Python webscraper bot

Oscar Lihou-Smith - AH Computing Project

SCN 230815933

Contents

[Analysis 4](#_Toc160804776)

[Description of the problem 4](#_Toc160804777)

[Purpose 4](#_Toc160804778)

[Advanced Higher concepts 4](#_Toc160804779)

[Integration 5](#_Toc160804780)

[Constraints 5](#_Toc160804781)

[UML Use Case Diagram 6](#_Toc160804782)

[Requirements Specification 7](#_Toc160804783)

[End-User Requirements 7](#_Toc160804784)

[Functional Requirements 10](#_Toc160804785)

[Project Plan 11](#_Toc160804786)

[Resources 11](#_Toc160804787)

[Task Table 12](#_Toc160804788)

[Gantt Chart 15](#_Toc160804789)

[Design 17](#_Toc160804790)

[Software Design 17](#_Toc160804791)

[Top Level Design 17](#_Toc160804792)

[Pseudocode 18](#_Toc160804793)

[UML Class Diagram 24](#_Toc160804794)

[UI Design 25](#_Toc160804795)

[Database Design 27](#_Toc160804796)

[ERD 27](#_Toc160804797)

[Data Dictionary 27](#_Toc160804798)

[Query Designs 27](#_Toc160804799)

[Implementation 28](#_Toc160804800)

[Software Implementation 28](#_Toc160804801)

[Program Code 28](#_Toc160804802)

[Screenshots of UI 33](#_Toc160804803)

[Database Implementation 35](#_Toc160804804)

[SQL Code 35](#_Toc160804805)

[Screenshots of Tables 36](#_Toc160804806)

[Log of Ongoing Testing 37](#_Toc160804807)

[Research and development of news skills and/or knowledge 43](#_Toc160804808)

[Testing 44](#_Toc160804809)

[Final Test Plan 44](#_Toc160804810)

[Test Reports 46](#_Toc160804811)

[Component Testing 46](#_Toc160804812)

[Integration Testing 53](#_Toc160804813)

[Interface Testing 57](#_Toc160804814)

[Usability Testing (with persona and user scenarios) 61](#_Toc160804815)

[End-User Testing (with selected candidate) 62](#_Toc160804816)

[Evaluation 65](#_Toc160804817)

[Fitness for purpose 65](#_Toc160804818)

[Future maintainability & Robustness 66](#_Toc160804819)

[APPENDIX A 67](#_Toc160804820)

# Analysis

## Description of the problem

### Purpose

This project aims to design and implement an advanced shopping list tool that allows users to log products from the online marketplace Amazon, enabling them to efficiently organise and analyse their potential purchases. The software should offer functionalities for sorting products by price as well as seamlessly outputting the data to a database for further management and analysis. The application will have an intuitive and user-friendly interface that facilitates easy navigation and efficient interaction with the software's features. This solution is tailored to PC enthusiasts and gamers who regularly seek hardware components and require a smart system to aid purchasing decisions by helping them compare various options and organise their shopping list efficiently.

### Advanced Higher concepts

The application will use the Object-Oriented programming methodology using an array of objects to store the data for each product. Objects will be sorted using the insertion sort algorithm.

There will be integration with an SQL database containing one table using various queries to enable persistent data storage.

Beyond the AH Concepts:

The process of extracting product information from websites using the Python module BeautifulSoup is a process that extends beyond the scope of the AH course.

Webscraping includes:

1. **Fetching**: The web scraping software sends an HTTP request to the target website to retrieve the web page's HTML content.
2. **Parsing**: Once the HTML content is retrieved, the scraping tool parses the HTML to extract the specific data of interest. This can include text, images, links, tables, or any other structured content.
3. **Extraction**: After parsing the HTML, the scraper identifies and extracts the desired data based on predefined criteria or patterns, such as HTML tags, CSS classes, or XPath expressions.
4. **Storage**: Finally, the extracted data is often stored in a structured format, such as a database, spreadsheet, or JSON file, for further analysis or use.

I also utilise the Tkinter library, a Python GUI toolkit, to create the simple application interface. Tkinter offers user-friendly tools and widgets for designing intuitive interfaces with buttons, menus, and text fields. It provides extensive event handling capabilities, enabling dynamic responses to user input.

### Integration

The frontend will be a windows application with a graphical user interface developed using a Python GUI library that integrates with an SQL database in the backend. The user interacts with the GUI allowing them to interpret the data stored in the database in a friendly and informational way. The program also interacts with the web as it sends HTML requests to webservers so the software can analyse and extract data.

### Constraints

Economic Constraints:

* Software and tools to be used in development are mostly open source otherwise they’re available at no cost as they are paid for by school.
* Project will be managed and implemented by a one-man development team therefore there are no wage costs for extra developers.
* PC used for development costs £2000.
* Investment in training: Python summer course costing £500.

Time Constraints:

Project must be deliverable by 1st of March therefore necessitating efficient project management and development processes to ensure timely delivery. Best time for installation is whenever possible.

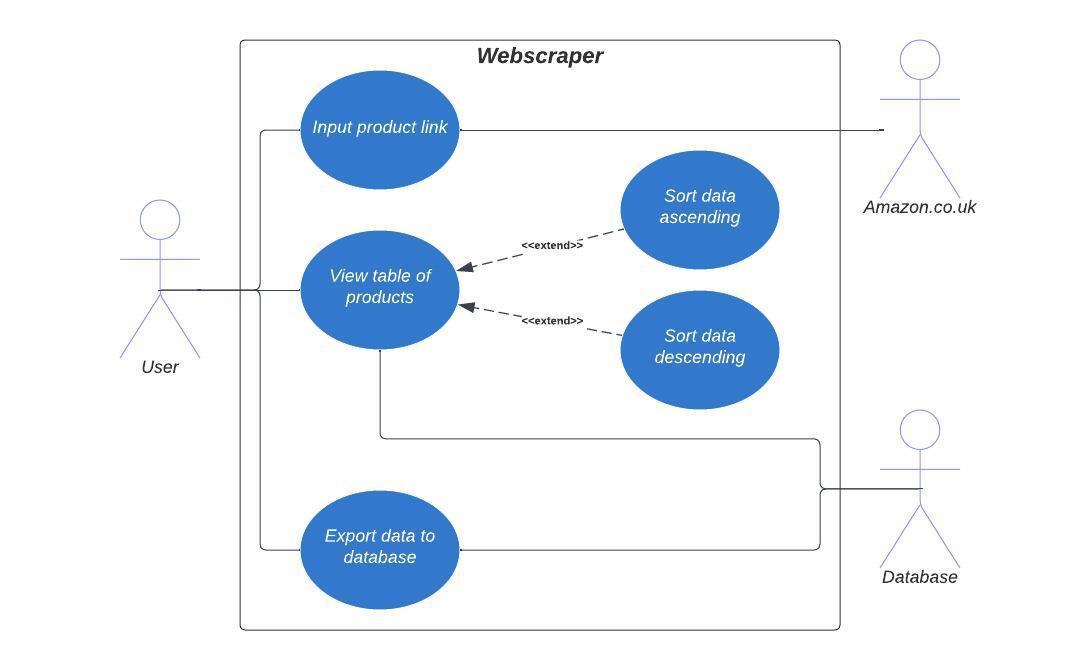
Legal Constraints:

* GDPR regulations will not be considered as the application won’t be storing personal user data.
* Amazon has some defences against web scraping bots (CAPTCHA) but it is not against the law.

Technical Constraints:

* A PC running Windows 10 or above
* A mouse/trackpad and keyboard
* Internet access
* A web browser
* XAMPP software with an external USB Flash drive to store the database

## UML Use Case Diagram



## Requirements Specification

### End-User Requirements

Users should be able to:

* enter the name of the product they want to log in the search bar and expect it to be displayed to them as well as being added to the table below.
* sort the table based on price ascending/descending.
* copy the link of a product in the table and be able to paste it into their web browser.
* export the data to an external database in order to save their list so it appears when reopening the program later.

Persona / Archetypical User

**Roger Gilmour** is a 19-year-old in his first year of university studying computer science. He is a keen computer enthusiast and loves to spend his downtime building machines in his room. He buys computer parts online every week but as he’s a student Roger has a limited budget for his expensive hobby. So, when Roger shops online he strives to find the best deals and prices for the hardware he intends to buy. This process typically takes hours of his valuable time that could be spent otherwise studying. When Roger finds an item he might want he usually writes it down on a napkin so that he can keep track of what he’s found. See, Roger really needs a smart shopping list tool on his PC (where he’s already doing his shopping and coursework) that could log all the items he’s interested in and be organised and saved for later. Roger would find great satisfaction and be very comfortable in using a software tool to aid his shopping research as he’s a computer nerd with a love for data organisation.

User Scenarios

* Roger has recently been in the process of searching for a new graphics card for his PC as his current one is not up for the challenge of modern triple A games. He opens his web browser and launces the Webscraper program finding his shopping list displayed to him just as he left it from his previous browsing.
* Roger copies the links from the table and pastes them in the search bar of his web browser so as to remind him of his options.
* Roger searches around the Amazon website and finds another new potential purchase! He copies the link of the product and pastes it into the Webscraper program. Once he’s done this he presses “Search”.
* The program displays the name and price of the product he just entered letting him know the software is adding the item Roger intended to add to his shopping list.
* Roger wants to see how expensive this item is compared to his other potential options, so he clicks the “Sort Ascending” button and discovers this product is comparatively cheaper compared to the others as its second from the top.
* Roger wants to save his shopping list so he can come back later once he’s thought about his decision, so he presses the “Export to database” button.

User Survey Analysis (see APPENDIX A)

Question 1: *What is your age range?*

Majority of respondents are teenagers or young adults.

Question 2: *What is your gender?*

Vast majority of respondents are male.

Question 3: *How frequently do you buy products online?*

Most respondents buy products online between a weekly and a monthly basis.

Question 4: *Do you try to find the best deals and prices for the products you intend to buy?*

Overwhelming majority attempt to find the best deals and prices for the products they intend to buy.

Question 5: *How much time do you usually spend researching and comparing prices before making an online purchase decision?*

Respondents tend to spend hours or even days researching and comparing prices before making online purchases.

Question 6: *How comfortable are you with using a tool to assist in your online shopping experience?*

Most respondents say they would be comfortable using a tool to assist in their online shopping experience however a third expressed a slight level of discomfort. I expect this is due to a tool such mine not already existing in the online shopping environment. I suspect this minority’s feelings will sway once user testing begins.

### Functional Requirements

Program startup

Input: Program started

Process 1: Product class is created with array setup.

Process 2: Connection to database opened.

Process 3: Database contents is read into program using SELECT queries and each item is added to the array of Product objects.

Process 4: UI is drawn with the table of products populated with array of objects.

Output: Table of products is displayed with contents from previous session.

User enters the product link:

Input: User enters product link in text box and presses search button.

Process 1: Input validation, check the string is a valid URL.

Process 2: Scrape the website for various pieces of data relevant to the product.

Process 3: Add new product with fetched data to list of products.

Output: Display data to label on screen and to table of products.

User sorts the products by ascending or descending order:

Input: Users presses the ascending or descending sort button.

Process: Insertion sort algorithm organises data by price.

Output: Display sorted products in table to user.

User exports the table of products to an external database:

Input: User presses the export button.

Process: List of products is looped through, and each item is inserted into the database using INSERT queries.

Output: Data stored in external database, connection is closed.

## Project Plan

### Resources

* Python will be the predominate programming language used for development.
* Python modules: “requests”, “BeautifulSoup”, “mysql.connector” and “tkinter”
* The IDE will be VS Code with the Python plugin.
* MySQL, Apache and the XAMPP tool will be used to integrate to database.
* MySQL commands and SQL will be used to communicate with database.
* USB to store database.
* One Windows PC will be used to do all development.
* Access to organisations network.
* OneDrive backup for database files in case USB lost.
* LucidChart to make UML Case and Class diagrams.
* Survey Monkey to produce survey.
* Survey respondents for end-user requirements.
* GanttProject for project planning.
* Figma for wireframes
* Notepad++ for todo list and editing SQL queries.
* Microsoft Word
* Google search
* StackOverflow
* Reddit
* W3Schools
* YouTube

### Task Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis | | | | |
| Start Date: 28/08/23 | | | | |
| End Date: 12/09/23 | | | | |
| TASK | | **Estimated Start Date** | **Estimated End Date** | **Resources Required** |
| Description of the problem | Problem outline | 28/08/23 | 29/08/23 | Word |
| AH Concepts |
| Integration |
| Constraints |
| UML Use Case Diagrams | Actors | 30/08/23 | 30/08/23 | LucidChart |
| Use cases |
| Relationships |
| Requirements Specification | Create user survey | 31/08/23 | 07/09/23 | Survey Monkey, survey respondents, Word |
| Analyse survey results |
| End-User Requirements |
| Persona, User stories & User scenarios |
| Functional Requirements |
| Project Plan | Identify Resources | 08/09/23 | 12/09/23 | Word, GanttProject |
| Identify tasks |
| Gantt Chart |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Design | | | | |
| Start date: 13/09/23 | | | | |
| End date: 29/09/23 | | | | |
| TASK | | **Estimated Start Date** | **Estimated End Date** | **Resources Required** |
| Software Design | Top Level Design | 13/09/23 | 28/09/23 | Word, LucidChart, Figma |
| Pseudocode |
| UML Class diagram |
| UI Design with annotated wireframe |
| Design integration |
| Database Design | Data dictionary for database | 29/09/23 | 29/09/23 | Word |
| Query designs for database |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Implementation | | | | |
| Start date: 02/10/23 | | | | |
| End date: 20/11/23 | | | | |
| TASK | | **Estimated Start Date** | **Estimated End Date** | **Resources Required** |
| Software Implementation | Program code | 2/10/23 | 20/11/23 | Visual Studio, Python and associated resources (see resource table) |
| Screenshots of program UI |
| Database Implementation | SQL code | 10/11/23 | 20/11/23 | XAMPP etc |
| Screenshots of Tables |
| Log of ongoing testing | What is being tested | 2/10/23 | 17/11/23 | Word |
| Descriptions of issues encountered |
| Descriptions of how issues were resolved |
| List of references |
| Research and development of news skills and/or knowledge | Any new skills and/or knowledge researched | 20/11/23 | 20/11/23 | Word |
| Why they were necessary |
| How new skills and/or knowledge was applied to project |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testing | | | | |
| Start date: 21/11/23 | | | | |
| End date: 29/11/23 | | | | |
| TASK | | **Estimated Start Date** | **Estimated End Date** | **Resources Required** |
| Final test plan | Project should meet all end-user and functional requirements | 21/11/23 | 22/11/23 | Word |
| Write a description of the tests to be carried out |
| Persona with a list of test cases that will be used for testing |
| Test and provide evidence | Screenshots showing inputs, outputs and any errors generated | 23/11/23 | 27/11/23 | Program testers, Word |
| Screenshots showing AH algorithms |
| Screenshots showing working SQL queries |
| Test report (with persona and test cases) | Describe results of each test | 28/11/23 | 29/11/23 | Word |

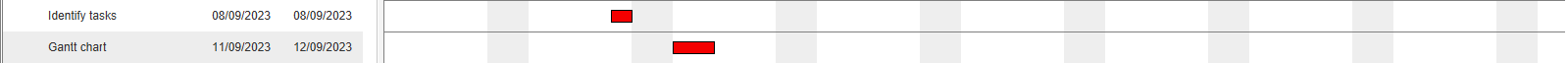
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Evaluation | | | | |
| Start date: 01/12/23 | | | | |
| End date: 06/12/23 | | | | |
| TASK | | **Estimated Start Date** | **Estimated End Date** | **Resources Required** |
| Report | Evaluate fitness of the solution to solve the problem | 01/12/23 | 06/12/23 | Word |
| Discuss how closely solution matches all requirements stated in requirements specification |
| Include results of testing |
| Discuss the future maintainability and robustness of solution |

### Gantt Chart

Analysis

A screenshot of a video game

Description automatically generated



Design

A screenshot of a graph

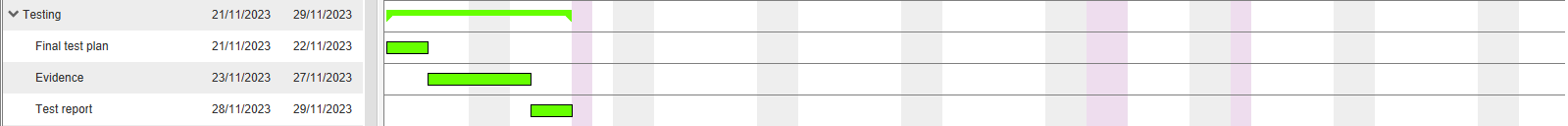
Description automatically generated

Implementation

A screenshot of a graph

Description automatically generated

Testing



Evaluation

A close up of numbers

Description automatically generated

# Design

## Software Design

### Top Level Design

Program Start

Search button pressed

Sort buttons pressed

Export button pressed

### Pseudocode

**The “Product” Class and establishing the array of objects:**

CLASS Product()

CONSTRUCTOR (name, price, amazon\_url):

name ⬅️ name

price ⬅️ price

amazon\_url ⬅️ amazon\_url

END CONSTRUCTOR

ENDCLASS

ARRAY of Products ⬅️ []

**Opening connection to database (integration):**

Setup a connection to the database using MySQL connector:

Host as “localhost”

User as “root”

Password as “”

Database as “webscraper”

**Populating array of objects with data from the database (integration):**

**(query designs shown on p27)**

Create “cursor” object to execute SQL queries.

Use the cursor object to execute SQL query to fetch data from database.

Use cursor to fetch all rows from results set.

Iterate over the rows and create Product objects.

FOR row in rows:

name, price, amazon\_url ⬅️ row

Add new Product object to array of Product objects with name, price and amazon\_url as attributes.

ENDFOR

**Creating the UI:**

Create the tkinter application window.

Set the name of the application window to “Amazon Webscraper”.

Product link entry field

Create a label that displays “Enter Product Link: ”.

Render the label in the top left corner of the window.

Create an entry field for the product link with a width of 90.

Render the entry field in the top right corner of the window.

Create a button that displays “Search” and links to the “search\_product” function.

Render the button in the centre of the row below.

Labels for the product name and price just entered (not visible until user input)

Create a label that displays “Product Name”.

Create a label that displays “Price”.

Display the table of products

Create a tkinter treeview widget with the columns “Product Name”, “Price”, “URL”.

Set the headings of each column to have the text “Product Name”, “Price”, “URL” respectively.

Set the width of the “Product Name” column to 500.

Set the width of the “Price” column to 40.

Set the width of the “URL” column to 150.

Render the table.

Add the products stored in the database to the table now that it has been drawn

CALL display\_products(products)

Sort ascending/descending buttons

Create a button that displays “Sort Ascending” and links to the “sort\_products\_ascending” function.

Render the button in the bottom left corner.

Create a button that displays “Sort Descending” and links to the “sort\_products\_descending” function.

Render the button adjacent to the other sort button.

Export to database button

Create a button that displays “Export to database” and links to the “export\_to\_database” function.

Render the button in the bottom right corner.

Start the tkinter event loop.

**“Search Product” function:**

FUNCTION search\_product()

product\_link ⬅️ what’s stored in the input box

    CALL fetch\_data(product\_link)

Render the product name and price labels to display the product logged to the user

    CALL display\_products(products)

ENDFUNCTION

**“Fetch Data” function:**

FUNCTION fetch\_data(product\_link)

    Set headers to mimic a web browser to avoid being blocked

    headers ⬅️ The headers

    Send an HTTP GET request to the Amazon URL

Raise an exception if response indicates an error eg error 404 not found.

If so, update the text in the product\_price\_label to the error.

    Parse the HTML content of the page using BeautifulSoup

Instantiate a BeautifulSoup object “soup” that can be used to parse the HTML received from the HTML request.

    Extract product name from page

Set product\_name to the value of the relevant text element in the HTML code using the find() method.

    Render the product\_name label to user.

    Extract product price from page

Set product\_price to the value of the relevant text element in the HTML code using the find() method.

Render the product\_price label to user.

    Remove £ sign from price string.

    Instantiate a new product object and add it to the array of objects with this scraped data (price converted to float from string)

ENDFUNCTION

**“Display Products” function:**

FUNCTION display\_products(products):

Delete all rows in table.

FOR product in products:

name ⬅️ product.name

price ⬅️ product.price

amazon\_url ⬅️ product.amazon\_url

Insert query to insert a new row into the table with the values name, price and amazon\_url

ENDFOR

ENDFUNCTION

**Insertion sort algorithms ascending / descending:**

FUNCTION sort\_products\_asc(products):

    FOR i ⬅️ 1 TO LENGTH(Products)-1 DO

        current\_product ⬅️ products[i]

        j ⬅️ i - 1

        WHILE j >= 0 AND current\_product.price < products[j].price DO

            products[j + 1] ⬅️ products[j]

            j ⬅️ j – 1

ENDWHILE

        products[j + 1] ⬅️ current\_product

ENDFOR

    Update the table UI

    CALL display\_products(products)

ENDFUNCTION

FUNCTION sort\_products\_asc(products):

    FOR i ⬅️ 1 TO LENGTH(Products)-1 DO

        current\_product ⬅️ products[i]

        j ⬅️ i - 1

        WHILE j >= 0 AND current\_product.price > products[j].price DO

            products[j + 1] ⬅️ products[j]

            j ⬅️ j – 1

ENDWHILE

        products[j + 1] ⬅️ current\_product

ENDFOR

    Update the table UI

    CALL display\_products(products)

ENDFUNCTION

**“Export to database” function (integration):**

FUNCTION export\_to\_database(products):

SQL Delete query to clear database.

FOR product in products:

SQL Insert query to add a new row into the table with the values name, price and amazon\_url

ENDFOR

Commit the query.

Close connection to database

ENDFUNCTION

### UML Class Diagram

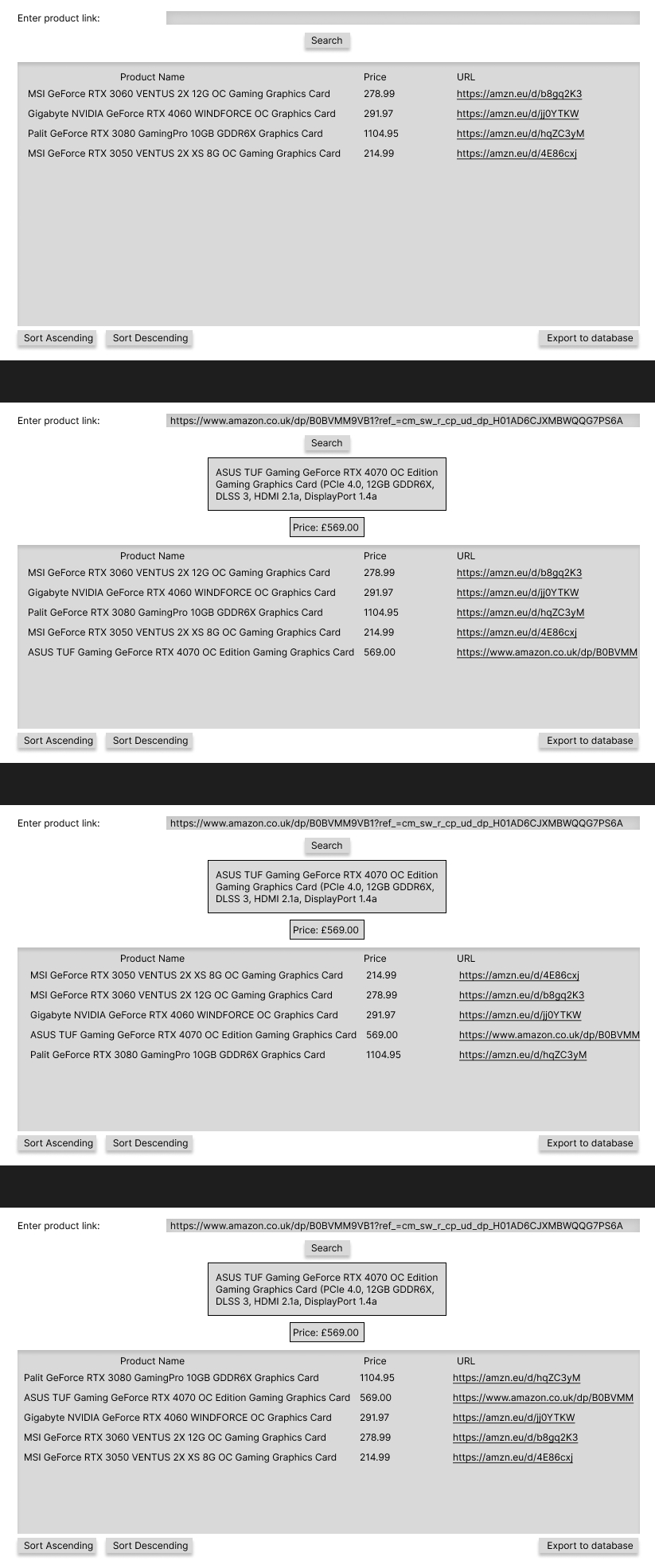


### UI Design

A screenshot of a computer

Description automatically generatedInitial screen

Product class is made, and array is setup. Connection to database is opened, contents is added to array of product objects and the UI is drawn with the table populated.

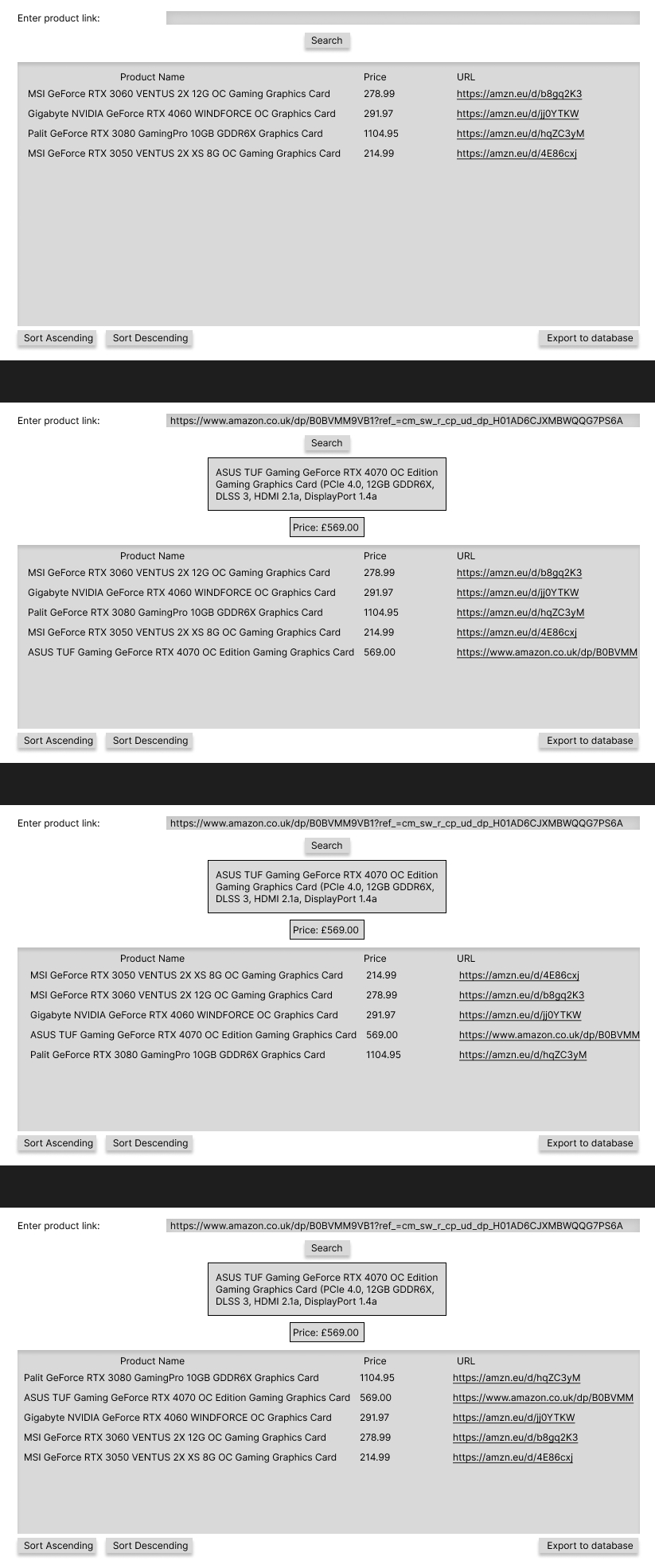
User inputs product link and search button pressed

User inputs amazon url here

Program checks link is valid once search button pressed (returns an error if input is not a web url or if the page cannot be accessed)

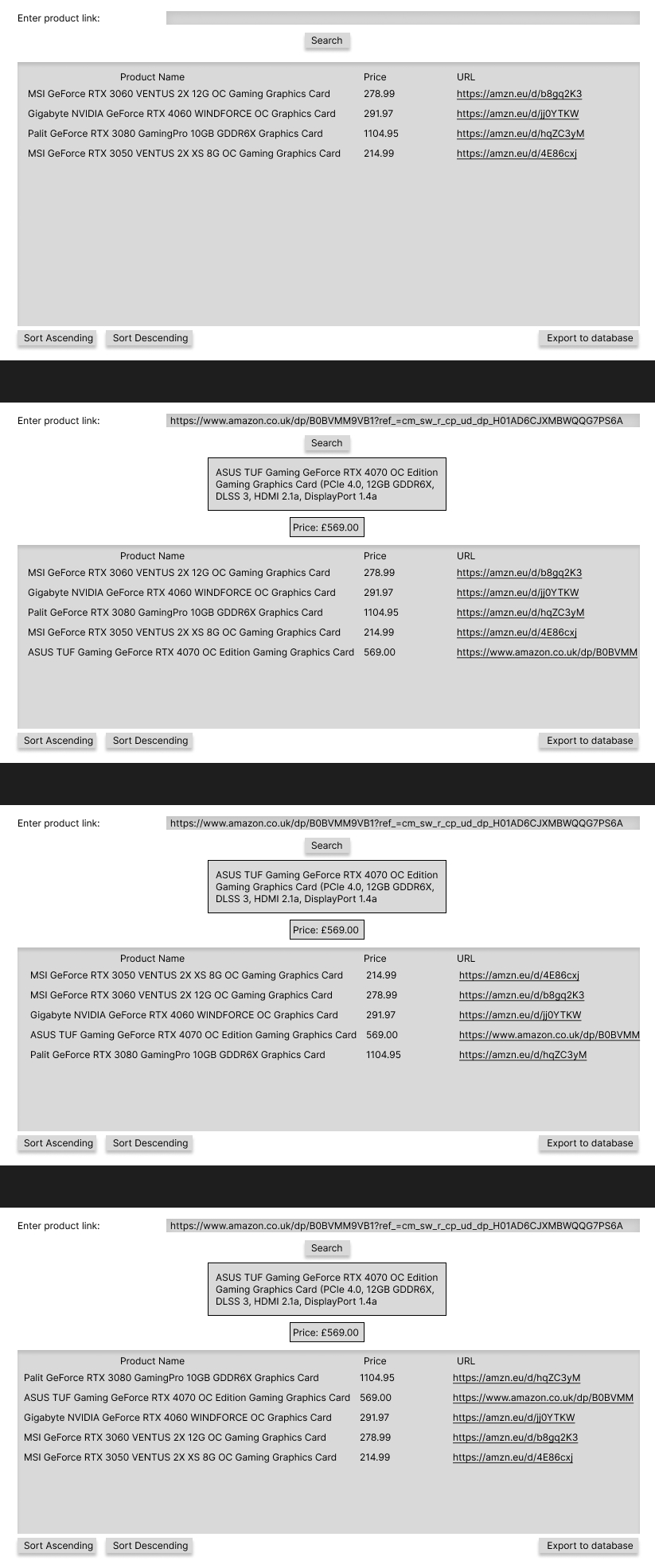
Name and price of product displayed to user in labels and new product added to the bottom of the table.

Website is scraped for relevant pieces of data and a new product object is appended to the array.

User clicks “sort ascending” button

“Sort Ascending” button is clicked, and the insertion sort algorithm orders the products by price ascending and updates the table.

User clicks “sort descending” button



“Sort Descending” button is clicked, and the insertion sort algorithm orders the products by price descending and updates the table.

## Database Design

### ERD

No ERD required as database is a flat file database.

### Data Dictionary

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attribute | Key | Type | Size | Required | Validation | Default |
| name | N/A | longtext | 4,294,967,295 characters | No | No | NULL |
| price | N/A | float | N/A | No | No | NULL |
| amazon\_url | N/A | longtext | 4,294,967,295 characters | No | Yes | NULL |

### Query Designs

SELECT query to fetch data from database at startup:

Fields: name, price, amazon\_url

Tables: Product

Search Criteria: none

DELETE query to delete all items in database:

Fields: name, price, amazon\_url

Tables: Product

Search Criteria: none

INSERT query to add each item in the array of product objects into the database:

Fields: name, price, amazon\_url

Tables: Product

Search Criteria: None

Values: variables in code

# Implementation

## Software Implementation

### Program Code

A white background with text

Description automatically generated

A white background with text

Description automatically generated

A white background with text

Description automatically generated

A white background with text on it

Description automatically generatedA white background with text

Description automatically generated

### Screenshots of UI

A screenshot of a computer

Description automatically generatedInitial Screen

A screenshot of a computer

Description automatically generated User inputs product link and search button pressed

A screenshot of a computer

Description automatically generatedUser clicks “sort ascending” button

A screenshot of a computer screen

Description automatically generatedUser clicks “sort descending” button

## Database Implementation

### SQL Code

MySQL code to create the database and the “Product” table:

CREATE DATABASE Webscraper;

CREATE TABLE Products (

name LONGTEXT,

price FLOAT,

amazon\_url LONGTEXT

);

Query to fetch data from database at startup:

SELECT name, price, amazon\_url FROM products

Query to delete all items in the database:

DELETE FROM Products;

Query to add each item in the array of product objects into the database:

INSERT INTO Products (name, price, amazon\_url) VALUES (%s, %s, %s)

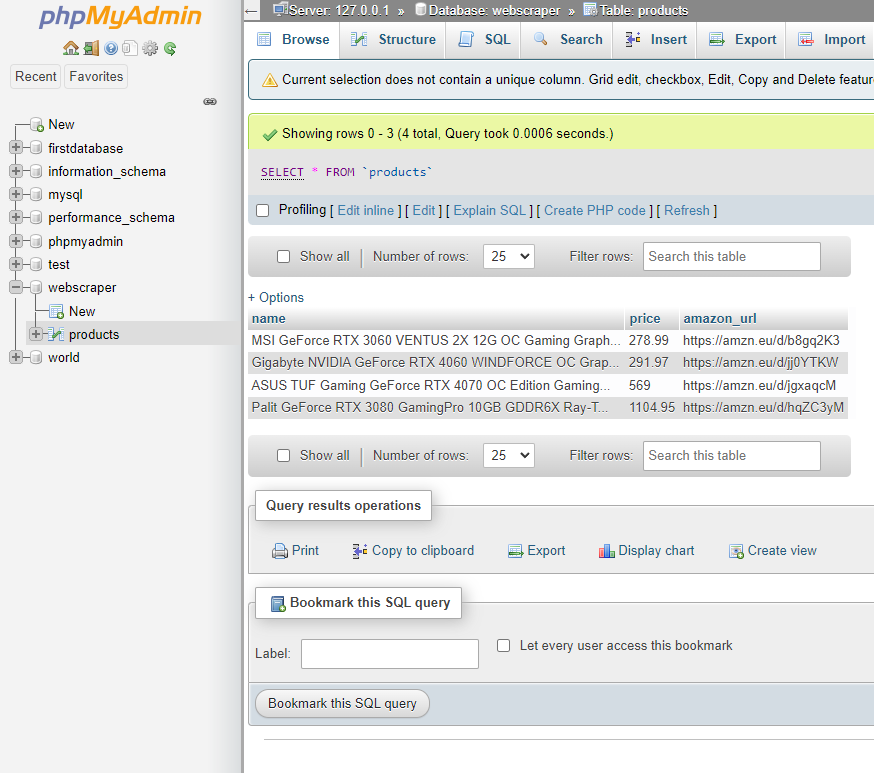
### Screenshots of Tables

Structure of implemented product table matches design:

A screen shot of a computer program

Description automatically generated

Initial data stored in table:



## Log of Ongoing Testing

**1**

|  |
| --- |
| **What is being tested?** |
| Connection to database works (integration testing) |
| **What issue occurred?** |
| Program runs but it cannot connect to database. |
| **How was issue resolved?** |
| Run xampp\_start.exe before opening the control panel. |
| **References used to solve issue:** |
| None |

**2**

|  |
| --- |
| **What is being tested?** |
| Table is populated on startup |
| **What issue occurred?** |
| Program did not run. |
| **How was issue resolved?** |
| Database name is “webscraper” not “product” (which is the table name). |
| **References used to solve issue:** |
| MySQL documentation. |

**3**

|  |
| --- |
| **What is being tested?** |
| User interface is drawn accurately according to design |
| **What issue occurred?** |
| Export button appeared in wrong place. |
| **How was issue resolved?** |
| Change the column number of the button to “1” not “2” |
| **References used to solve issue:** |
| Tkinter documentation |

**4**

|  |
| --- |
| **What is being tested?** |
| Search button works and product data is successfully fetched from amazon page |
| **What issue occurred?** |
| Product name and price could not be found on webpage as it couldn’t identify the HTML element that contained that value. |
| **How was issue resolved?** |
| Looked at the HTML code for the amazon webpage with the Chrome inspect element tool and discovered that different webpages have slightly different layouts. Program only works if the price of the product is not discounted and if there aren’t multiple price points. |
| **References used to solve issue:** |
| Google, YouTube |

**5**

|  |
| --- |
| **What is being tested?** |
| New product is added to array |
| **What issue occurred?** |
| No issue, the product was successfully added to the array of objects with the relevant data. |
| **How was issue resolved?** |
| N/A |
| **References used to solve issue:** |
| None |

**6**

|  |
| --- |
| **What is being tested?** |
| New product is displayed at end of table |
| **What issue occurred?** |
| Product was added to end of table but all other items in table were duplicated. |
| **How was issue resolved?** |
| Line 69 was added to delete all items already displayed in the table so when table is redrawn with the additional product, the previous items aren’t duplicated. |
| **References used to solve issue:** |
| Tkinter documentation |

**7**

|  |
| --- |
| **What is being tested?** |
| Sort ascending button works, and table is reordered |
| **What issue occurred?** |
| Products were ordered the wrong way round. |
| **How was issue resolved?** |
| “>” changed to “<” |
| **References used to solve issue:** |
| None |

**8**

|  |
| --- |
| **What is being tested?** |
| Sort descending button works, and table is reordered |
| **What issue occurred?** |
| No issues as previous test identified a mix up in the greater than and less than signs. |
| **How was issue resolved?** |
| N/A |
| **References used to solve issue:** |
| None |

**9**

|  |
| --- |
| **What is being tested?** |
| Export to database functionality works |
| **What issue occurred?** |
| Error when export to database button pressed. |
| **How was issue resolved?** |
| Didn’t commit the SQL Query (line 26) |
| **References used to solve issue:** |
| StackOverflow |

## Research and development of news skills and/or knowledge

|  |
| --- |
| **Skill/Knowledge Researched** |
| How to use the Python module Tkinter (researched using YouTube and the Tkinter documentation). |
| **Why was the skill/knowledge necessary** |
| I needed to learn how to use Tkinter to build the UI for my program. Tkinter is a very easy to use module that can be used to create simple GUIs for a windows application. |
| **How skill/knowledge was applied to the project** |
| I used Tkinter to build the UI for my program and I had each button link to a function. |

|  |
| --- |
| **Skill/Knowledge Researched** |
| How to use XAMPP (researched using my teachers, Google and YouTube) |
| **Why was the skill/knowledge necessary** |
| I need XAMPP, Apache and MySQL to integrate my project with a backend database. |
| **How skill/knowledge was applied to the project** |
| Used XAMPP with a USB Flash drive to store my products database. |

|  |
| --- |
| **Skill/Knowledge Researched** |
| How to use the BeautifulSoup python module. |
| **Why was the skill/knowledge necessary** |
| So I could successfully webscrape data off of the amazon product pages. |
| **How skill/knowledge was applied to the project** |
| Used the BS module and find() method to webscrape product data from the relevant elements within the requested HMTL files. |

|  |
| --- |
| **Skill/Knowledge Researched** |
| How complex webpages are structured. |
| **Why was the skill/knowledge necessary** |
| The program must scrape data from large complex HTML files so it was necessary to learn more complex syntax and explore the pages using the inspect element tool. |
| **How skill/knowledge was applied to the project** |
| I was able to locate the elements in the HTML that contained the data I needed and wrote code that could retrieve it. |

# Testing

## Final Test Plan

**Component Testing**

For component testing I will test each individual section of my code to ensure they work as expected and meet my functional requirements. I have omitted any parts related to the integration with the backend as this is a separate testing section.

|  |  |  |
| --- | --- | --- |
| Test ID | Test Case | Test Passed? |
| 1.1 | Product class created and array setup | ✅ |
| 1.2 | Products displayed in table | ✅ |
| 1.3 | User input is validated | ✅ |
| 1.4 | Data successfully fetched from webpage | ✅ |
| 1.5 | New product data displayed to user in labels | ✅ |
| 1.6 | New product created and added to array | ✅ |
| 1.7 | Insertion sort ascending works | ✅ |
| 1.8 | Insertion sort descending works | ✅ |

**Integration Testing**

For integration testing I will be checking the backend and the main program communicate correctly.

|  |  |  |
| --- | --- | --- |
| Test ID | Test Case | Test Passed? |
| 2.1 | Connection to database opens | ✅ |
| 2.2 | Data read in from database into product array | ✅ |
| 2.3 | Data successfully exported | ✅ |
| 2.4 | Connection to database closed | ✅ |

**Interface Testing**

For interface testing I will compare my program user interface to my wireframe designs using this as an indication of the accuracy of my interface implementation. I will focus on checking positioning and sizes of elements as well as ensuring text is easily readable.

|  |  |  |
| --- | --- | --- |
| Test ID | Test Case | Test Passed? |
| 3.1 | Startup screen | ✅ |
| 3.2 | “Search” button pressed | ✅ |
| 3.3 | “Sort Ascending” button pressed | ✅ |
| 3.4 | “Sort Descending” button pressed | ✅ |

**Usability Testing (with persona and user scenarios)**

For usability testing, I will complete tests based on the persona of Roger Gilmour created during the analysis section of my project.

|  |  |  |
| --- | --- | --- |
| Test ID | Test Case | Test Passed? |
| 4.1 | Program opens with Roger’s shopping list displayed to him | ✅ |
| 4.2 | Roger can copy the link from the table | ✅ |
| 4.3 | Roger can input the new product link and press “Search” | ✅ |
| 4.4 | Roger can see the name and price of the new product displayed to him | ✅ |
| 4.5 | Roger can sort the table by price ascending | ✅ |
| 4.6 | Roger can save the state of the list by pressing “Export to database” | ❌ |

**End-User Testing (with selected candidate)**

For end-user testing I will provide a selected candidate with a list of instructions to follow. I will closely observe how the user interacts with my interface and note how my software performs.

|  |  |  |
| --- | --- | --- |
| Test ID | Test Case | Test Passed? |
| 5.1 | User opens blank shopping list | ✅ |
| 5.2 | User inputs an invalid URL and presses “Search” | ❌ |
| 5.3 | User inputs another new product link and presses “Search” | ❌ |

## Test Reports

### Component Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.1 | Product class created and array setup | Product class is defined with attributes: name, price and amazon\_url. Array is defined. | As excepted. | ✅ |
| Output: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.2 | Products displayed in table | The table in the UI is populated with the products including the name, price and URL. | As expected. | ✅ |
| Output in UI: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.3 | User input is validated | The input link is validated to ensure it is an accessible webpage. | As expected. | ✅ |
| Output (link is valid so product data is displayed to user as expected later on):  A screenshot of a computer  Description automatically generated | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.4 | Data successfully fetched from webpage | The name and price of the product is successfully fetched and stored in variables. | As expected. | ✅ |
| The name and price of the product is fetched using BeautifulSoup and stored in the variables: “product\_name” and “product\_price” on lines 86 and 91.    The product page:        Output: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.5 | New product data displayed to user in labels | The new product is displayed at the bottom of the UI table. | As expected. | ✅ |
| Output:  A screenshot of a computer  Description automatically generated | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.6 | New product created and added to array | A new product object is instantiated with the required attributes and added to the product array. | As expected. | ✅ |
| Output: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.7 | Insertion sort ascending works | Button works and sort algorithm correctly sorts data. New arrangement displayed correctly in table. | As expected. | ✅ |
| Output: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 1.8 | Insertion sort descending works | Button works and sort algorithm correctly sorts data. New arrangement displayed correctly in table. | As expected. | ✅ |
| Output: | | | | |

### Integration Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 2.1 | Connection to database opens | Program and MySQL connect without any errors. | As expected. | ✅ |
| XAMPP running with no issues:    Output:  Program starts with no errors that indicate a failed connection to the database. | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 2.2 | Data read in from database into product array | Data from the database is correctly read using the SQL SELECT query and each row is converted into a new product object and added to the products array. | As expected. | ✅ |
| Data in database:    Data now in products array: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 2.3 | Data successfully exported | Each product object in the products array is successfully inserted into the database using SQL INSERT queries. | As expected. | ✅ |
| Data in products array:    Data now exported into database: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 2.4 | Connection to database closed | Connection to database closed. | As expected. | ✅ |
| Output:  Connection to database closed. | | | | |

### Interface Testing

Please note the actual products in the table are arbitrary.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 3.1 | Startup screen | Startup screen UI appears similar to wireframe design. | As expected. | ✅ |
| Wireframe Design:  A screenshot of a computer  Description automatically generated  Actual UI:  A screenshot of a computer  Description automatically generated | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 3.2 | “Search” button pressed | Screen once the “Search” button is pressed appears similar to wireframe design. | As expected. | ✅ |
| Wireframe:    Actual UI:  A screenshot of a computer  Description automatically generated | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 3.3 | “Sort Ascending” button pressed | Screen once the “Sort Ascending” button is pressed appears similar to wireframe design. | As expected. | ✅ |
| Wireframe:    Actual UI:  A screenshot of a computer  Description automatically generated | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 3.4 | “Sort Descending” button pressed | Screen once the “Sort Descending” button is pressed appears similar to wireframe design. | As expected. | ✅ |
| Wireframe:    Actual UI:  A screenshot of a computer screen  Description automatically generated | | | | |

### Usability Testing (with persona and user scenarios)

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.1 | Program opens with Roger’s shopping list displayed to him | Roger’s shopping list from last time is displayed in the UI table so he can continue with his research immediately. | ✅ |

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.2 | Roger can copy the links from the table | Roger double clicks the links in the UI table and pastes them into his web browser. | ✅ |

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.3 | Roger can input the new product link and press “Search” | Roger inputs his new link into the search field and presses “Search” | ✅ |

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.4 | Roger can see the name and price of the new product displayed to him | Roger sees the name and price of the product he just entered, verifying to him that the program is adding the correct item to his list. | ✅ |

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.5 | Roger can sort the table by price ascending | Roger presses the “Sort Ascending” button and sees the items in his list reordered. | ✅ |

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Result | Passed? |
| 4.6 | Roger can save the state of the list by pressing “Export to database” | Roger managed to press the button but found it confusing there was no verification that the button had worked. | ❌ |

### End-User Testing (with selected candidate)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 5.1 | User opens blank shopping list | Program opens with nothing in the UI table. | As expected. | ✅ |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 5.2 | User inputs an invalid URL and presses “Search” | The program displays an error message indicating the link is invalid. | Program did not alert user that the link was invalid however the terminal did. | ❌ |
| Test subject input the link for an EBAY webpage not an Amazon webpage: <https://www.ebay.co.uk/itm/226022900962?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=yQcNkM8VRrK&sssrc=2047675&ssuid=&widget_ver=artemis&media=COPY>  Output (Name and price elements could not be scraped because the webpage has a different structure): | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Expected Result | Actual Result | Passed? |
| 5.3 | User inputs another new product link and presses “Search” | The product name and price are displayed to user as well as being added to the end of the table. | The product price is displayed in the label, but it contains additional text. A new Product object cannot be created as it cannot convert the string to a float. Hence the new product is not displayed at the bottom of the table. | ❌ |
| The product has a discount applied so the price element in the HTML contains additional text along with the numerical value:    Output:  Program displays price in price label, but it also includes additional text describing the applied discount.  New product is not added to table as there was an error with creating the product object since the string cannot be converted to a float.    Console error: | | | | |

# Evaluation

### Fitness for purpose

Meeting of Requirements:

1. Program startup

The program effectively completes all startup processes, ensuring the UI is properly populated with existing data from the database.

1. Product data retrieval

The program accurately retrieves and displays product details from user-provided links at rapid speed. However, testing has concluded that some products do not get added to the table upon input. This is because some products on Amazon have applied discounts meaning the price element in the HTML that contains the numerical value of the product also contains additional text that confuses the program and stops it from being able to instantiate a new product object and add it to the table. As well as this, the validation messages of user inputs did not display to the user but did show as an output in the program terminal.

1. Data sorting

The sorting functionality efficiently organises product data based on price.

1. Exporting to external database

The export feature seamlessly transfers product data to an external database, ensuring data persistence and facilitating further analysis or usage. However, testing concluded the user found it confusing that there was no pop up to confirm to the user that the export button has been clicked and that their data has been safely exported.

Further points:

* During testing I found that the price of the product was sometimes unable to be found by my program whilst the name of the product always was found. Upon further investigation, I could not conclude why this was as there was no correlation between what product I was scraping and when the program couldn’t find the price data. Sometimes the same product’s price data could be found one day and then not the next day. Further research needed.
* There is no use of getter or setter methods in my code as my program is so simple there is no need for abstraction.
* Future development could include converting from a web scraping based program into using the official Amazon API, but for my project I did not have access to this. I could have also included a search feature for the list as well as the ability to delete products previously added to the list. I could include scraping more data from each product such as the stock level and review star rating. Unfortunately due to time constraints I could not allow this scope creep.

Overall, my solution meets all functional requirements outlined, providing a robust solution for retrieving, organizing, and storing product data from the web. The implementation demonstrates proficiency in web scraping techniques, database interaction, and user interface development.

Top of Form

### Future maintainability & Robustness

My program code is structured in a logical order with the functions defined appropriately. There is an abundance of clear and concise comments to aid maintainability. However, as for robustness, there are some inputs that throw errors such as invalid URLs, or links to certain products that have discounts applied to them meaning the program can’t find the price.

# APPENDIX A

End-User Survey Results

1. *What is your age range?A graph with different colored bars

Description automatically generated*

2. *What is your gender?A graph with different colored squares

Description automatically generated*

3. *How frequently do you buy products online?*

A graph with different colored rectangles

Description automatically generated

4. *Do you try to find the best deals and prices for the products you intend to buy?*

A graph with different colored squares

Description automatically generated

5. *How much time do you usually spend researching and comparing prices before making an online purchase decision?*

A graph with different colored rectangles

Description automatically generated

6. *How comfortable are you with using a tool to assist in your online shopping experience?* A graph with different colored rectangles

Description automatically generated