School of Electronic Engineering and Computer Science

Final Report

Programme of study:
MSci Computer Science

Project Title:
LEGO Price Tracker App

Supervisor:
Dr Nikos Tzevelekos

Student Name:
Filip Sroka

Final Year
Undergraduate Project 2021/22

Date: 04.05.2022

Abstract

The pandemic that started in 2020 resulted in millions of people in the UK being forced to stay at home. This played a significant role in increasing the popularity of building and collecting LEGO as a hobby. In addition, LEGO started marketing its product toward the adults. As a result, many people in the LEGO community do not have the experience to know where to buy LEGO from and what is a reasonable price for the particular set. The more experienced community members spend a few hours a day tracking and searching for LEGO deals.

This report will discuss the process of creating a desktop app that aims to help new members of the LEGO community make decisions on when to buy a LEGO set. Moreover, for people who have been involved with LEGO for a more prolonged time, it provides them with tools that will reduce their time spent searching for deals and assist them in planning their future purchases.

Contents

Table of Tables

Table of Figures

Chapter 1: **Introduction**

1.1 Background

The LEGO Group has been the largest toy company since 2015 [1]; seven LEGO sets are sold worldwide every second [2]. During the initiation period of the pandemic, as many people were confined in their homes, they began relentlessly searching for other means of entertainment, which significantly increased the popularity of LEGO [3]. This led The LEGO Group to target its products to adults by introducing more appealing designs and an 18+ age rating [4].

The existence of websites aimed at LEGO fans, such as Brickset, assists its users in cataloguing their collections. In addition to this, there is Rebrickable, which allows its users to find alternative models to build and design or sell instructions for their model. The LEGO Group also purchased BrickLink in 2019, the most prominent secondary market for LEGO products [5].

LEGO products are a more lucrative investment than gold, stocks, and bonds. This resulted in multiple newspaper articles, making investing in LEGO go mainstream. Soon Facebook groups were formed where people discuss which set will make a good investment, how many to buy, at what price to buy and when to sell by looking out for leaks and rumours about upcoming sets [6].

There are some apps dedicated purely to tracking the prices of LEGO sets like BrickSpace, BrickEconomy and Brickwatch; however, they are for different markets or are not accurate and have very few functionalities making them not practical. Those that do the general price tracking seem to do it inaccurately or do not provide the user with enough data to help them decide whether it is a good time to make the purchase. As a result, people who have only now joined the LEGO community do not have the knowledge or experience to make a good decision regarding LEGO purchases.

Furthermore, the existing consumers (already a part of the community) spend a few hours daily tracking prices and looking for deals. However, this can be automated, and all necessary information can be provided in one place, thus decreasing the time spent on LEGO deal hunting-related activities.

LEGO releases hundreds of LEGO sets every year, almost every month, making it very hard to keep track of what is currently on the market. This leads to people missing out on some sets and having regrets in the future when the given set is worth hundreds and, in some cases, thousands of pounds on the secondary market, making it unobtainable for the majority.

In order to solve this problem, a system needs to be developed that will accurately track prices, help the user make a judgement on the price, provide them with data in a user-friendly way to assist their decision and recommend them sets that they might like to prevent them from future regrets.

1.2 Problem Statement

Most of current price trackers only show the small picture to the users as they show the current price but no history of how the price changed over time. Therefore, inexperienced buyers will have a hard time deciding whether to go ahead with their purchase. Those

who provide the price history are either limited to one retailer or do not record some price changes.

Additionally, some price trackers track the prices of incorrect items. Instead of tracking the price of the LEGO set, they track the price of the LED light kit, acrylic case or stand (not LEGO branded). This results in click-baiting the user as the price of these accessories tends to be significantly lower than the actual set, thus wasting the user's time.

Current recommendation systems provide users with sets that they already might have or want (sets they are aware of). Retailers do not have access to users' owned and wanted sets which causes this issue. As a result, they must spend time browsing LEGO sets on the LEGO website to be updated on the newest releases to avoid missing out.

Lastly, few systems allow users to track prices only for selected sets. Typically, people in the LEGO community have a list of sets they intend to buy. They are only waiting for the price to drop, so currently, when using the price tracker, they are forced to enter the set ID for each set they wish to check the price as the systems that support this feature is not sufficient.

1.3 Aim

This project aims to develop a market-ready desktop app that will allow users to track LEGO sets' prices across multiple retailers. Additionally, it will tell the user how reasonable the price for that set is and display the data to back it up. Furthermore, users will import their wanted list and display the current prices for all the LEGO sets on that list. The recommendation system will use the data about users' collection to provide them with other sets they might like.

The app's unique feature will be helping the users decide which combination of sets to buy from their imported wanted list to reach the threshold needed to qualify for promotion on LEGO's website. The prices on LEGO's website are typically retail; most of the time, they are cheaper somewhere else at any point in time. Therefore, when buying from LEGO directly, people overpay. The algorithm will display the best options based on average discounted price, current cheapest elsewhere and lowest price recorded of that set.

1.4 Objectives

- Track prices - Providing the user with accurate live prices of LEGO sets across the range of retailers.
- Visualise the price changes - Display graph showing how the price changes for the LEGO set over time.
- Rating prices - Using collected data, determine how good/bad is the current price of the set of interest.
- Importing collection and wanted list - Allow users to import their collection as a CSV file which can be imported from Brickset.

- LEGO shopping assistant - An algorithm will help the users determine the best combination of sets to buy to reach the threshold to qualify for the current promotion on lego.com or LEGO Store.

- Recommendation system – An algorithm that will recommend sets to buy based on the users' collection and wanted list.

1.5 Report Structure

Chapter 2 will focus on findings during the market research phase of the project. It will compare current price tracking and deal websites to discuss more in-depth issues associated with them.

Chapter 3 will state the functions and non-functional requirements for the project.

Chapter 4 will outline the application's design, the use case diagram and the diagram showing how different components interact.

Chapter 5 will discuss the implementation, issues encountered, and actions to resolve them.

Chapter 6 will focus on testing the application.

Chapter 7 will summarise the findings and discuss how the project can evolve in the future.

Chapter 2: **Literature Review**

2.1 Price Tracking

2.1.1 Obtaining Data

To be able to track prices across multiple websites, we need to harvest the required data. Most of the retailers that have a good selection of LEGO sets do not have an API, and those that do, have a minimal number of API calls that can be done for free. The prices to increase that number are high, and unfortunately, the project does not have any budget. Due to the above reasons, we are going to use web scraping.

2.1.2 Web Scraping

Web scraping is extracting data from a website. It allows us to efficiently collect data that is publicly available on different websites without manual labour. It accesses the specified fields by the selector-based XPath, CSS selectors, or JS path. The extracted data can then be saved to a database, text, or CSV file [7].

There are many web scraping tools such as Selenium and Puppeteer. Puppeteer is the faster one, but it only supports Node.js [8]. There is also the Requests library for python, which would be even faster than puppeteer as it eliminates the need for a web browser. However, this library is more likely to be detected as a bot and will not work well with dynamic web apps, which load additional content after the page is loaded. Dynamic webs apps allow for features such as infinite scroll or show more information after pressing a button without the need of reloading the page [9].

2.1.3 How can LEGO sets be identified?

Each LEGO set ever released has a unique set ID which is general knowledge. Even if a set gets rereleased with the same box art and nothing about it got changed, it will be given a new set ID [10].

The number of digits in the LEGO sets varied over time. Up to 1980 great majority of the sets had 3-digit IDs, then until the year 2013, sets had a 4-digit IDs. Ever since, sets that are sold in stores have 5-digit set IDs. Some sets have longer IDs than 5-digit, but these sets are typically rare, exclusive sets available, for example, only to LEGO employees or people who participate in LEGO House tours that are not available from any retailer [11].

Figure 1 - Figure above depicts the circled set ID on the box of LEGO 92176 NASA Apollo Saturn V [21]

The set ID is most of the time visible at the front of the packaging, regardless of whether it is a box or a polybag. There are some exceptions, for example, Minifigure Factory. Although there is no set ID on the box, the set still has a set ID (5005358); however, like the rare exclusive sets mentioned above, these are not available from any retailer. In this case, it was a gift with purchase. In other words, a person had to spend

above a certain threshold in LEGO Store or lego.com to qualify for the promotion and receive the set free of charge.

2.2 Recommendation System

2.2.1 Types of Recommendations Systems

There are three types of recommendation systems content-based filtering, collaborative filtering, and a hybrid of the two.

Content-based filtering is based on what users indicate that they like. Then the recommendations system uses that information to provide the user with similar content. The advantage of this recommendation system is that it only requires data about the items that the user indicated. However, by doing so, it will enclose the user in their bubble by only recommending them similar items [12].

Collaborative filtering is using data about other users and comparing one user against other similar users. Using this data suggests to them what these similar users like them liked. The advantage of this recommendation system is that it will not enclose the user in a bubble which the content-based one is doing. However, to make it work, a large amount of data about multiple users is required [13].

The hybrid of the two is ideal as it would make use of both types of data to make better suggestions [14].

2.2.2 Available Data

Out of the types of recommendations systems discussed in section 2.2.1, the only feasible one is content-based filtering, as there is not enough data available to implement collaborative filtering.

Implementing a content-based filtering system requires a tagged list of all currently available LEGO products. There is a Brickset API which provides the user with such data; however, that would make the project heavily dependent on them, and Brickset could withdraw the API key as they do not allow using their API to make competitor apps.

The LEGO website has sufficient information on the product page, which can be used as tags. Additionally, they have product images that can be used to generate more tags using Google Vision API [15].

2.2.3 Ways of Calculating Similarity

When implementing content-based filtering, the tagged list of products is used to determine how similar every product is to every other product. There are many ways of calculating similarity. The appropriate metric is selected based on the type and amount of data compared. Some examples of such metrics are cosine similarity, Jaccard index and Euclid distance.

Cosine similarity calculates the cosine angle between the two vectors in multidimensional space. It disregards the magnitude of the vectors and focuses on whether they point in a similar direction. It gives a value between -1 and 1, inclusive. 1 implies the vectors are pointing in the same direction, and -1 implies vectors pointing in the opposite directions [16]. The mathematical formula is:

$$\cos(\theta) = \frac{A\,B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2}\sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Jaccard index measures similarity between two sets by calculating the intersection of two sets over their union. It gives values between 0 and 1. The closer the value is to 1, the more similar the two sets are. It works based on whether a given element exists in both sets. It is suitable for determining the similarity of qualitative data; however not as good for quantitative data as it would only check whether there is a value like that in the other set or not [17]. The mathematical formula is:

$$J(A, B) = \frac{|\ A \cap B\ |}{|\ A \cup B\ |}$$

Euclidean distance measures similarity by calculating the distance between two points in Euclidean space. It gives any natural number. To do it, it makes use of Pythagoras' theorem. Opposite to the Jaccard index is perfect for quantitative data [18]. The mathematical formula is:

$$D(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

2.3 Pricing of LEGO sets

The retail price of a LEGO set is set by The LEGO Group themselves; however, retailers can price the sets however they want [19]. Although this is a case majority of the retailers stick to the suggested retail price (in some cases, they round it to the nearest British pound).

Each LEGO set has a retail price, but also each LEGO set has an actual market value. Despite having set retail prices, some sets are typically listed for much lower. An example is a set released in 2019, LEGO 75255 Star Wars Yoda. Its retail price before 2022 was £89.99, but most of the time, it could be seen listed on multiple websites for about £72.00.

Any discount that is lower than 5% of the retail price is negligible. This is because LEGO runs a loyalty program, and each person who takes part in the program is called a “VIP”. From each purchase made from LEGO Store or lego.com, the person earns 5% cashback in “VIP points”, which can be redeemed on LEGO’s website to get vouchers or other rewards.

2.4 Competitor Apps

2.4.1 CamelCamelCamel

CamelCamelCamel is a web app that tracks prices on amazon around the world. In addition, it provides the user with information such as the lowest and highest price

recorded with a date when that happened. It also displays a graph of how the price changes over time. Furthermore, it provides Amazon Price Drop Alerts. This feature allows the user to specify when to send an email to notify them when the price of a particular item drops below the specified price.

The data displayed on CamelCamelCamel is excellent as it registers every price drop and provides users with only essential data, but it does not help them interpret it. It would tell the user if it is the best price, but this does not imply a reasonable price, making it unfriendly for new people in the LEGO community.

Figure 2 Screenshot from CamelCamelCamel page for LEGO 75255 Yoda [22]

Its most significant drawbacks are not live prices and that it is only limited to Amazon and no other retailer. Weirdly enough, despite being focused only on Amazon, it does not apply vouchers that Amazon allows to apply for certain items and thus records the item's value without the voucher. The Amazon Price Drop Alerts that it provides is an excellent idea; however, emails arrive a few hours after the price drop; therefore, any outstanding offer would be sold out by then.

2.4.2 PriceSpy

PriceSpy is a web app that tracks prices across multiple websites. It provides the user with the lowest recorded price, a graph showing the price history, user reviews, price alerts, allows the user to compare the item to other products and make a list of items to track.

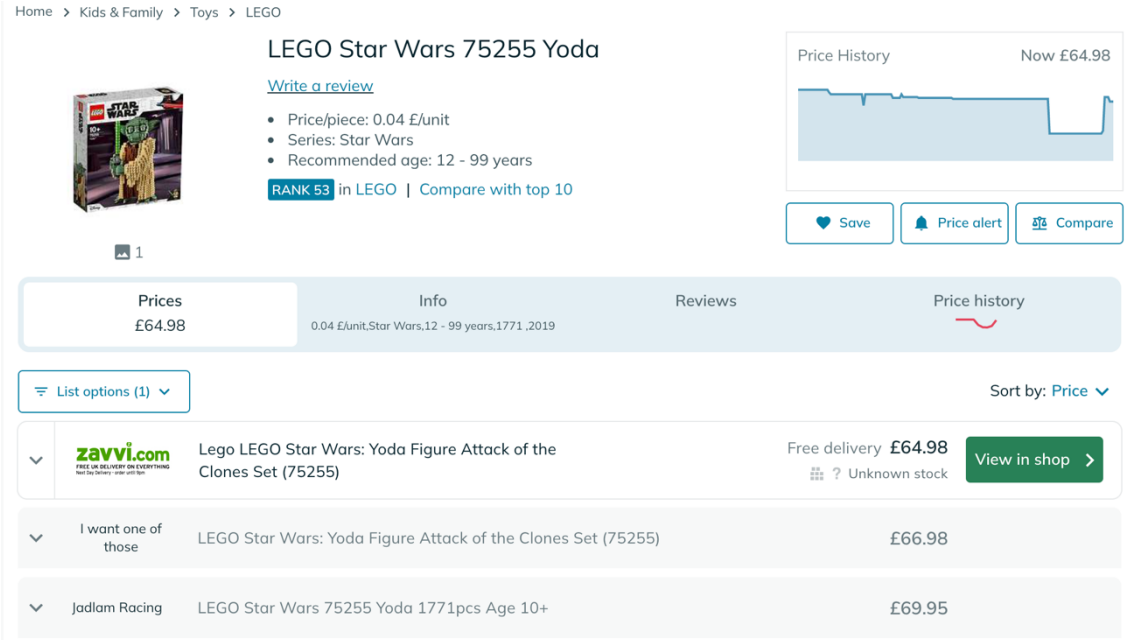

Figure 3 - Screenshot of a page from PriceSpy for LEGO 75255 Yoda [23]

The apps biggest issue is that it tracks the prices of the wrong items. For example, instead of tracking the price of LEGO Yoda 75255 on Amazon, it tracks a LED light kit for that set that is not even LEGO branded. The price of the LED kit is significantly lower than the price of the sets. This fault defeats the list's purpose as the items on the list only display the item's lowest price. As a result, the user gets click baited by the incorrect data.

Additionally, the app has a graph which shows too much information making it hard to read and understand. Similarly, to CamelCamelCamel, it does not apply Amazon vouchers when recording the price.

2.4.3 HotUKDeals

HotUKDeals is a web app that also has an app for IOS and Android. Its users post deals on this website; anyone can post a deal. Each deal can be voted hot or cold by any user. Hot means it is a good deal; cold means it is a bad deal. Under each deal, people can add comments.

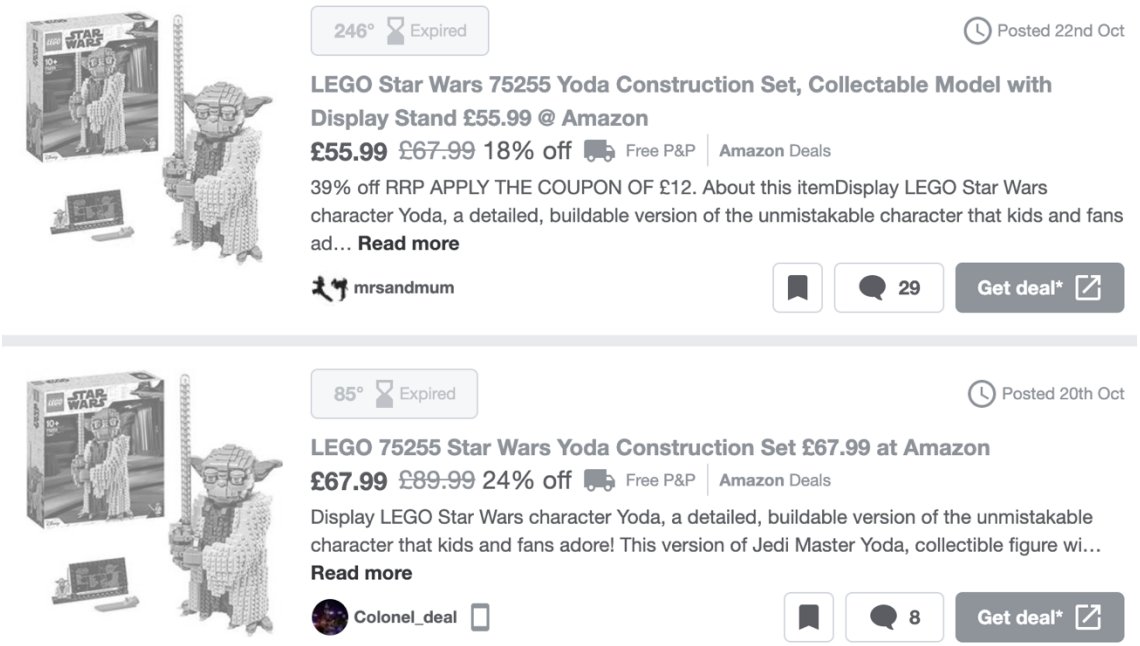

Figure 4 - Screenshot of a page from HotUKDeals for LEGO 75255 Yoda [34]

Additionally, there are threads where people can discuss the topic defined by the thread. It also provides automated price tracking, but it is provided by PriceSpy, which was discussed above. HotUKDeals uses gamification to make deal hunting more fun, and it awards users with badges by completing specific achievements.

What is excellent about HotUKDeals is the communities that are built in the threads. People can give each other advice and discuss their experiences. This is highly beneficial for HotUKDeals as it makes people stay around as they get attached and build relationships with other people. In terms of deals, they are accurate because they are done manually; as a result, they include any voucher or promotional codes found online and in-store deals.

As accurate as the prices are on HotUKDeals, the website relies on the users' input. People cannot be able to post deals faster than a web scraper would. However, a web scraper would not necessarily find all vouchers and would not find in-store deals.

2.4.4 PriceRunner

PriceRunner is a web app that tracks prices across multiple retailers. It provides users with reviews, statistics, price alerts and compares features.

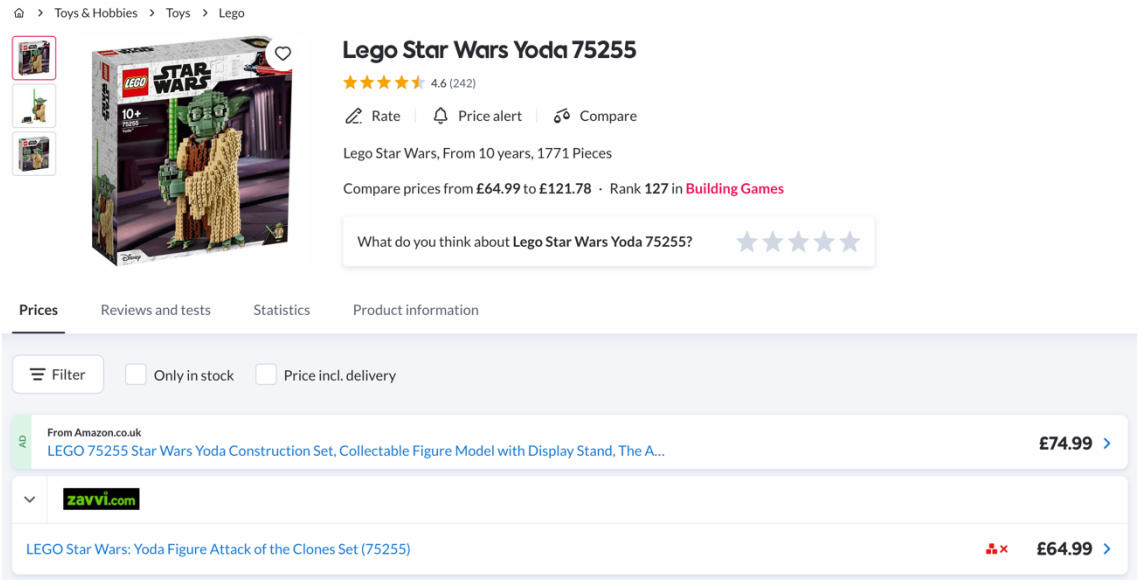

Figure 5 - Screenshot of a page from PriceRunner for LEGO 75255 Yoda [24]

This app does not inform users about the lowest recorded price or average price of the item. It provides the user with a graph; however, the user needs to click the “Statistics button” to see the price history. Although the graph is informative due to the lack of information about the lowest price and average price, a user needs to search through the graph to find the lowest price. This makes the user prone to mistakes. Lastly, it does not track prices after applying amazon vouchers.

2.4.5 Brickset

Brickset is a web app well known for making a database of LEGO products, but they also provide a price tracking feature.

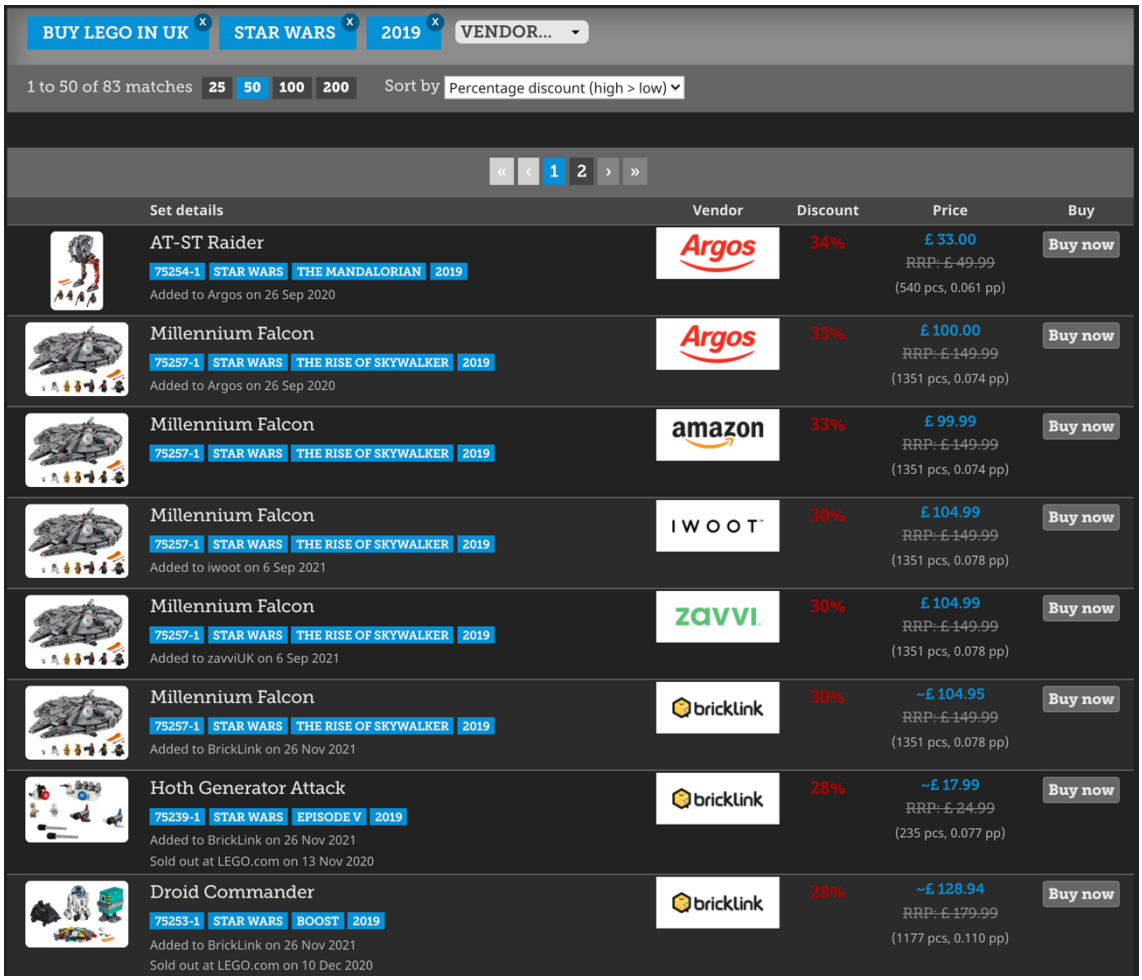

Figure 6 - Screenshot of a page from Brickset for LEGO Star Wars sets from the year 2019 (you cannot specify to view only a particular set) [25]

It allows users to apply filters or limit price tracking to the wanted list. However, its biggest drawback is that there is no data, and it is just price tracking. There is no graph showing how the price changes over time, no information about the lowest price or no information about the average price. This type of price tracker is only suitable for more experienced LEGO fans. Moreover, they display duplicates of the same set, making a list much longer and arduous to read for users with long wanted lists. Lastly, similarly to other price trackers, they do not apply Amazon vouchers from the listed price.

2.4.6 Others

There are some price tracking apps that focus exclusively on LEGO, such as BrickSpace, BrickEconomy and Brickwatch. BrickSpace targets the Netherland market, so it does not work well for the UK market. BrickEconomy focuses more on the investing side of LEGO, and it tracks prices mainly from the US and converts them to GBP, making it not practical. Brickwatch only tracks prices from the LEGO website.

2.4.7 Comparison

	CamelCamelCamel	PriceSpy	HotUKDeals	PriceRunner	Brickset
Automated	Yes	Yes	No	Yes	Yes
Correct item tracked	Yes	No	Yes	Yes	Yes
Applying Amazon vouchers	No	No	Yes	No	No
Average price	Yes	No	No	No	No
Price history graph	Yes	Yes	No	Yes	No
Price rating	No	No	No	No	No
Wanted list integration	No	Yes	No	No	Yes

Table 1 - Comparison of five competitor apps

Based on the market research, no one tracks prices with Amazon vouchers, which is a significant issue as Amazon vouchers could even introduce an additional 10% discount and hit record low prices. Moreover, many price trackers forget to provide a user with information such as minimum price or average price, which would help the user decide.

Some of the price history graphs are very good, such as the one provided by CamelCamelCamel; however, some graphs like the one from PriceSpy look confusing and not user-friendly. They are also often displayed at the bottom of the page rather than at the top when they give the user the most information about the price. None of the apps discussed in the previous section provides the user with a price rating. Lastly, very few apps provided the user with wanted list integration. Those that did have it have many areas to improve due to the app's other features not working properly or their lack.

Chapter 3: **Requirements Analysis**

3.1 Functional Requirements

1. Cloud MongoDB database – The system needs to have cloud database to store all the data.

2. Guest register, Guest login - If the Guest clicks the Login button and does not have an account, they can register.

3. Guest search for a specific set using its ID - The Guest can search for the LEGO set by clicking on the search bar and entering its set ID.

4. Display the average discounted price and graph showing the price history - The system will retrieve data collected from the database, calculate this value, and plot the graph.

5. Report bad tracking – The Guest can report if wrong data is or was tracked.

6. Rate the prices - The system will retrieve data collected from the database and display an appropriate rating based on the current price.

7. The system must allow Users to import their collection from Brickset as a CSV file - The User can upload a CSV file. The file will be parsed to retrieve needed data and stored in the database.

8. The system must provide the users with a recommendation of which sets to buy to qualify for the promotion - The list will be based on the lowest recorded price for each set. The algorithm will only consider sets from the members imported wanted list.

9. Recommendation System - The system will select a random set from users owned and wanted collection and recommend them other similar set.

10. The system should record the prices of all LEGO sets in production - The system should scrape all LEGO sets' prices and store them in the database and lowest price in separate collection.

3.2 Non-functional Requirements

1. The system should respond to the user in a reasonable amount of time - The database queries and scrape live data from the websites.

2. The system should store all the data in the MongoDB cloud database - The members' login details price statistics of all LEGO sets will be stored in the MongoDB database.

3. The system should have a user-friendly and intuitive user interface - The user should display enough information to help the user decide without overwhelming them with statistics. The graph should be easy to read with all critical dates and the lowest price marked.

Chapter 4: **Design**

4.1 Technologies Used

4.1.1 Node.js and Puppeteer

Puppeteer is a library that only supports Node.js; therefore, the scrapers will be developed in Node.js. Puppeteer is faster than other tools for automating web browsers, and speed is vital as prices will be scraped live. It also makes use of web browsers, making it harder to be detected as a bot and scrape data where content is loaded dynamically.

4.1.2 Python, Matplotlib and Tkinter

For the remaining components of the app, Python is used as it allows for relatively rapid development, easy debugging, and data analysis, which is the primary aspect of the project.

Matplotlib is a Python library allowing for data visualisation. It is used for showing the price history of the LEGO sets.

Tkinter is a standard Graphical User interface for Python. It allows for rapid development of the user interface. Initially, the project was supposed to be a web app; however, it became more algorithm heavy, leaving little time for developing the user interface. The user interface was crucial for the project to display collected and processed data more clearly.

4.1.3 MongoDB Cloud Database

A cloud database was needed that is compatible with both Node.js and Python. MongoDB meets those criteria. MongoDB consists of a database which consists of collections which consist of documents.

Data in MongoDB is stored as JSON files, making it convenient to access and manipulate. It is straightforward and quick to set up as it has a simple schema.

4.1.4 Amazon Web Services

Amazon Web Services (AWS) provides a free EC2 Linux instance that allows running the scraper all the time. Initially, the scraper was supposed to run on the ITL machine; however, it has an outdated version of Node.js, so the scraper would not work, and it cannot be updated due to a lack of permissions.

4.2 Algorithms and Metrics

4.2.1 Knapsack

LEGO Shopping assistant will make use of the adaptation of a knapsack algorithm to find the top 10 combinations of sets to buy to qualify for the promotion.

It will use a bottom-up approach rather than top-bottom with memoization. The top-bottom approach would consider all possible combinations which are not needed. Once the threshold is met, there is no point in considering more sets to add to the combination.

This is because, at best, the loss will not change; however, the total would increase, which does not make sense.

4.2.2 Euclidean Distance and Jaccard Index

Euclidean Distance and Jaccard Index are the two metrics used to calculate the similarity between two LEGO sets to build a content-based recommendation system. Euclidean distance is suitable for calculating similarity where the magnitude of the vectors is essential; however, it is not suitable for high dimensions. That is why it is going to be used for quantitative data. If it was used for tags, every tag would require a new dimension where 1 would represent that the given tag is present and 0 that it is not. That is why Jaccard Index will be used for qualitative data.

4.3 Use Case Diagram

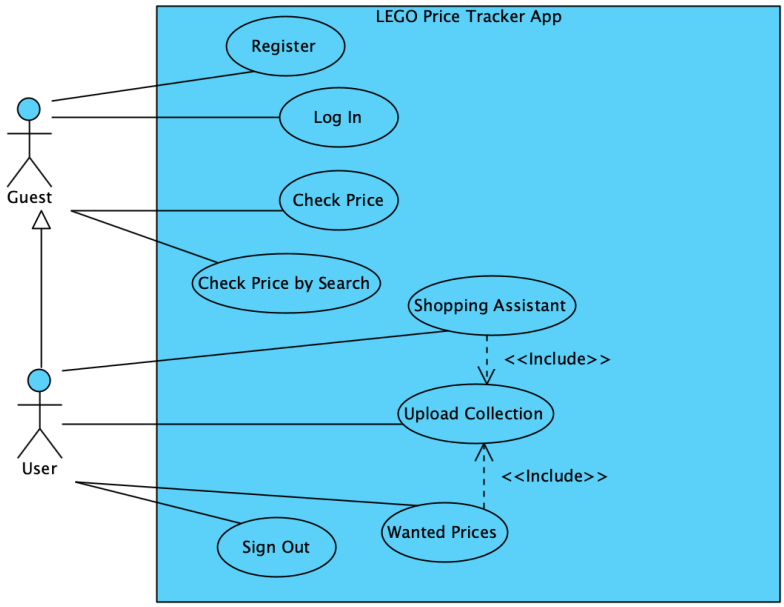

Figure 7 Use Case Diagram

The diagram above shows the high-level scope of the app and how different actors can interact with the system.

The two actors are the Guest and the User. A Guest can Register, Log In, Check Price and Check Price by Search.

The Guest becomes a User after registering or logging in. It can do everything that Guest can do, but additionally, it has access to more features. They can upload a collection, and after they upload their collection, they get Shopping Assistant and Wanted Prices.

4.4 System Architecture

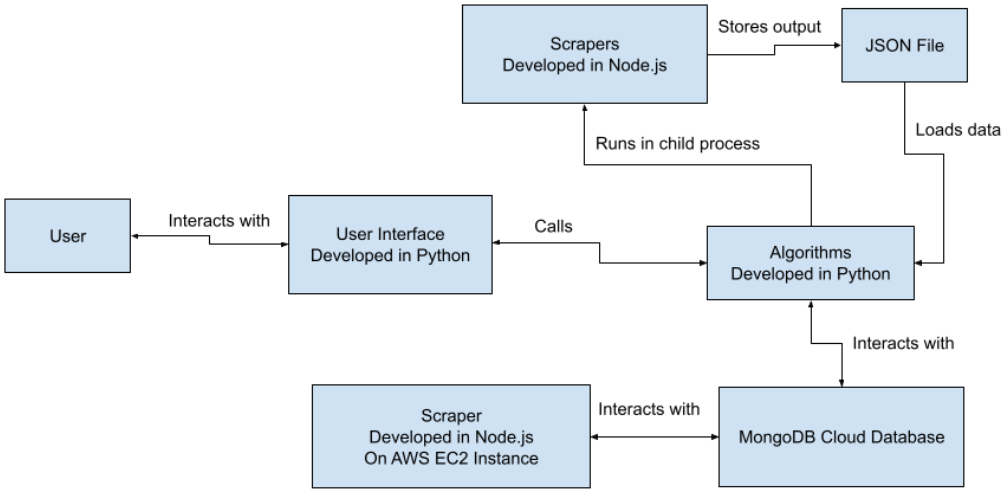

Figure 8 System Architecture

The figure above shows the system architecture.

The User interacts with the system through the user interface, and the user interface displays data to the user.

The user interface calls algorithms, and returned data is displayed in the user interface.

Some algorithms create a fork of the process, and the child process executes the scrapers. The data collected by the scraper is saved into a JSON file. The parent process waits until the child process finishes and loads the JSON file. Some algorithms interact with the MongoDB Cloud database.

One scraper is on an AWS EC2 instance that scrapes the prices of all LEGO sets and inserts the data into the MongoDB Cloud database.

4.5 User Interface

4.5.1 State Transition

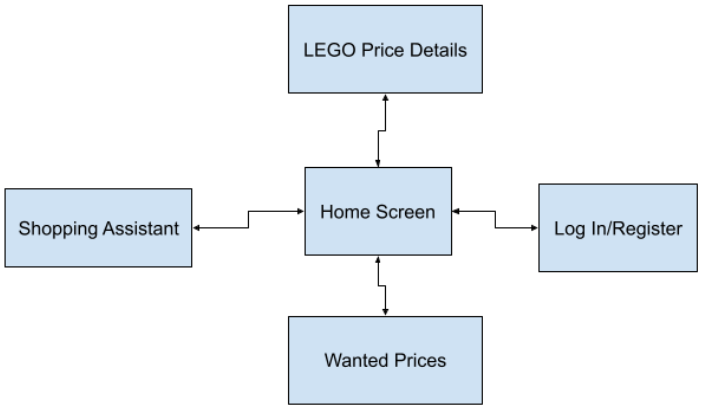

Figure 9 State Transition Diagram

The diagram above shows depicts navigation between different screens. Everything is accessed from Home Screen, and to access something different, the User needs to return to the Home Screen.

4.5.2 Login Screen

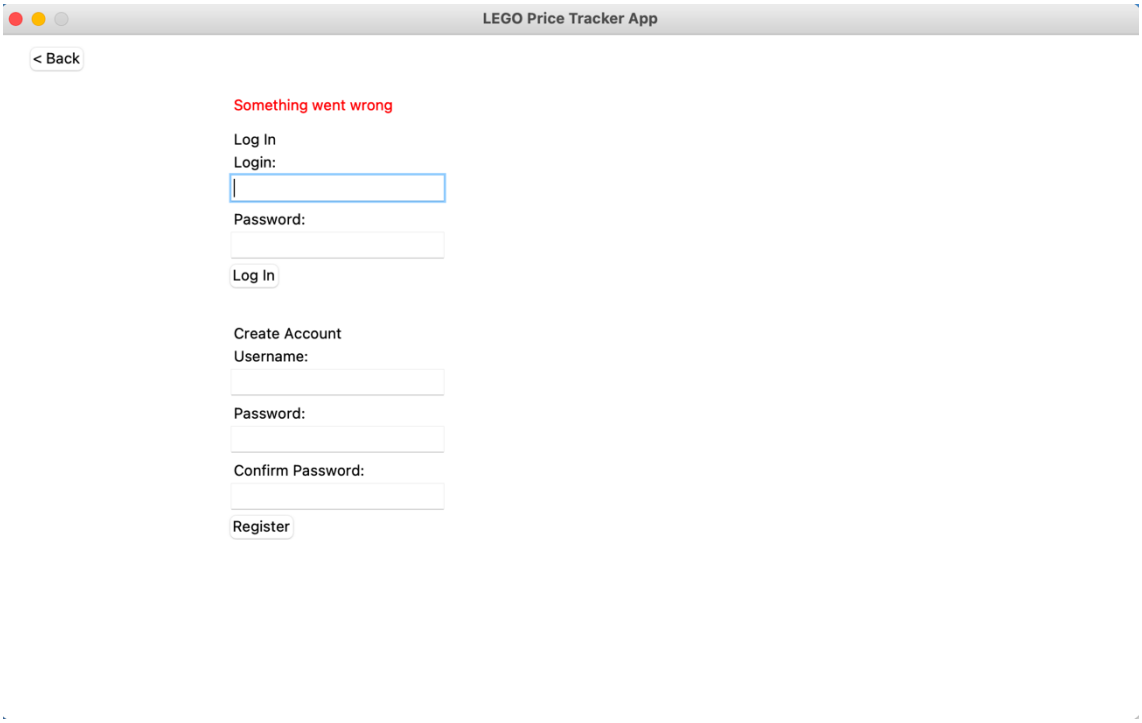

Figure 10 Log In/Register

The figure above depicts Log In/Register. The Guest can log in or register. If something goes wrong, the message in red is displayed at the top. The Guest can return to the home page by pressing the < Back button. After pressing Log In or Register button, if the action is performed successfully, the Guest will become a User and be redirected to the home screen.

4.5.3 Home Screen

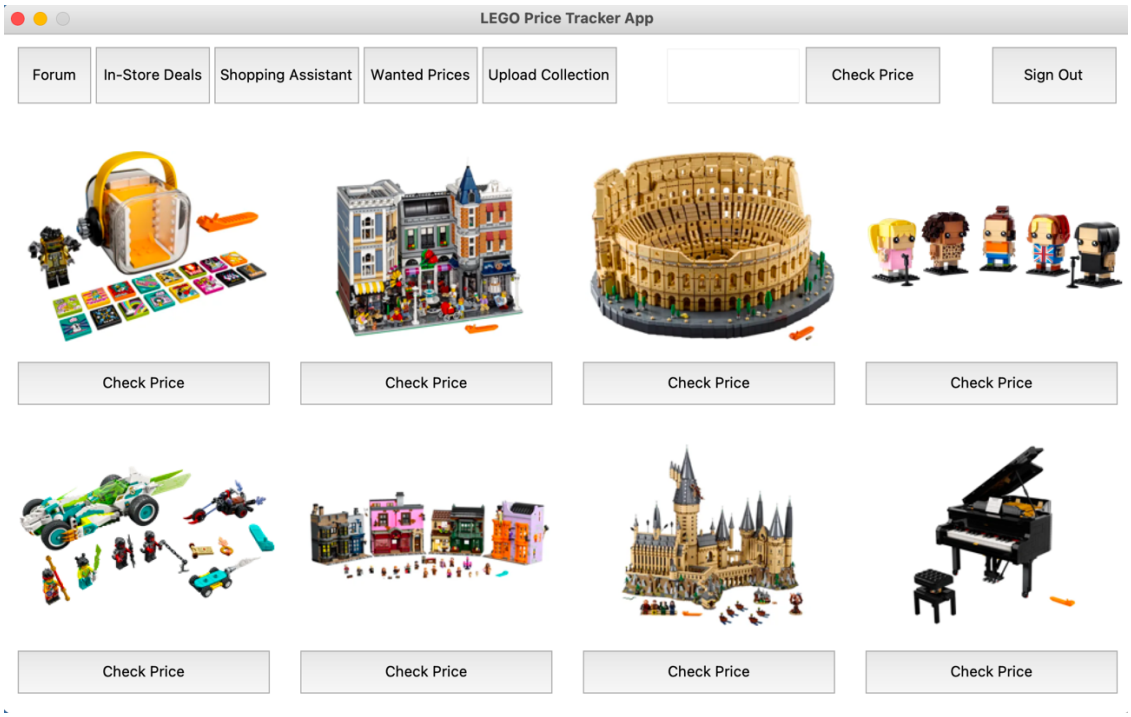

Figure 11 Home Screen

The figure above depicts Home Screen when the user is Logged In.

The eight pictures are recommendations based on users owned and wanted sets that were uploaded. If the user was not logged in the sets would be selected at random. By clicking Check Price button under the picture a screen would be loaded with price detailes of the corresponding set.

User can type set ID into the entry box and click Check Price button to see price of the specific LEGO set.

For Wanted Prices and Shopping Assistant button to be enabled the user needs to upload their wanted sets that can be exported as CSV file from Brickset.

When Sing Out button gets clicked the User is signed out and page is reloaded.

4.5.4 LEGO Price Details

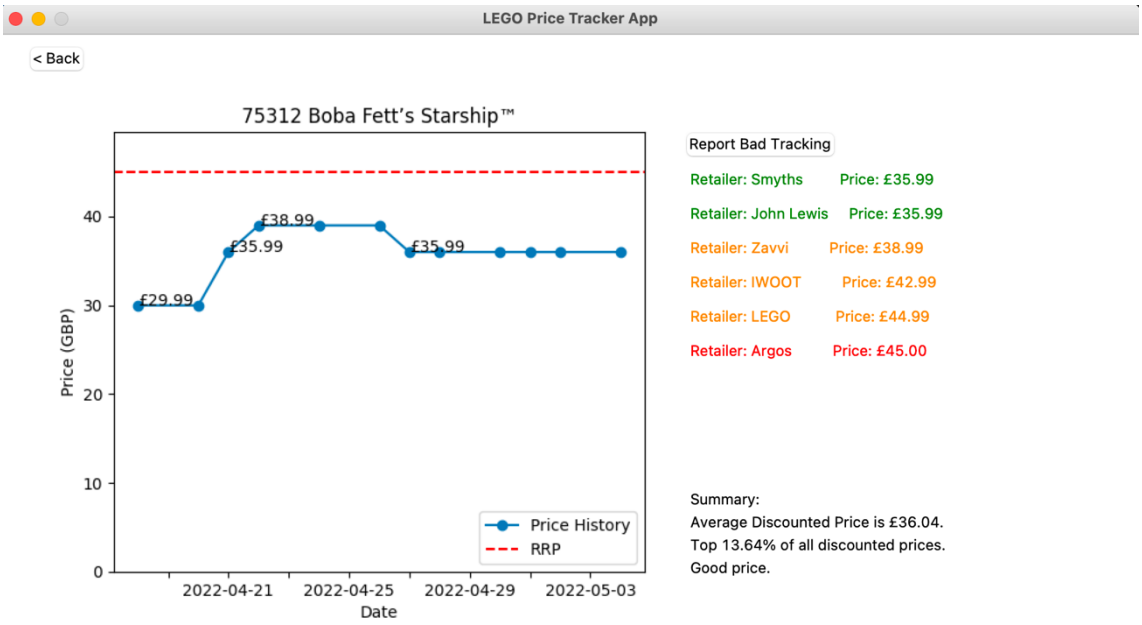

Figure 12 LEGO Price Details

The image above depicts LEGO Price Details. The graph shows the price history of the given LEGO sets labelling any price change. The blue line is the lowest discounted price at the given time, and the red dotted line is the retail price of the given set. All of the data is stored in the MongoDB database. A scraper collects it on an AWS EC2 instance.

To the right of the graph, there is a Report Bad Tracking. Suppose the user notices anomaly on the graph or some of the current prices are incorrect. In that case, the data will be inserted into a collection in MongoDB Cloud Database to be later investigated.

Below all live prices of the given set are displayed. They are sorted in ascending order and colour coded. Green is used for the best prices; red is used for the worst prices, and dark orange is anything in the middle. Each label is a hyperlink redirecting to the page where the set can be purchased.

In the bottom right corner, there is a summary of the data. It displays the lowest average discounted price; how does it compare to other discounts and rates the price.

The User can return to the home screen by pressing < Back button in the top left corner.

4.5.5 Wanted Prices

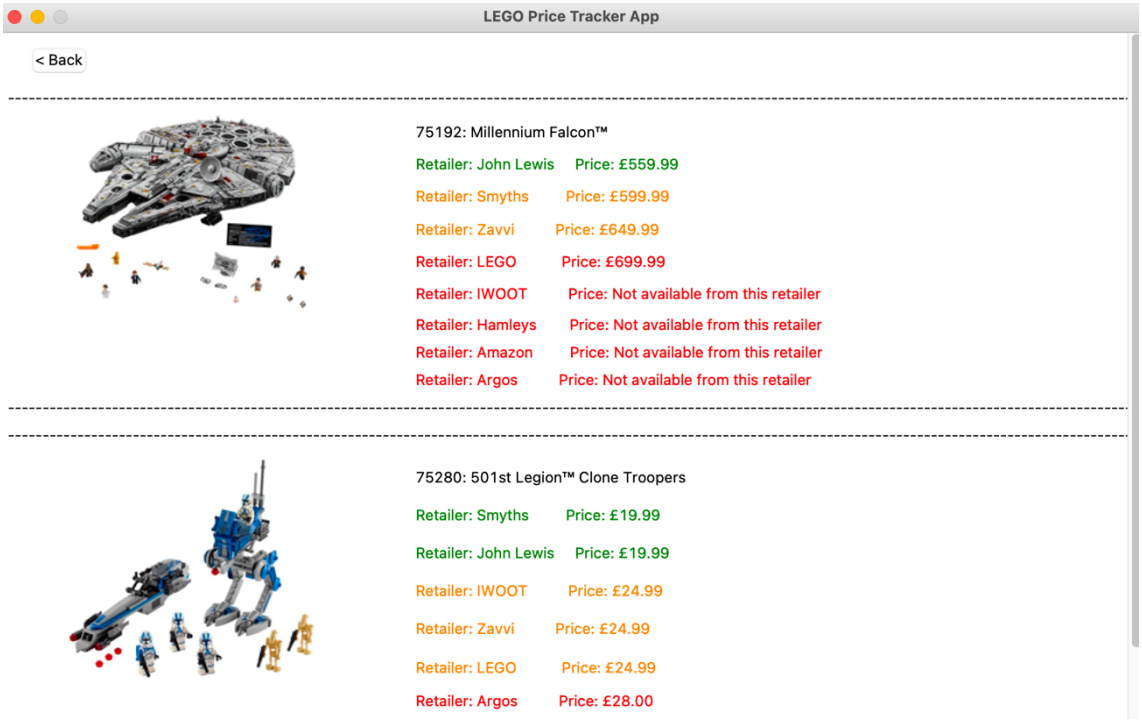

Figure 13 Wanted Prices

The image above depicts Wanted Prices. It scrapes the prices of all sets uploaded as a wanted list and displays the prices of all of them in one place. The scroll was implemented to allow for it. The colour coding works the same as for LEGO Price Details, and each price is also a hyperlink. The < Back button returns to the home screen.

4.5.6 Shopping Assistant

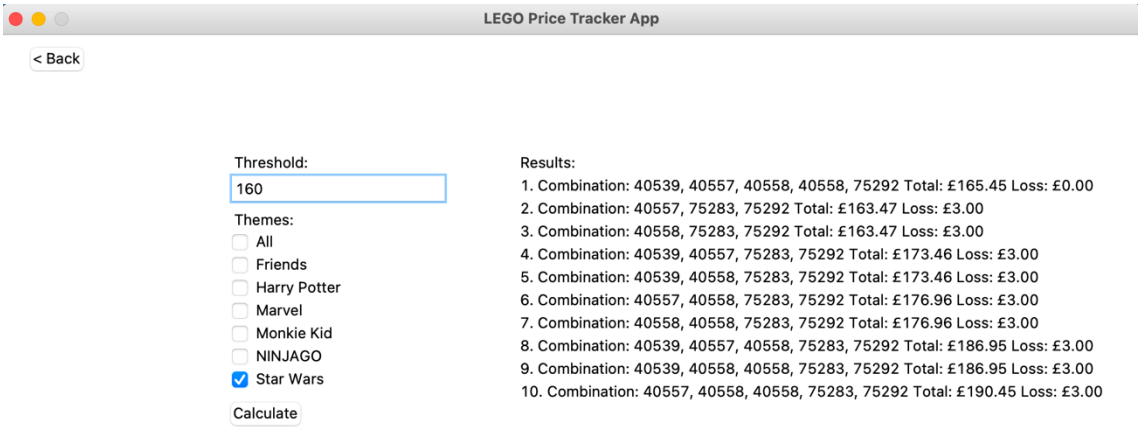

Figure 14 Shopping Assistant

The figure above shows Shopping Assistant. The algorithm finds up to 10 combinations of the sets to buy to qualify for the promotion (reach threshold) and minimise loss (amount overpaid by not buying the set on a discount).

The threshold in the input box needs to be specified. The themes are a filter if the promotion is theme specific. On the right, the results are displayed. They are sorted based on loss and then total. The < Back button returns to the home screen.

4.5.7 Forum and In-Store Deals

After further evaluation, Forum and In-Store Deals will not be implemented for now. This is because currently, there would be no moderation of the forum or in-store deals, which could cause ethical issues as there would be no control over what people post.

Chapter 5: **Implementation**

5.1 Scraping price of a single LEGO set

5.1.1 Choosing retailers

The app's main objective is to track the prices of LEGO sets. The retailers that data will be scraped from are retailers that receive LEGO exclusive sets. These are sets only sold directly from LEGO and that retailer, no one else. These retailers are Amazon, Argos, LEGO, Smyths and Zavvi. It was also decided to track prices from Hamleys and IWantOneOfThose even though they do not receive LEGO exclusive but often have significant discounts on LEGO products. Scraping data from all retailers that sell LEGO would not make sense as it would take longer to implement and overwhelm the user with the amount of data.

5.1.2 Algorithm

The automated algorithm opens eight browsers concurrently (one for each retailer). Then using the set ID passed as command line argument, it creates a link with a search query. After the page loads it then scrapes the data and closes the browser.

The automated algorithm is:

1. Launch the web browser
2. Go to the website
3. Scrape Data
4. Close the Browser

5.2 Getting data about LEGO sets

5.2.1 Where to get the data from?

We need a list of all LEGO sets to keep track of their price and have data to build recommendation system. There is no list of all the LEGO sets currently in production. Brickset lists all sets categorised by year, but LEGO sets do not have the same shelf-life. In most cases, it is two years, so I could compile a list of sets from the past two years; however, some popular sets like LEGO 75192 Millennium Falcon have been in production since October 2017, so this approach would not have the full coverage.

5.2.2 Algorithm

The optimal algorithm we constructed to compile the list is visiting the page with all the sets, then grabbing all the links and moving to the next page with all the sets. Then it visits each scraped link and scrapes all the needed data from the product page.

The collected data is saved into a text file.

The automated algorithm is:

1. Open the web browser
2. Go to the LEGO page with all sets
3. Scrape all the links
4. Grab the number of pages
5. Go to the next page and repeat step 3 till you visit all the pages

6. For each link, go to each page and scrape the data
7. Close web browser
8. Save the data into the text file

5.3 Tracking Price of All LEGO Sets

5.3.1 Why is it needed?

Part of the app's functionality is displaying data on previous prices of given LEGO sets. This data will also be used in the LEGO shopping assistant component and rating prices. To make these functionalities possible, there needs to be a scraper that runs all the time and collects that information.

5.3.2 Algorithm

This algorithm aims to scrape the prices of all LEGO sets, insert them into the collection in the MongoDB database, and insert the lowest recorded price for the given set into a separate collection. This is done to improve the algorithm's efficiency, discussed in section 5.5.

The algorithm is a hybrid of algorithms presented in sections 5.1 and 5.2. It uses scraped links to extract set IDs and compile a list of them; it then runs an algorithm from 5.1 on each set ID in the list.

It runs until interrupted; it updates the list of sets to scrape every time the inner loop is completed, meaning no human interaction is required.

The main difference between this algorithm and the one in 5.1 is that this one is scraping the data sequentially rather than concurrently. The reason for this will be discussed more in-depth in section 5.8.3. The automated algorithm is:

1. Open the web browser
2. Go to the LEGO page with all sets
3. Scrape all set IDs
4. Grab the number of pages
5. Go to the next page and repeat step 3 till you visit all the pages
6. Close the web browser
7. For each set, repeat steps 8-13
8. Open the web browser
9. Go to the website
10. Scrape data
11. Close the web browser
12. Insert data to the database with all the price history
13. Check if any of the prices are lower than previously recorded; if yes, update the entry in the database with all the lowest prices
14. Go back to step 1

5.4 Displaying Price of Wanted Sets

5.4.1 What is it useful for?

Users can upload their wanted list within the app. By doing so, they will be able to use this future to check live prices of all sets they want across multiple websites with one button click.

5.4.2 Algorithm

The algorithm is almost identical to the one from 5.1. Instead of taking a single set ID it takes a string of set ID separated by commas. It then converts the string into a list and calls scraping script on each set ID. The automated algorithm is:

1. For each set do steps 2-5
2. Launch the web browser
3. Go to the website
4. Scrape Data
5. Close the Browser

5.5 Recommendation System

5.5.1 Overview

The type of the recommendation system is content-based filtering. Developing it consisted of multiple stages:

1. Collecting data
2. Obtaining more data
3. Calculating similarity
4. Recommendation

Collecting data was already discussed in section 5.2 so this section will focus on the remaining 3 stages.

5.5.2 Obtaining more data

The algorithm from 5.2 obtains data about sets; however, it also scrapes links to images. These images can be fed into google Vision AI API, which generates tags based on an image. These tags are then added to the rest of the tags. Thanks to it, we get information about what the item is and how it looks.

Some of the generated tags could be applied to every set, for example, ‘Lego’, ‘Construction set’ and ‘Toy block’. They were not disregarded as they indicate that the given item looks more like a LEGO set than others.

The output of this algorithm is saved into a text file.

5.5.3 Calculating similarity

As discussed in section 4.2.2, two similarity metrics were used: Euclidean distance and Jaccard index.

The algorithm iterates through data generated by the algorithm in 5.4.2 and compares one set to every other set.

The algorithm puts quantitative data into a vector and qualitative data into a set. The two vectors are compared using Euclidean distance and the two sets using the Jaccard index. The data is stored in the dictionary in the form:

{set A ID: [(set B ID, Euclidean distance value, Jaccard index value), …], …}

After calculating the values, they are inserted into both rows for set A ID and set B ID to avoid repeating calculations.

The average of Euclidean distance value and Jaccard index value cannot be calculated yet. This is because the Euclidean distance value could be any natural number, whereas the Jaccard index is a value between 0 and 1; therefore, the Jaccard index, in most cases, would become insignificant. The Euclidean distance value needs to be normalised (changed to a value between 0 and 1).

The normalisation is done using the following formula:

$$normalised\ value = \frac{value - minimum\ value}{maximum\ value - minimum\ value}$$

The minimum and maximal values of Euclidean distance were recorded when the calculation was done initially.

For each value in the dictionary we stored, we normalise it. The normalised value in Euclidean distance will mean the opposite of what it means in the Jaccard index. Currently, the 1 in Euclidean distance means it is the greatest distance, and 0 means the distance is minimal; the Jaccard index value is the other way around. Therefore, we also need to subtract the normalised value from 1.

Now we can calculate the mean value of the two similarity values, and if it is greater than 0.75, insert it into the new dictionary. The value 0.75 was selected as no row is left empty with this value. The new dictionary structure is:

{set A ID: total similarity, [(set B ID, similarity), …], …}

The total similarity is the sum of all similarity values in the given row. The value is incremented whenever the mean value is calculated, and the mean value is inserted into both rows to avoid repeated calculations.

After the row in a dictionary is complete, the list of tuples (set ID, similarity) is sorted based on similarity value in descending order.

The dictionary is saved into a JSON file.

The algorithm in steps is:

1. Calculate Euclidean distance and Jaccard index
2. Normalise Euclidean distance and subtract it from 1
3. Calculate the mean of the two values
4. Save the dictionary into a JSON file

5.5.4 Recommendation

The most accurate recommendation would be selecting a random set from users' owned or wanted list and recommending the first item in the dictionary. The problem with this approach is that the recommendation would often be repeated, making it quite dull for the user to keep being fed the same content.

Instead, any set with at least a 0.75 similarity to a given set can be recommended. Sets that have higher similarity are more likely to be recommended. By doing so, the recommendations will still be relevant and more diversified.

This is executed by selecting a random set from users' owned or wanted collection. That set ID of the selected set is used as a key to access the row in the dictionary (output of 5.4.3). Then another random number is generated between 0 and the total similarity of that row. We then move from left to right along with the list of tuples (set ID, similarity).

Before moving to the next element to the right, we subtract the similarity from the randomly generated value each time. Once that value becomes 0 or less, we stop, and that set gets added to the set of sets to recommend. This gets repeated until the set has a size of n. The algorithm in steps is:

1. Select a random set from users' owned or wanted list
2. Use it as a key and generate a random number between 0 and total similarity
3. Move from left to right of the list, each time subtracting the current element's similarity. If the number becomes 0 or less, go to the next
4. Add the set to the set of sets to recommend
5. Repeat all steps until the set has size n

The above algorithm can only be executed if the user is logged in and uploaded at least their owned or wanted list. If they did not, then a completely random set is recommended.

5.6 LEGO Shopping Assistant Component

5.6.1 What is it doing and why is it needed?

The assistant will retrieve the price history of all the sets from the database and suggest the optimal combination of sets to buy to qualify for the promotion.

LEGO does not offer discounts on its website too often. If they do, it is typically on items that are selling exceptionally poorly, yet that price most of the time is still higher than from other retailers. Typically, when people buy from LEGO, they pay the full retail price (they overpay). To attract customers, apart from offering 5% cashback through their loyalty program, they offer a gift with purchase (GWP). To receive it, a customer needs to spend on qualifying items over a specified threshold, for example, £150 on any LEGO sets or £75 on Star Wars sets. Most of the time, GWP can be only obtained by participating in the promotion, and they have a limited run. Once the offer is over or they sell out, they might not be available ever again.

5.6.2 How does it work?

We have the data about the lowest price for each set. The algorithm will group the sets to meet the promotion criteria and minimise the money the person is overpaying.

This is a version of a knapsack problem as it is also a combination problem that aims to find an optimal combination given certain restrictions; therefore, the problem can be solved with a similar approach.

In the knapsack problem, we aim to maximise the value of items under specific weight restrictions. In our problem, we aim to minimise the overpaying amount and make the price of items add up above a certain amount.

5.6.3 Algorithm

The algorithm used to solve this problem is an adaption of a knapsack problem. It is an NP-hard problem, and it uses using the bottom-up approach.

It starts by separating sets into two lists, sets that do not require matching because they qualify for the promotion already and sets that require to be matched with other sets.

It considers all possible combinations of sets that require matching; however, the recursive branch gets cut out once the solution is found as adding more sets to the given

combination would not make sense. This is because adding more sets would increase the total spend and, at best, will keep the loss the same.

After finding a valid combination, it adds the given combination to the set once it reaches a specified threshold.

Lastly, the list is sorted in ascending order based on loss value and total value, and the first ten entries get returned.

5.7 Rating Prices

5.7.1 Algorithm

The algorithm works very similarly to the insertion sort algorithm's insertion part, except it does not insert it. It only determines the index where it should be inserted. The algorithm discussed in this section has linear complexity.

The algorithm loads from the database into the array all the prices of the given set, disregarding any discounts less than 5% of the retail price. Then the array is sorted in descending order. After that, we move along the array until we reach a point where the next value is less than the current value. We then compare the index at which we stopped with the array's length to determine the percentile in which the current price lies. Then appropriate String is returned. The algorithm in steps is:

1. Load prices disregarding any discount lower than 5%
2. Sort the array
3. Determine index where the current price should be inserted
4. Determine percentile
5. Return appropriate message

5.7.2 Why disregarding any discount lower that 5%?

The goal is to show how good the given discount is compared to previous discounts. Some retailers price their sets a few pence under the retail price; therefore, they would corrupt the data. 5% is also the percentage of cashback the person gets by buying directly from LEGO at any point, assuming the given set is in stock. Therefore, any set can be purchased at a 5% 'discount' directly from LEGO. It is not an actual discount, but the app's target audience will spend it at some point; therefore, it does not make a significant difference for them.

5.8 Troubleshooting

5.8.1 Scraping price of a single LEGO set

Initially, the program would call the function to scrape from one website and wait before moving on to the next website. This would take 40 seconds which is too long. Using JavaScript Promise.all feature allowed us to lower it to 8 seconds. It would call all the functions and then wait for them to finish. Additionally, any request for CSS or Image gets interrupted, which reduces time even further.

Another problem was scraping data from Amazon. When searching for LEGO using its set ID on all the other websites, it would be the first item that appeared; however, Amazon runs ads. The first few items would be adverts, and the number of ads would vary. The workaround was to grab all the items listed on the page and perform a linear search to

find which items contained a matching set ID in their description. Unfortunately, this raised another problem. Amazon also sells LED kits, stands, display cases, and fake copies of LEGO sets that contain the set ID in their description; therefore, it was needed to compile a list of keywords to avoid. They needed to be chosen accurately to avoid disregarding real LEGO sets.

The biggest problem was that Smyths detected the scrapper as a bot and did not allow it to scrape from their website. The workaround was to make a google search, grab all search results, and perform the linear search looking for the Smyths website. The price, which is the only data we are interested in, is visible in that search without accessing Smyths' website.

5.8.2 Getting data about LEGO sets

LEGO website is incredibly inconsistent when presenting the data of the sets, which made scraping significantly more challenging as it required a more complex scraper that would be able to extract the correct information regardless of the page layout. Additionally, it made debugging much more challenging as it was unclear what and why it failed.

The prime example of that was extracting data about sets' dimensions. For some cases, the data was extracted successfully, and for some, it failed. Most of the sets had this piece of data within the "Specification" section. This section was at times loaded automatically, and at times it required press of the "Specification" button to load it, which was not clear.

This inconsistency made it very hard to deduct the source of error as the scraper was doing the correct jQuery, and the page's content was loading. The actual issue was that the scraper was not actually pressing the button but only navigating to it. So, when the scraper was successful, it was because the "Specification" section was already loaded.

5.8.3 Tracking prices of all LEGO sets

Initially, the script was supposed to be running on the ITL machine, and it could scrape data of any set in about 8 seconds. The node version on that machine is outdated and does not support puppeteer.

Instead, a free version of the AWS EC2 instance was used, which is not too powerful. As a result, the script needed to be adjusted, and now it takes about 30 seconds to scrape data about any LEGO set.

AWS has a credit system for the burstable performance of the instance. After about 20 minutes, when the credits run out, the CPU utilisation drops to about 10% and time increases to 3 minutes per set. This is associated with the way AWS EC2 works, not the optimisation of the algorithm.

5.8.4 Recommendation System

The hard part about building the recommendation system was obtaining data, which troubleshooting was already described in 5.8.2.

Google Vision AI API caused some minor trouble because, at times, it was returning no tags; however, increasing the time interval between the calls seemed to increase the success rate. As a result, the script needed to be run four times on rows that had no tags generated (rows with no tags had a length of 15).

The whole process of collecting, generating tags and calculating similarity could have been fully automated, only requiring one button press. Unfortunately, each script needs

to be run separately. This is because the API we are using has a limit of 1000 free calls per month. To not surpass the free limit, the process requires monitoring if something goes wrong. There were slightly over 700 sets, and some required to be retagged again as described above. Even if the script were fully automated and would not go over the limit, it would not be adequately tested as it can be run at most once per month.

Chapter 6: **Evaluation**

6.1 Graphical User Interface Testing

Graphical User Interface Testing was performed manually interacting with the user interface and observing its behaviour. The results are recorded in the table below:

Test	Expected Behaviour	Passed?
Launching app	Home screen loaded as Guest	YES
Clicking “Check Price” button with empty entry box	User stays on the home screen	YES
Clicking “Check Price” button with invalid set ID in the entry box	User stays on the home screen	YES
Clicking “Check Price” button with “10274” in the entry box	Screen with data about the “10274” is displayed	YES
Clicking “Report Bad Tracking” button	Message “Report Sent Successfully” appears on the screen and button disappears.	YES
Clicking all hyperlinks on the screen displaying data	Correct page opened in the browser	YES
Clicking “< Back” button on the screen with displayed data	Home screen loaded	YES
Clicking “Check Price” in each column and row	Screen with data about the correct set is displayed	YES
Clicking “Log In/Register” button	Log In and Register screen loaded	YES
Clicking “< Back” button in the Log In and Register screen	Main screen loaded	YES
Clicking “Log In” button with empty “Login” and “Password” entry fields	Message “Something went wrong“ displayed	YES
Clicking “Log In” button with invalid “Login” and valid “Password”	Message “Something went wrong“ displayed	YES

Clicking “Log In” button with valid “Login” and invalid “Password”	Message “Something went wrong“ displayed	YES
Clicking “Log In” button with valid “Login” and valid “Password”	Home screen loaded as User	YES
Clicking “Register” button with empty fields	Message “Something went wrong“ displayed	YES
Clicking “Register” button with existing “Username”	Message “Something went wrong“ displayed	YES
Clicking “Register” with unmatched “Password” and “Confirm Password”	Message “Something went wrong“ displayed	YES
Clicking “Register” with valid credentials	Home screen loaded as User	YES
Clicking “Upload Collection”	Pop-up window allowing to upload CSV file format	YES
Clicking “Wanted Prices”	Screen with pricing of all wanted sets displayed	YES
Clicking all hyperlinks on the screen displaying pricing	Correct page opened in the browser	YES
Clicking “< Back” button on the screen displaying pricing	Home screen loaded	YES
Clicking “Shopping Assistant” button	Shopping Assistant screen loaded	YES
Clicking “Calculate” button with empty “Threshold”	No results displayed	YES
Clicking “Calculate” button with “Threshold” specified and “All” selected for “Themes”	Results displayed	YES
Clicking “Calculate” button with too high “Threshold” to meet	No results displayed	YES
Clicking “Calculate” button with various selection for “Themes”	Sets from not selected “Themes” are disregarded	YES

Clicking “Back” button in Shopping Assistant screen	Home screen displayed	YES
Clicking “Sign Out” button	Home screen displayed as Guest	YES

Table 2 Graphical User Interface Testing Results

6.2 Unit Testing

Pythons’ module unittest was used to perform Unit Testing on Shopping Assistant and Rating Prices algorithms.

6.2.1 Shopping Assistant

Test	Expected output	Passed?
Empty list	Empty list	YES
Sets cannot meet the threshold	Empty list	YES
Multiple combinations have the same loss	List ordered to display lowest total first	YES
Quantity of each set is 1	List is outputted sorted based on loss and total ascending	YES
Quantity of some sets is greater than 1	Combination with the same sets is considered. No duplicate combinations.	YES

Table 3 Shopping Assistant Unit Testing Results

6.2.2 Rating Prices

Test	Expected output	Passed?
179.99, []	“WAIT FOR IT TO GO ON SALE!!!”	YES
49.99, [49.99]	“Best price!”	YES
22.39, [19.99]+[22.40]*10	“Very good price!”	YES
35.27, [22.99]*3+[38.13]*15	“Good price.”	YES
99.99, [74.45]*2+[110.13]*7	“OK price…”	YES
57.67, [52.55]	“You can do better…”	YES

Table 4 Rating Prices Unit Testing Results

6.3 Testing Recommendation System

To evaluate the Recommendation System, pairs of similar sets were selected manually. The pairs selected were sets of various sizes and themes.

The set ID of one of them was used as a key to look up the row in the similarity dictionary and investigate whether the other set ID was present. If it is, it means the given set can be recommended and thus, the recommendation system is accurate. The results are represented in the table below.

Pair	Present?
75192, 75252	YES
10272, 10284	YES
10255, 21330	YES
42131, 42114	YES
43197, 71040	YES
10295, 10265	YES
10280, 10289	YES
40377, 40476	YES
76204, 76203	YES
76908, 76907	YES

6.4 Testing Price Scrapers

Each of the three scrapers that scrape prices execute the same jQueries therefore, they will have the same accuracy. However, the evaluation is performed based on the data collected by the scraper on the AWS EC2 instance.

The accuracy of the scraper is significantly better than those currently on the market, and it successfully applies amazon vouchers which none of the other ones does. The accuracy is better, but it is still not 100%. The primary source of errors is amazon due to misleading descriptions of items by private sellers who sell other construction toys. It successfully filters out many of them by checking it against a list of banned words; however, adding every word to the list needs to be carefully evaluated, not unintentionally, to ban some actual LEGO set.

A button “Report Bad Tracking” was added to the screen displaying data about a particular set to combat it. Currently, there is no filter for how many reports a given user

can send or to remove duplicates from the database if two different users send the same report. However, this can be easily implemented if the number of reports is overwhelming (which should not happen, the data collected is very accurate).

6.5 User Acceptance Testing

The application was showcased to a few peers. No data was recorded.

6.5.1 What users liked?

The general feedback was that people liked the application. They found it easy and quick to use. All the essential functionality that they needed was there. When data was displayed, it was easy to understand, and the amount of information was not overwhelming.

They particularly liked the Shopping Assistant as they had never seen a feature like that before. It replicates the decision-making process they would have to go through when planning their future LEGO purchase. It does what typically takes them 15-30 minutes in few seconds.

6.5.2 What needs to be improved?

The main issue pointed out was the user interface not communicating clearly enough when the user needs to wait. The cursor icon turns into a loading circle by default; however, it might appear that the app froze rather than some algorithm running in the background.

Apart from that, some minor cosmetic issues like displaying retailers' names and prices were not all aligned. The length of each is the same; it is to do with characters having different widths. The text would need to be split into two separate labels.

Furthermore, each label is a hyperlink to the retailers' website to buy the set. This was not clear. It is aesthetically pleasing to have the link hidden; however, users were not aware of it.

6.6 Summary

The graphical user interface, algorithms and recommendation system perform as expected. Scrapers are very accurate; however, they do not have 100% accuracy; this will be improved over time thanks to users' "bad price tracking" reports. The user interface is very intuitive and quick to use. There are some cosmetic issues in terms of the user interface; however, they are not a high priority as they do not impact the usability of the app.

Chapter 7: **Conclusion**

7.1 Things I learned

7.1.1 JavaScript

My JavaScript knowledge before starting the project was fundamental. It was limited to making simple JavaScript queries to update some CSS attributes of some elements. Now I feel more confident in using the language.

7.1.2 Web Scraping

Prior to starting the project, I had no web scraping experience. I was learning it as I went. The hardest part was to determine what is possible with web scraping and learn techniques to scrape harder to find data, avoid bot detection and find ways to make the algorithm faster.

7.1.3 Algorithms

The project required me to develop way more algorithms than initially planned. In total, I developed ten algorithms ranging from determining the percentile in which the given number lies to creating a LEGO recommendation system. Each algorithm had multiple versions to make it more efficient and more elegant.

7.1.4 Recommendation Systems

This task required me to do much research to understand the types of recommendations system and assess which one is feasible and most suitable for my project. I learned that there are multiple techniques to calculate the similarity between two given items. The algorithms also vary from very primitive ones to highly advanced, for which top companies like Netflix are willing to pay millions [20]. Now I have a much better understanding of the topic. It was surprising how complex recommendation systems can get considering it is a present feature in most of the current apps. It was fascinating to see how different LEGO sets are similar to each other using pure data.

7.1.5 Things that helped

I am a LEGO collector in my spare time, and I have been heavily invested in the hobby for the past five years. Being fully immersed in the hobby helped me understand the needs of a LEGO fan. I have been analysing LEGO prices for years, predicting discounts, manually calculating the best sets to buy to qualify for the promotion, and observing what other LEGO fans think by following dedicated LEGO pages, deal websites, and Facebook groups. All of that knowledge was now put to use. It helped me identify the struggles of LEGO fanatics and come up with tools to fix them.

Despite being active in the LEGO community, I still had to do market research to find unpopular apps that track and identify why they might have failed.

7.2 Future Work

7.2.1 Most Important Changes

Due to time limitations and my project becoming more algorithm heavy, I could not make a web app and settled for a desktop app. Making a web app would make it more accessible to a broader group of people and track the prices of LEGO sets on the go.

The most urgent change is updating links to the referral links to make a commission by redirecting people to buy from the retailer's website. Doing so would allow me to get funds for scaling out, for example, having a more extensive database and advertisement to attract more users.

Additionally, the all of the users' owned and wanted sets would be stored in the database rather than only the sets currently in production. For hosting a more extensive cloud database, funding is crucial. Having users' collection stored in the database would allow me to implement a more sophisticated hybrid recommendation system. Making a better recommendation is crucial; better recommendations increase the likelihood of a sale through the app and thus project generating profit.

7.2.2 Scaling out

If the app proves successful in the UK market, it can be expanded to other markets. Particularly the USA, Germany, and China, which are giant markets for LEGO. Although it will require adjustments as different countries have different price trends, retailers and LEGO availability, the functionality will be the same; however, further research would be required to understand these markets better.

If the LEGO Price Tracker App succeeds, the same model can be adapted for other brands, which gives much room for scalability.

References

[1] G. Erhverv, 04 September 2014. [Online]. Available: https://finans.dk/live/erhverv/ECE6996783/Lego-er-nu-verdens-største/?ctxref=ext. [Accessed 28 November 2021].

[2] N. Sawaya, n.d.. [Online]. Available: https://www.fi.edu/sites/default/files/PressKit_ArtOfTheBrick_Quirky_LEGO_Facts.pdf. [Accessed 28 November 2021].

[3] J. Partridge, 10 03 2021. [Online]. Available: https://www.theguardian.com/lifeandstyle/2021/mar/10/lego-sales-soar-on-back-of-covid-lockdowns-and-nintendo-tie-up. [Accessed 28 November 2021].

[4] H. Millington, 13 05 2020a. [Online]. Available: https://brickset.com/article/51364/lego-ditches-creator-expert-branding-in-favour-of-18-marking. [Accessed 28 November 2021].

[5] The LEGO Group, 26 November 2019. [Online]. Available: https://www.lego.com/en-us/aboutus/news/2019/november/lego-bricklink/. [Accessed 28 November 2021].

[6] V. Dobrynskaya and J. Kishilova, “ScienceDirect,” 20 September 2021.

[7] H. Kaur, “GeeksForGeeks,” 09 November 2021. [Online]. Available: https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/. [Accessed 23 November 2021].

[8] GeeksForGeeks, “GeeksForGeeks,” 02 July 2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-puppeteer-and-selenium/. [Accessed 21 March 2022].

[9] WeblinkIndia, “WeblinkIndia,” 27 June 2017. [Online]. Available: https://www.weblinkindia.net/blog/static-vs-dynamic-website-advantages-disadvantages. [Accessed 21 March 2022].

[10] C. Wharfe, 2 November 2020. [Online]. Available: https://www.brickfanatics.com/whats-changed-about-lego-ideas-92177-ship-in-a-bottle/. [Accessed 28 November 2021].

[11] H. Millington, 31 October 2020b. [Online]. Available: https://brickset.com/article/54434/the-history-of-5-digit-set-numbers. [Accessed 23 November 2021].

[12] Google, “Google,” 2022. [Online]. Available: https://developers.google.com/machine-learning/recommendation/content-based/basics. [Accessed 4 May 2022].

[13] A. Ajitsaria, “Real Python,” 2022. [Online]. Available: https://realpython.com/build-recommendation-engine-collaborative-filtering/. [Accessed 4 May 2022].

[14] J. Chiang, “medium,” 26 June 2021. [Online]. Available: https://medium.com/analytics-vidhya/7-types-of-hybrid-recommendation-system-3e4f78266ad8. [Accessed 04 May 2022].

[15] Google, “Google Cloud,” 2022. [Online]. Available: https://cloud.google.com/vision. [Accessed 04 May 2022].

[16] S. Prabhakaran, “machinelearningplus,” 22 October 2018. [Online]. Available: https://www.machinelearningplus.com/nlp/cosine-similarity/. [Accessed 04 May 2022].

[17] Zach, 23 December 2020. [Online]. Available: https://www.statology.org/jaccard-similarity/. [Accessed 04 May 2022].

[18] M. Lüthe, “towardsdatascience,” 17 November 2019. [Online]. Available: https://towardsdatascience.com/calculate-similarity-the-most-relevant-metrics-in-a-nutshell-9a43564f533e. [Accessed 04 May 2022].

[19] C. Wharfe, “brickfanatics.com,” 22 September 2021. [Online]. Available: https://www.brickfanatics.com/lego-responds-to-rumours-of-price-rises-2022/. [Accessed 28 November 2021].

[20] K. Hafner, “The New York Times,” 02 October 2006. [Online]. Available: https://www.nytimes.com/2006/10/02/technology/02netflix.html. [Accessed 04 April 2022].

[21] The LEGO Group, “lego.com,” 2020. [Online]. Available: https://www.lego.com/en-gb/product/lego-nasa-apollo-saturn-v-92176. [Accessed 28 November 2021].

[22] CamelCamelCamel, “camelcamelcamel.com,” 2021. [Online]. Available: https://uk.camelcamelcamel.com/product/B07NDB2SFH?context=search. [Accessed 28 November 2021].

[23] PriceSpy, “pricespy.co.uk,” 2021. [Online]. Available: https://pricespy.co.uk/kids-family/toys/lego/lego-star-wars-75255-yoda--p5200290. [Accessed 28 November 2021].

[24] PriceRunner, “pricerunner.com,” 2021. [Online]. Available: https://www.pricerunner.com/pl/72-5058994/Toys/Lego-Star-Wars-Yoda-75255-Compare-Prices. [Accessed 28 November 2021].

[25] Brickset, “brickset.com,” 2021. [Online]. Available: https://brickset.com/buy/country-uk/order-percentdiscount/theme-Star-Wars/year-2019. [Accessed 28 November 2021].

[26] Meta, “facebook.com,” 2022. [Online]. Available: https://www.facebook.com/business/help/164749007013531?id=401668390442328. [Accessed 23 March 2022].

[27] Google, “Google Cloud,” 2022. [Online]. Available: https://cloud.google.com/vision/?utm_source=google&utm_medium=cpc&utm_campaign=emea-gb-all-en-dr-bkws-all-all-trial-e-gcp-1011340&utm_content=text-ad-LE-any-DEV_c-CRE_574627776421-ADGP_Hybrid%20%7C%20BKWS%20-%20EXA%20%7C%20Txt%20~%20AI%20%26%20ML%20~%20Vi. [Accessed 23 March 2022].

[28] upwork, “upwork,” 06 April 2021. [Online]. Available: https://www.upwork.com/resources/what-is-content-based-filtering. [Accessed 04 April 2022].

[29] L. Zahrotun, February 2016. [Online]. Available: https://core.ac.uk/download/pdf/86430721.pdf. [Accessed 04 April 2022].

[30] S. Prabhakaran, “machine learning +,” 22 October 2018. [Online]. Available: https://www.machinelearningplus.com/nlp/cosine-similarity/. [Accessed 04 April 2022].

[31] DeepAI, “DeepAI,” 2022. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/jaccard-index. [Accessed 04 April 2022].

[32] HotUKDeals, “hotukdeals.com,” 2021a. [Online]. Available: https://www.hotukdeals.com/search?q=lego%2Bstranger%2Bthings. [Accessed 28 November 2021].

[33] HotUKDeals, “hotukdeals.com,” 2021b. [Online]. Available: https://www.hotukdeals.com/search?q=lego%2Becto%2B1. [Accessed 28 November 2021].

[34] HotUKDeals, “hotukdeals.com,” 2021c. [Online]. Available: https://www.hotukdeals.com/search?q=lego%2075255. [Accessed 28 November 2021].

Appendix A – Risk Assessment

Risk	Likelihood	Severity	Impact	Prevention
Retailers stop allowing bots on the web pages	Medium	Medium	Possible delays in the development of other features until a solution is found	Make another plan in advance
Shortage of LEGO sets	Low	High	Very few LEGO sets to track prices of and retail prices of available LEGO sets. Will not be able to collect diversified data	Start tracking prices as soon as possible.
Learning new skills	High	Medium	Possible delays in the development of other features	Look into skills needed before the planned data of starting development
Poor time management	Low	High	Rushed project, not properly thought out and tested	Drop other not necessary duties when sensing too much workload to handle
Losing access to Brickset API key	Low	Medium	No convenient way to import the sets. Lack of sets classification to themes.	Ensure not to abuse Bricksets' terms and conditions
Cannot debug the code	Medium	Medium	No progress on the project	Produce a clear code with split into methods and commented
Feature too hard to implement	Medium	Medium	No progress on the project	Try implementing other features in the meantime and discuss alternatives with the supervisor
Losing work	Low	High	Possible delays in the development of other features	Make a backup of every piece of work
Flawed Design	High	Medium	Reimplementing feature again, causing delays	Do in-depth research before implementing

Appendix B – Project Plan

Milestone	Date
Progress Slides	29 Nov - 5 Dec
Presentation	6 Dec - 12 Dec
Regularly Tracking Prices	13 Dec - 19 Dec
Generate Graphs	20 Dec - 26 Dec
Import Owned and Wanted List from Brickset	27 Dec - 2 Jan
LEGO Shopping Assistant	3 Jan - 16 Jan
Primitive Recommendation System	17 Jan - 23 Jan
Predicting discounts on the new sets	24 Jan - 30 Jan
User Interface	31 Jan - 20 Feb
Testing	21 Feb - 6 Mar
Draft Report	7 Mar - 20 Mar
Forum	21 Mar - 27 Mar
Posting Deals	28 Mar - 3 Apr
Testing	4 Apr - 10 Apr
Polishing Final Report	10 Apr - 24 Apr
Project Video	25 Apr - 6 May
Final Report Submission	4 May
Viva	9 May - 20 May