comprehensive user manual or providing in-app help would greatly improve usability and solve the issue we had from the test participant that did not know how to navigate the interface.

11.4.3 Optimizing the Bot's Performance

The bot's performance could be further optimized to improve its speed and efficiency through the use of server requests to capture and send requests to servers which is quicker than loading the website and interacting with it. A major feature that we could implement is an Captcha solver Artificial intelligence, this would likely take the use of CNN (Convolutional Neural Networks) and a large data set to train the algorithm on.

12 Appendix A - Project Logs

Coding - 12/09/2022

I created the initial plain window for the GUI using flutter and at this time the app design was meant to be based on the prototype design

Research - 21/09/2022

research on flutter and how to code in dart to be able to program the user interface as I was out of depth

Meeting - 1/10/2022

I had a meeting with my supervisor to confirm my project and discuss any changes to the project we need to make.

Research - 15/10/2022

I did some research on the current state of shopping bots and the resale market

Research - 17/10/2022

I did some research on Selenium and HTTP Client which are the tools that I will use to interact with the website.

Interim Report - 6/11/2022

I worked on my interim report - Introduction and Requirement Specification

Interim Report - 7/11/2022

I worked on my interim report - Background Research, Software Design and Gantt Chart

Meeting - 9/11/2022

I had a meeting to look over my report and go over ethical considerations, and the addition of data analytic

Interim Report - 8/11/2022

I worked on my interim report - References, Log, Professional Considerations and Appendix

Research - 21/11/2022

research on flutter SDK(Software Development Kit) and how to use GitHub to manage my files

Coding - 26/11/2022

Started working on the GUI application again to create the home screen still according to the prototype design

Coding - 30/11/2022

Implemented the releases, sneaker news and calendar section on the application.

Coding - 04/12/2022

Finished the layout for the home screen, added the tasks screen and implemented simple navigation by a side bar.

Coding - 21/12/2022

Completed the tasks screen according to prototype layout. Implemented data management via providers to allow task data to be accessible when navigating the user interface.

Research - 04/12/2022

Researched the different server-client connections and which one would be best

Coding - 06/01/2023

Added the Web socket connection and integrated the bot to the user interface.

Coding - 11/12/2022

Created the method to pass data from the User Interface to the bot scripts via the web socket connection

Coding - 12/12/2022

Cleaned up code and made the user interface to make functions and widgets for repeated code.

Coding - 20/01/2022

Completed fully the Home screen layout

Questionnaire - 14/02/2023

Sent out the questionnaire to many contacts I have and on social media.

Questionnaire - 17/02/2023

Wrote analysis on the feedback from the questionnaire

Coding - 20/02/2023

Integrated the feedback and redesigned the home screen.

Coding - 22/02/2023

Integrated the feedback and redesigned the tasks screen.

Coding - 28/02/2023

Implemented the profiles screen and profile creation functionality.

Coding - 07/03/2023

Bug Fixes & added profile group functionality.

Research - 010/03/2023

Research on saving data after the application has been closed.

Coding - 17/03/2023

Profile groups added fully in terms of design

Coding - 20/03/2023

Profiles implemented with task creation.

Coding - 24/03/2023

User Interface monitors the number of success, failures and money spent on the User Interface.

Coding - 28/03/2023

implemented a second website, palaceskateboards.com

Testing - 30/03/2023

Let individuals test based on a test criteria

Testing - 02/04/2023

Received results for the testing.

Research - 04/04/2023

Research on models for my sentiment analysis classifier

Coding - 06/04/2023

Sentiment analysis implemented along with GUI layout of analytics page

Research - 07/04/2023

Research on How to implement multi-threading

Coding - 10/04/2023

Multi threading implemented.

Research - 11/04/2023

Research on the best method to use for a search engine like feature

Coding - 12/04/2023

NLE (Natural Language Engineering) implemented to find the correct product on the website.

Coding - 14/04/2023

Layed out the websockets better.

Research - 15/04/2023

Research on Tor and how to implement the TOR proxy into a selenium and chromedriver

Coding - 17/04/2023

Implemented Tor

13 Appendix B - Questionnaire Results

What changes would you recommend for this application?

6 responses

Make it multi lingual

lbr ... font

Possibly brighter colours,

On the profile screen, make add profiles and add group buttons more visible, they look like plain text. The home screen could be easier to read, less information packed in a small area.

Brighten up the background - stick to a simple design with a light/two colour scheme feed

Adding a walkthrough demo/ tutorials section teaching people how go use all the features

What do you think of the colour scheme used in this software application design?

6 responses

Great colours

Love it

Solid choice of colours - nothing too fancy but still effective. Could maybe use brighter colours to enhance excitement for consumer

The colour scheme is alright, however the analytics page has some colours that aren't consistent with the overall colour scheme

Not easy on the eye - looks almost computerised rather than a user friendly site

Clear. Easy to see wins abs failures

What impression does the app's design give you about the brand?

6 responses

Very little y

Purely sneaker bot, maybe add some clothes to show can be used for not just sneakers

Strong attention to detail - tireless work on format. Also shows it's very easy to use

Technology company

Literally feels like I'm coding

Committed to getting the customer what they want

If No, explain why?

2 responses

- Points mentioned previously. There is also inconsistency with the menu bar, its on the side on some pages and at the top on others. This could confuse your users
- Too complicated

If No, explain why ?

1 response

- There is no page here for adding tasks

If No, explain why ?

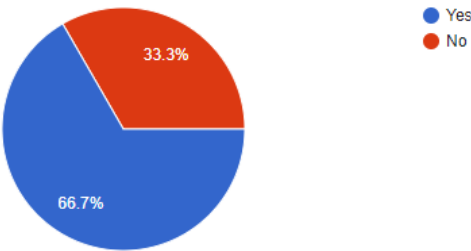
2 responses

- The buttons are not clear enough, it takes some looking around
- Home page is too busy - subheadings are not "dumb proof"

Is it clear how you would start a task?

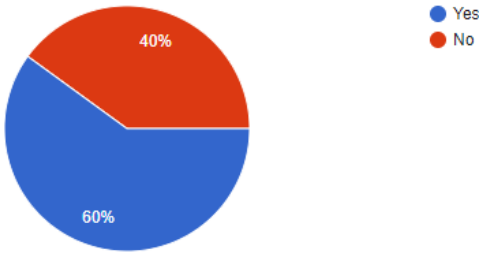
6 responses

 Copy



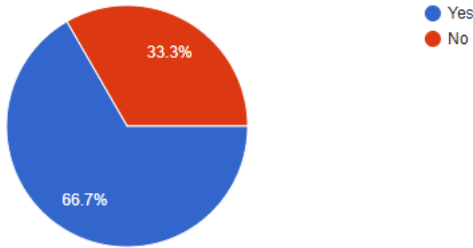
Is it clear how you would add a profile?
5 responses

 Copy



Is this a good design?
6 responses

 Copy



14 Appendix C - Test Plan Results

14.1 Bot-Web Interaction Functional Requirements

- 1.1 - Use bot on trapstar.com successfully (navigate to checkout)
- 1.2 - Use bot on palaceskateboards.com (navigate to checkout)
- 1.3 - Use TOR bot on trapstar.com successfully (navigate to checkout with Tor)
- 1.4 - Use TOR bot on palaceskateboards.com successfully (navigate to checkout with Tor)
- 1.5 - Bot uses data from GUI profiles.
- 1.6 - Bot finds correct product on website.

14.2 User Interface Functional Requirements

- 2.1 - Let user create their own task
- 2.2 - display status information about each task
- 2.3 - display information of task after purchase
- 2.4 - display upcoming sneaker information
- 2.5 - display data from sentiment analysis system
- 2.6 - be able to start task through the GUI
- 2.7 - be able to store card details on the user interface
- 2.8 - be able to create profile groups and create tasks through them

14.3 Sentiment Analysis Functional Requirements

- 3.1 - Return the number of positive and negative sentiments.
- 3.2 - Return the top 15 keywords that occur in those sentiments

14.4 Non-functional requirements

- 4.1 - Magnetic User Interface Design
- 4.2 - simple and straight-forward
- 4.3 - App is responsive

14.5 Results

All three users had all successful tests except from one instance where their computer froze due to using the sentiment analysis artificial intelligence

References

- [1] Araash Ahuja. *Entering the Sneaker Resale Industry*. 2020.
- [2] Bernardas Ališauskas. *Web scraping speed: Processes, threads and Async*. Nov. 2022. URL: <https://scrapfly.io/blog/web-scraping-speed/>.
- [3] Bernardas Ališauskas. *How to bypass perimeterx when web scraping in 2023*. Apr. 2023. URL: <https://scrapfly.io/blog/how-to-bypass-perimeterx-human-anti-scraping/#behavior-analysis>.
- [4] Bernardas Ališauskas. *Web Scraping With a Headless Browser: Puppeteer*. Dec. 2022. URL: <https://scrapfly.io/blog/web-scraping-with-puppeteer-and-nodejs/>.
- [5] Rachael Chapman. *Guide to IP rotation to avoid IP ban: Limeproxies*. Oct. 2020. URL: <https://limeproxies.netlify.app/blog/rotating-proxies-scraper>.
- [6] Jacob Devlin et al. *Bert: Pre-training of deep bidirectional Transformers for language understanding*. URL: <https://aclanthology.org/N19-1423/>.
- [7] Peter Leo Gorski, Luigi Lo Iacono, and Hoai Viet Nguyen. In: *WebSockets* (2015). DOI: 10.3139/9783446444386.fm.
- [8] *Introduction to widgets*. URL: <https://docs.flutter.dev/ui/widgets-intro>.
- [9] Matthew A. Jaro. *Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida*. 1989. DOI: 10.1080/01621459.1989.10478785.
- [10] Karen Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*. Jan. 1972. URL: <https://www.emerald.com/insight/content/doi/10.1108/eb026526/full/html>.
- [11] Karmaz95. *Karmaz95/evasion: AV evasion techniques*. URL: <https://github.com/Karmaz95/evasion>.
- [12] Rhys Lewis. *Internet sales as a percentage of total retail sales (ratio) ()*. 2022. URL: <https://www.ons.gov.uk/businessindustryandtrade/retailindustry/timeseries/j4mc/drsi>.
- [13] Sarah E Michigan. *Sneaker Bots & Botnets: Malicious Digital Tools That Harm Rather than Help E-Commerce*. 2021.
- [14] Tomas Mikolov et al. *Efficient estimation of word representations in vector space*. Sept. 2013. URL: <https://arxiv.org/abs/1301.3781>.
- [15] Peter Beverloo. URL: <https://peter.sh/experiments/chromium-command-line-switches/>.
- [16] Puppeteer. *Puppeteer/Puppeteer: Node.js API for Chrome*. URL: <https://github.com/puppeteer/puppeteer>.
- [17] Victor Sanh et al. *Distilbert, a distilled version of Bert: Smaller, faster, cheaper and lighter*. Mar. 2020. URL: <https://arxiv.org/abs/1910.01108v4>.
- [18] *Shared_preferences : FlutterPackage*. Mar. 2023. URL: https://pub.dev/packages/shared_preferences.
- [19] Nikita Silaparasetty. *The Tensorflow Machine Learning Library*. 2020. DOI: 10.1007/978-1-4842-5967-2_8.
- [20] *Terms & conditions*. URL: <https://uk.trapstarlondon.com/pages/terms-conditions-1>.
- [21] *Terms and conditions*. URL: <https://boring.palaceskateboards.com/row/terms-and-conditions>.
- [22] Inc. The Tor Project. *Tor*. URL: <https://www.torproject.org/about/overview.html.en>.
- [23] *What are the latest user agents for popular web browsers?* URL: <https://www.whatismybrowser.com/guides/the-latest-user-agent/>.
- [24] *What is human-computer interaction (HCI)?* URL: <https://www.interaction-design.org/literature/topics/human-computer-interaction>.