**COMP1204 Data Management**

**Coursework-2 LINUX**

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **No.** | **Name** | **University Email** |
| 1. | Brandon Ting Wee Kang | bwkt1n22@soton.ac.uk |
| 2. | Pang Jia Hui | jhp1n22@soton.ac.uk |

**Submitted Documents in ZIP File:**

|  |  |  |
| --- | --- | --- |
| **No.** | **Documents** | **File Name** |
| 1. | Report | DM CW2 – Report.pdf |
| 2. | Scraper/Tracker Script: | product\_details\_tracker.sh  product\_details\_graph.sh |
| 3. | Database | pricetracker.sql |
| 4. | Github Repository | <https://github.com/brandon-nx/OnlinePromotionStoreTracker> |

# 1.0 Introduction

The project is an online store promotion tracker that monitors key data such as pricing, rating, stock availability and number of items sold for products listed on the Shoppu platform. By collecting this data regularly and tracking changes over time, the tracker intends to assist customers in making more informed purchasing decisions. In today's dynamic online business, where prices and promotions change quickly, automated tracking of online store promotions has various advantages over manual monitoring. It allows consumers to identify price trends and promotions easily. Furthermore, the tracker simplifies the decision-making process for customers, allowing them to make quick decisions and act promptly on opportunities to save money.

The project collects data using Unix scripts to ensure that it is collected within a particular period and stored in a MySQL database. Once enough data has been collected, the next step is to produce graphs to visualise trends and analyse patterns in the data, allowing users to track data and make quick decisions about purchasing things on an online platform.   
Overall, the online store promotion tracker is an effective tool for consumers seeking to optimize their online shopping experience by staying up to date with special offers, price variations, and stock reminders on the Shoppu platform. The tracker enables customers to make better purchase decisions and save more money by providing them with fast and accurate information.

# 2.0 Table of Content

[1.0 Introduction 2](#_Toc166339588)

[2.0 Table of Content 3](#_Toc166339589)

[3.0 Timeline 4](#_Toc166339590)

[4.0 Database Design 5](#_Toc166339591)

[5.0 Scraper/Tracker Script (product\_details\_tracker.sh) 6](#_Toc166339592)

[5.1 Fetching Web Data 6](#_Toc166339593)

[5.2 Parsing Data 7](#_Toc166339594)

[5.3 Data Manipulation 8](#_Toc166339595)

[5.4 Insert into Database 9](#_Toc166339596)

[6.0 Contab Setup and Error Handling 11](#_Toc166339597)

[6.1 Crontab Setup 11](#_Toc166339598)

[6.2 Error Handling Examples 12](#_Toc166339599)

[7.0 Plotting Script (product\_details\_graph.sh) 13](#_Toc166339600)

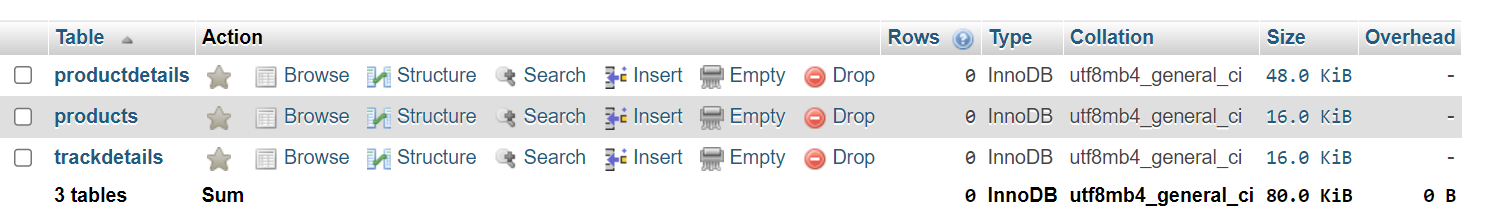
[8.0 Conclusion 17](#_Toc166339601)

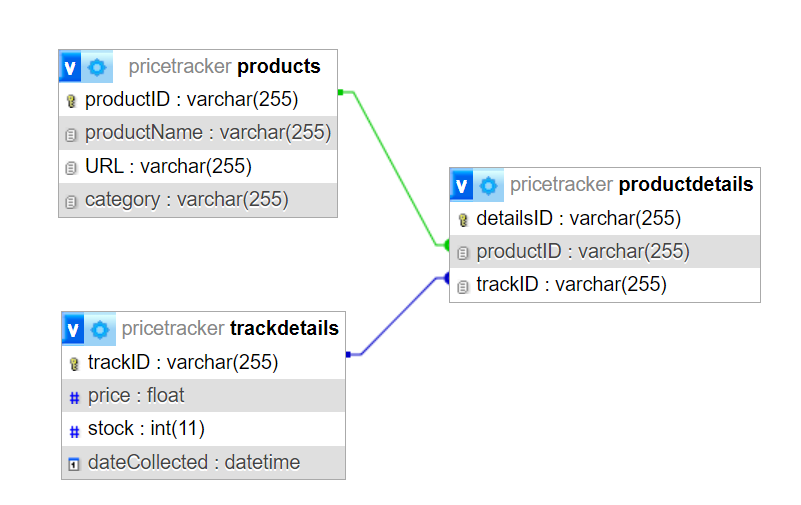
[9.0 Appendices 18](#_Toc166339602)

# 3.0 Timeline



# 4.0 Database Design





**Table ‘productdetails’**



**Table ‘products’**



**Table ‘trackdetails’**



# 5.0 Scraper/Tracker Script (product\_details\_tracker.sh)

## 5.1 Fetching Web Data

PRODUCT\_URLS=(

    https://www.publicpackaging.com/showproducts/productid/4312040/cid/448076/11pcs-foodgrade-silicone-kitchen-measuring-tools-ready-stock-measuring-spoon/

    https://www.publicpackaging.com/showproducts/productid/4312333/cid/448072/super-clean-gel-compound-cleaning-gel-jelly-dust-cleaning-70g%E5%8D%A4/,

    https://www.publicpackaging.com/showproducts/productid/4312382/cid/448073/dish-wash-pure-colour-pad-2-pcs-in-1-pack/,

    https://www.publicpackaging.com/showproducts/productid/4312283/cid/448072/creative-desktop-shake-lid-mini-trash-bin-%E5%88%9B%E6%84%8F%E6%A1%8C%E9%9D%A2%E6%91%87%E7%9B%96%E8%BF%B7%E4%BD%A0%E5%9E%83%E5%9C%BE%E6%A1%B6/,

    https://www.publicpackaging.com/showproducts/productid/4312254/cid/448073/kitchen-knife-3pcs-set-fruit-knife-pemotong-sayur-dadu-multi-slicer-%E6%B0%B4%E7%9A%AE%E6%B0%B4%E6%9E%9C%E5%88%80%E6%B2%BE%E6%9D%BF%E4%B8%89%E4%BB%B6%E5%A5%97/

)

for URL in "${PRODUCT\_URLS[@]}"; do

    PRODUCT\_NAME=$(echo "$URL" | awk -F '/' '{print $(NF-1)}' | cut -c1-30)

    OUTPUT\_FILE="${PRODUCT\_NAME}\_page.html"

    curl -s "$URL" -o "$OUTPUT\_FILE"

    # Check if curl succeeded

    if [ $? -ne 0 ]; then

        echo "Failed to fetch data from $URL"

        continue

    fi

    echo "Web data successfully fetched and saved to $OUTPUT\_FILE"

    parseData "$OUTPUT\_FILE" "$PRODUCT\_NAME" "$URL"

done

## 5.2 Parsing Data

Use tools like **grep** and **awk** to extract relevant information from the HTML content.

parseData() {

local file="$1"

local product\_name="$2"

local product\_url="$3"

# Parse price

price=$(grep -oP 'product:price:amount" content="\K[\d.]+' "$file")

if [ -z "$price" ]; then

echo "Error: Price not found for product: $product\_name"

return 1

fi

# Parse stock

stock=$(awk 'BEGIN{RS="<"; FS=">"; IGNORECASE=1} /class="product\_qty\_availble"/ && !found {print $2; found=1}' "$file" | grep -oP '\d+' | head -n 1 | tr -d '\n')

if [ -z "$stock" ]; then

echo "Error: Stock not found for product: $product\_name"

return 1

fi

# Parse category

category=$(grep -oP 'property="product:category" content="\K[^"]+' "$file")

if [ -z "$category" ]; then

echo "Error: Category not found for product: $product\_name"

return 1

fi

echo "Parsed Data: Product: $product\_name, Price: RM$price, Stock: $stock, Category: $category"

dataManipulation "$product\_name" "$price" "$stock" "$category" "$product\_url"

}

## 5.3 Data Manipulation

Process and convert extracted data into appropriate formats.

dataManipulation() {

local product\_name="$1"

local price="$2"

local stock="$3"

local category="$4"

local product\_url="$5"

# Check if price is a valid number

if ! [[ $price =~ ^[0-9]+(\.[0-9]+)?$ ]]; then

echo "Error: Invalid price format for product: $product\_name"

return 1

fi

# Check if stock is a valid integer

if ! [[ $stock =~ ^[0-9]+$ ]]; then

echo "Error: Invalid stock format for product: $product\_name"

return 1

fi

# Convert price to a float if it's not already

price=$(printf "%.2f" "$price")

# Ensure stock is an integer

stock=$(printf "%d" "$stock")

echo "Manipulated Data: Product: $product\_name, Price: RM$price, Stock: $stock, Category: $category"

insertIntoDatabase "$product\_name" "$price" "$stock" "$category" "$product\_url"

}

## 5.4 Insert into Database

Add MySQL commands to insert collected data into the database, ensuring error handling for database interactions.

insertIntoDatabase() {

product\_name="$1"

price="$2"

stock="$3"

category="$4"

product\_url="$5"

# Get the current datetime

current\_datetime=$(date '+%Y-%m-%d %H:%M:%S')

# Insert data into the products table

url\_exists=$(checkURLExists "$product\_url")

if [ "$url\_exists" -eq 0 ]; then

new\_product\_id=$(generateProductID)

# Insert new product if URL does not exist

if ! insert\_data "products" "productID, productName, category, URL" "'$new\_product\_id', '$product\_name', '$category', '$product\_url'"; then

return 1

fi

else

echo "No need to insert product; URL already exists."

new\_product\_id=$(getProductIDbyURL "$product\_url") # Get existing productID if URL exists

fi

# Insert data into the trackdetails table

new\_track\_id=$(generateTrackID)

if ! insert\_data "trackdetails" "trackID, price, stock, dateCollected" "'$new\_track\_id', $price, $stock, '$current\_datetime'"; then

return 1

fi

# Insert data into the productdetails table

new\_details\_id=$(generateDetailsID)

if ! insert\_data "productdetails" "detailsID, productID, trackID" "'$new\_details\_id', '$new\_product\_id', '$new\_track\_id'"; then

return 1

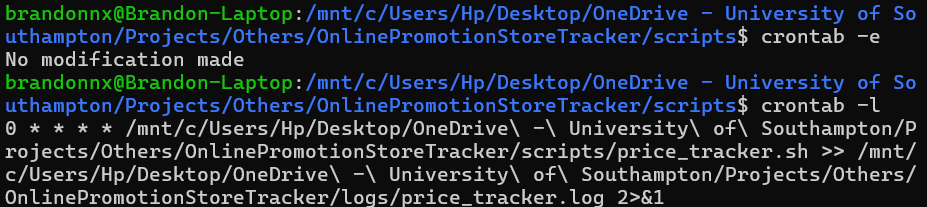
fi

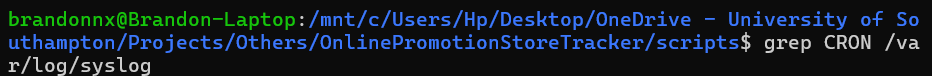
}

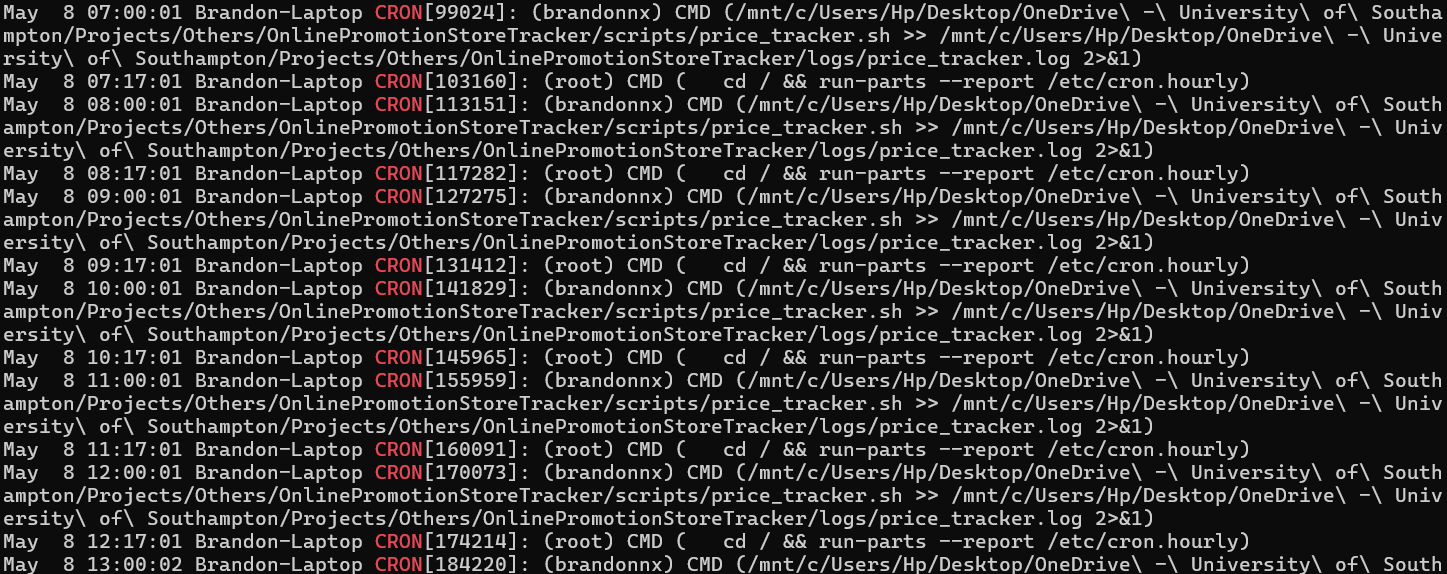
}

# 6.0 Contab Setup and Error Handling

## 6.1 Crontab Setup







## 6.2 Error Handling Examples

1. Error handling if data cannot be inserted into the database.

    error=$(mysql -u"$DB\_USER" -p"$DB\_PASS" -D"$DB\_NAME" -e "$query" 2>&1 > /dev/null)

    if [ ! -z "$error" ]; then

        echo "Error inserting into $table: $error"

    else

        echo "Data inserted successfully into $table."

    fi

1. Error handling if the price is not a valid number.

# Check if price is a valid number

if ! [[ $price =~ ^[0-9]+(\.[0-9]+)?$ ]]; then

echo "Error: Invalid price format for product: $product\_name"

return 1

fi

1. Error handling if number of stocks can't be fetched from specific product

# Parse stock

stock=$(awk 'BEGIN{RS="<"; FS=">"; IGNORECASE=1} /class="product\_qty\_availble"/ && !found {print $2; found=1}' "$file" | grep -oP '\d+' | head -n 1 | tr -d '\n')

if [ -z "$stock" ]; then

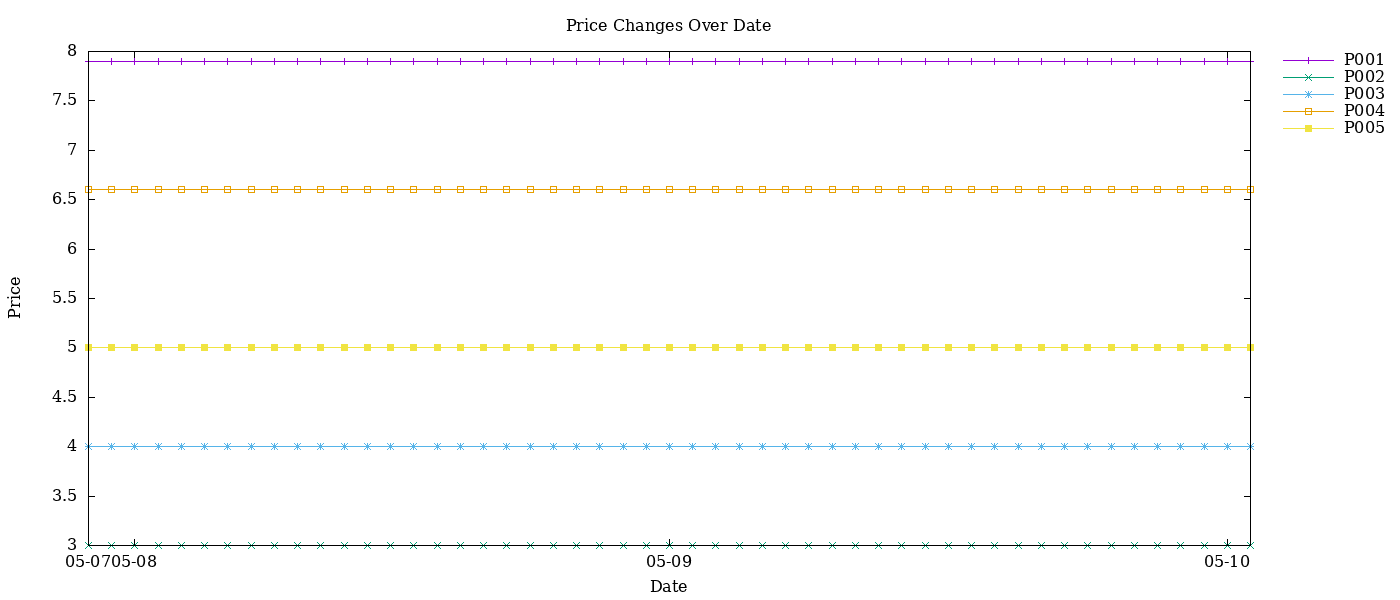
echo "Error: Stock not found for product: $product\_name"

return 1

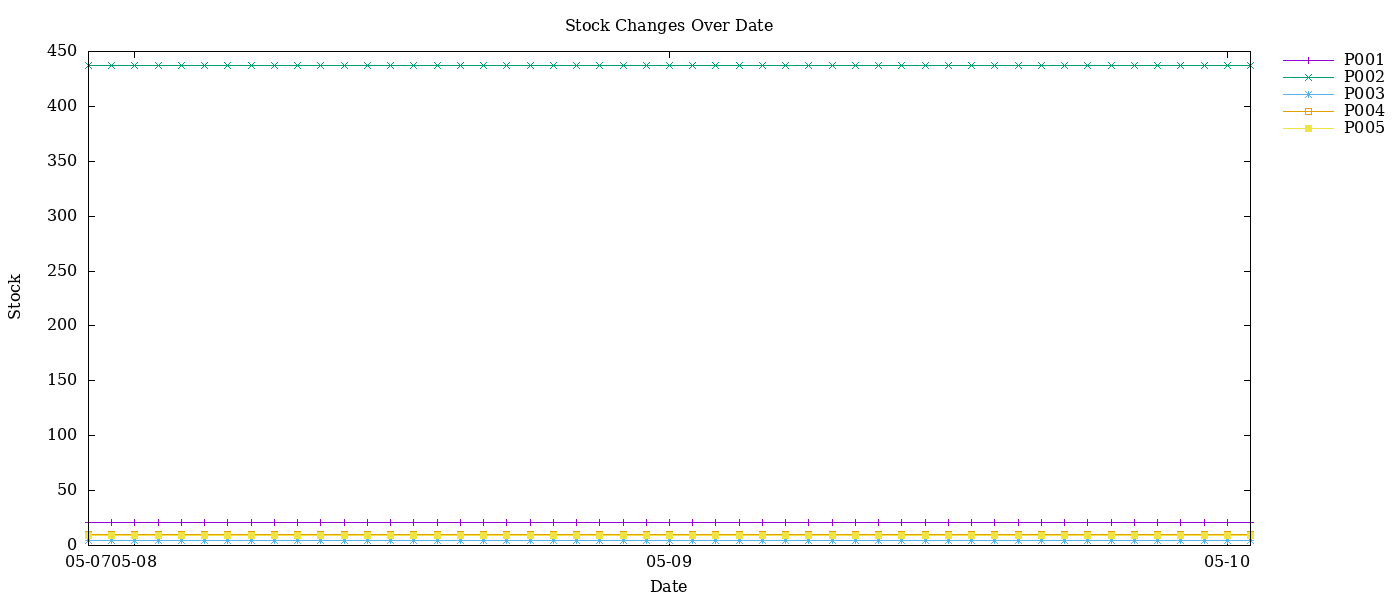
fi

# 7.0 Plotting Script (product\_details\_graph.sh)

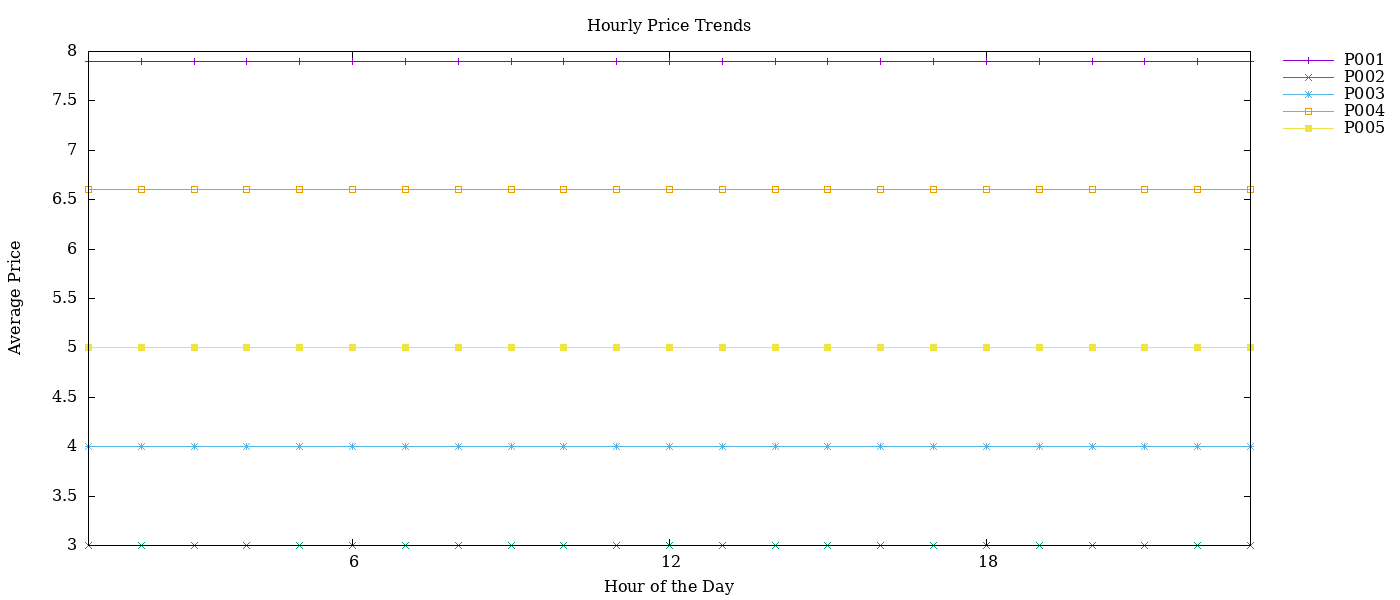
1. **Price Changes Over Date:**



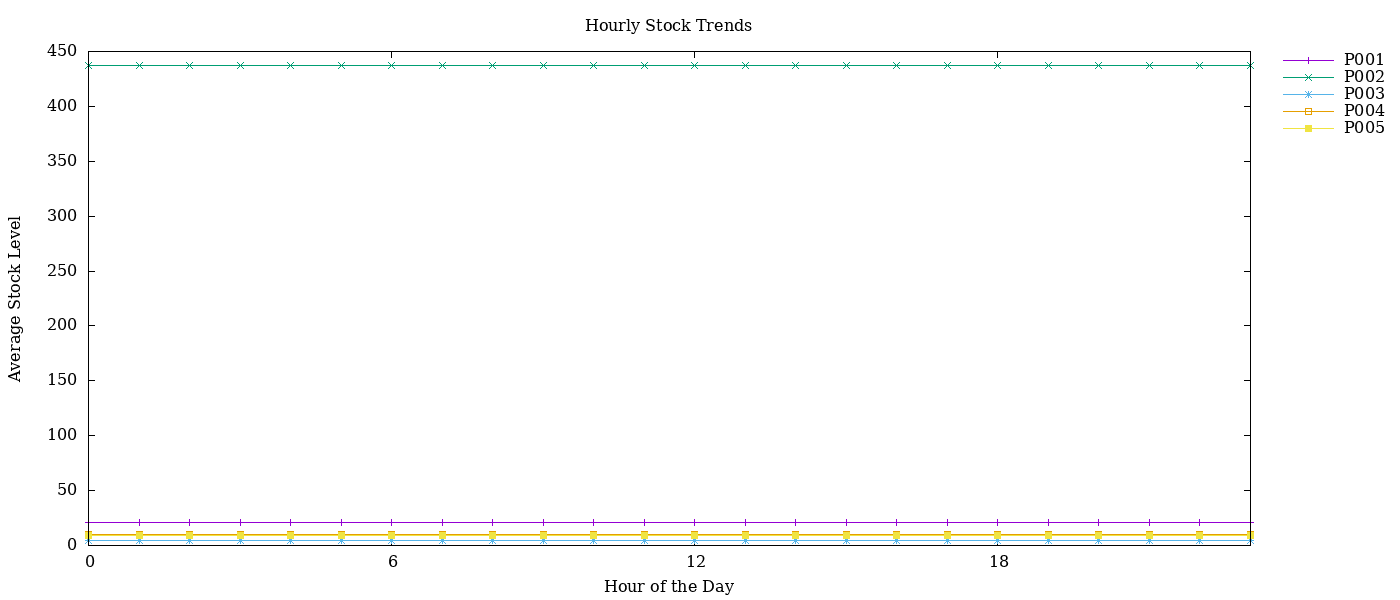
1. **Stock Changes Over Date:**



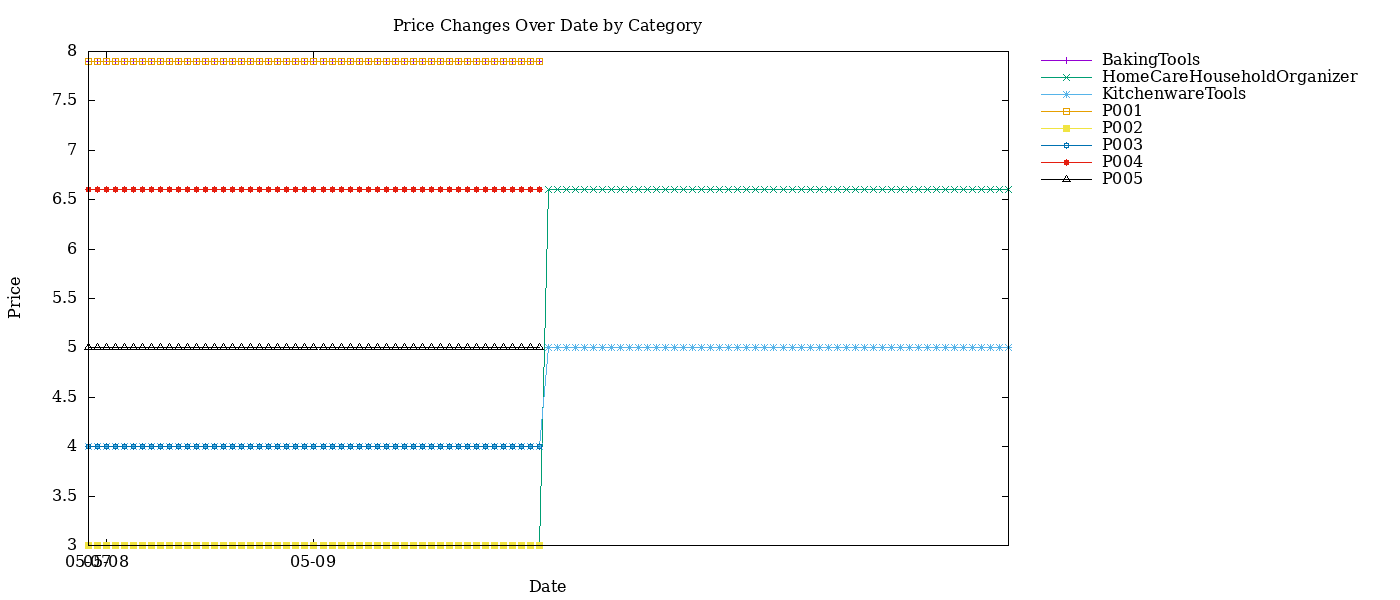
1. **Hourly Price Trends:**



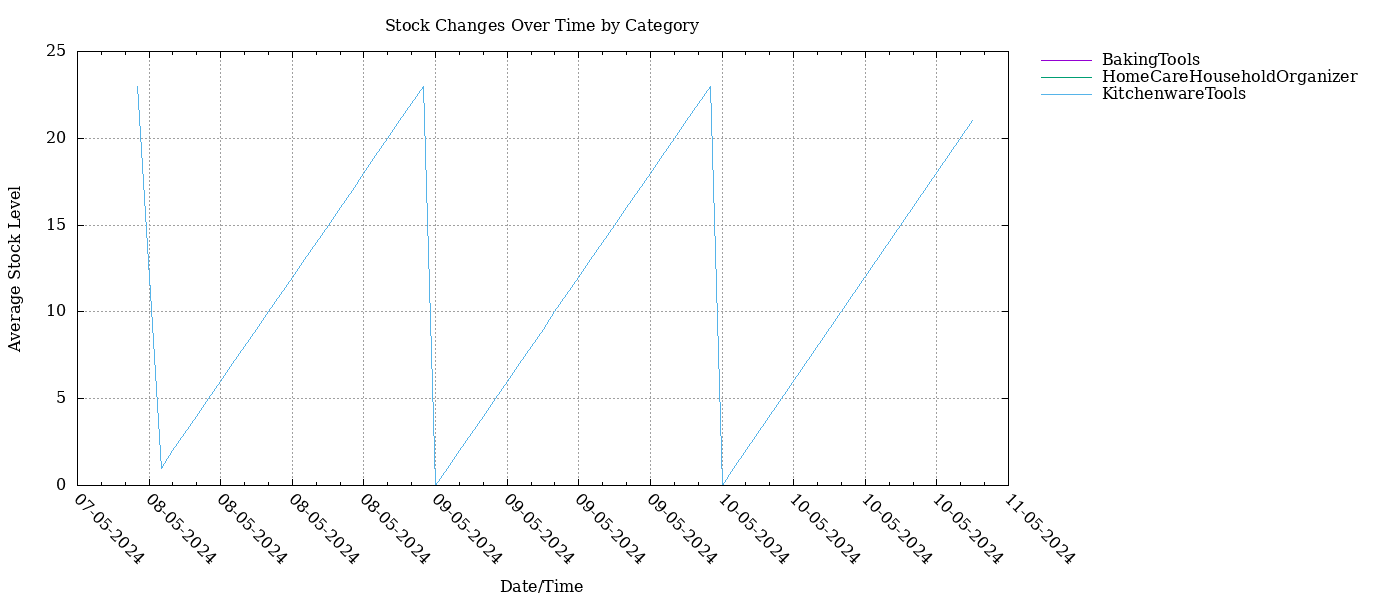
1. **Hourly Stock Trends:**



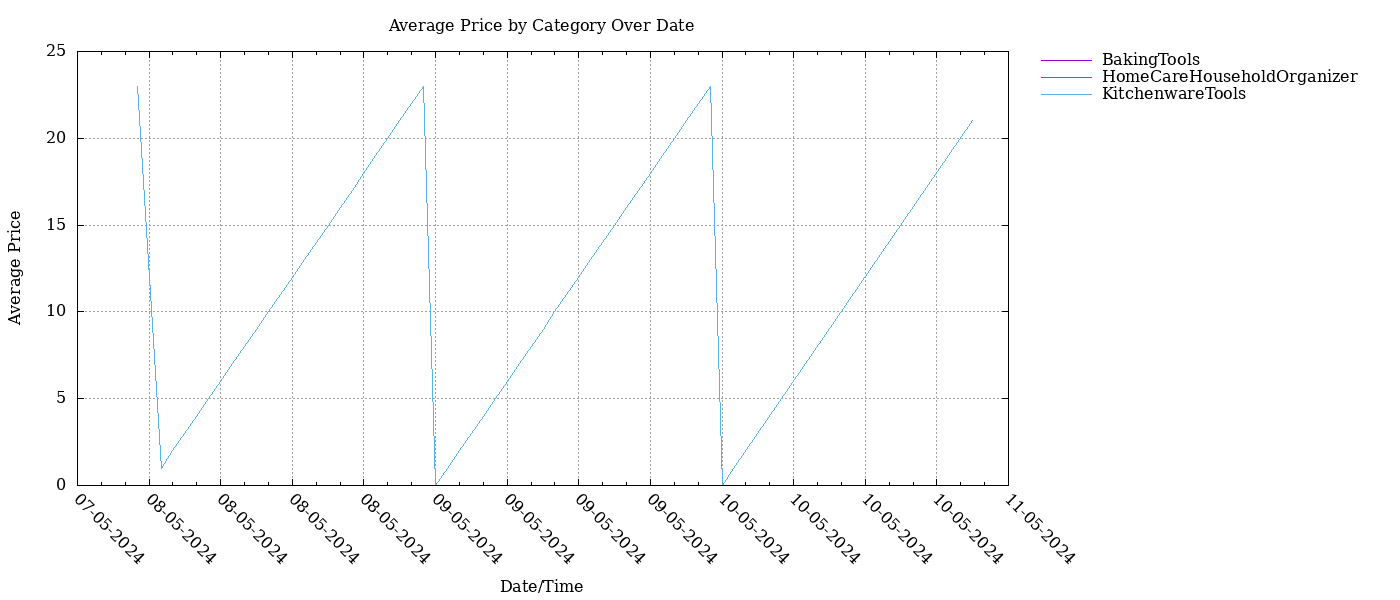
1. **Price Changes Over Date by Category:**



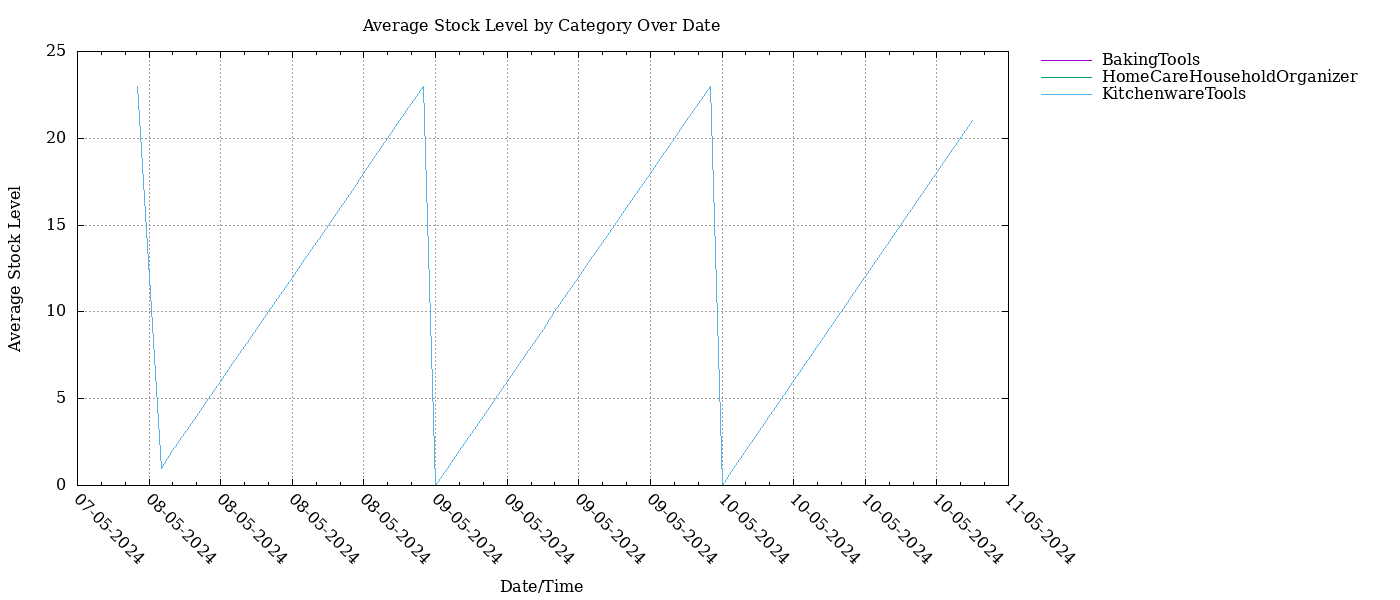
1. **Stock Changes Over Date by Category:**



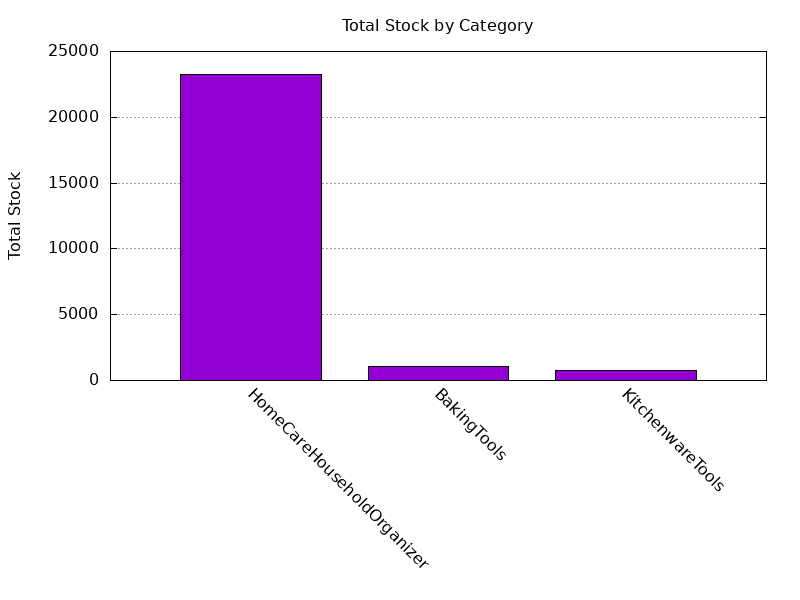
1. **Average Price by Category Over Date:**



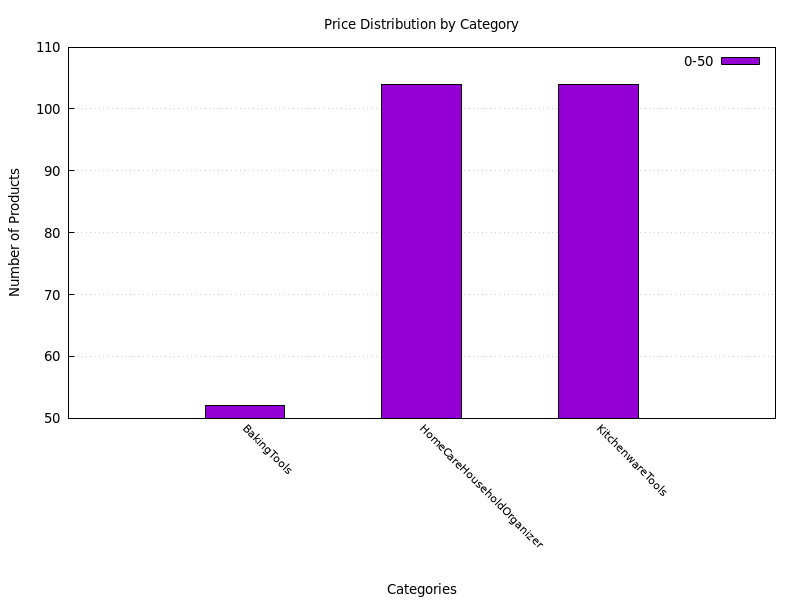
1. **Average Stock Level by Category Over Date:**



1. **Bar Chart of Total Stock by Category:**



1. **Bar Chart of Price Distribution by Category:**



# 8.0 Conclusion

To summarise, the online store promotion tracker developed for the Shoppu platform represents a significant development in the field of online shopping platforms. The tracker is an invaluable tool for consumers intending to make informed purchasing decisions because it consistently collects and analyses key data such as pricing, rating, stock availability, and the number of things sold. Throughout this project, we have demonstrated how automated tracking can provide consumers with fast and accurate information on product promotions and pricing patterns. By eliminating the need for manual data collection and analysis, the tracker simplifies consumer decision-making, allowing them to move quickly on opportunities to save money and take advantage of promotional offers. The project also further proved the reliability of Unix scripts for data collection and storage.

By using Unix scripts to collect data periodically and store it in a MySQL database, we ensure that users have access to the most current information when making purchasing decisions. Furthermore, the implementation of data visualisation techniques like graph plotting enables users to monitor trends pricing patterns and stock availability, allowing them to make intelligent decisions about when to buy, sell, or wait for great deals. Looking ahead, there are various chances for future improvements and expansions to the online store promotion tracker. This involves enhancing data collection techniques, providing support for new online tools, and including features like email notifications for price decreases and stock updates.

By continuing to innovate and improve the existing structure, we can increase the tracker's utility and efficiency in supporting consumers with their online shopping needs. In conclusion, the online store promotion tracker is a useful tool for consumers wishing to improve their online shopping experience by staying updated about special offers, pricing variations, and stock availability on the Shoppu platform. By providing customers with fast and accurate information, the tracker enables them to make better purchase decisions and save more money.

# 9.0 Appendices



fake\_product\_page.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Product Page</title>

</head>

<body>

    <div class="product-details">

        <span class="price" data-price="1300.50">RM1300.50898</span>

        <div class="stock-info" data-stock="20">20 units available.</div>

        <div class="ratings" data-rating="4.7">Rated 4.7 out of 5 stars</div>

    </div>

</body>

</html>

test\_parseDataAndDataManipulation.sh

#!/bin/bash

# Function to parse data

parseData() {

    local file="$1"

    local product\_name="$2"

    # Parse price from the new format

    price=$(grep -oP 'class="price" data-price="\K[\d.]+' "$file")

    # Parse stock from the new format

    stock=$(grep -oP 'class="stock-info" data-stock="\K[\d.]+' "$file")

    # Parse rating from the new format

    rating=$(grep -oP 'class="ratings" data-rating="\K[\d.]+' "$file")

    echo "Parsed Data: Price: RM$price, Stock: $stock units, Rating: $rating stars"

    dataManipulation "$product\_name" "$price" "$stock" "$rating"

}

# Function to manipulate data

dataManipulation() {

    local product\_name="$1"

    local price="$2"

    local stock="$3"

    local rating="$4"

    # Convert price to a float

    price=$(printf "%.2f" "$price")

    # Ensure stock is an integer

    stock=$(printf "%d" "$stock")

    # Convert rating to one decimal place

    rating=$(printf "%.1f" "$rating")

    echo "Manipulated Data: Product: $product\_name, Price: RM$price, Stock: $stock units, Rating: $rating stars"

}

# Testing the parseData function with the new HTML file format

echo "Testing parseData function..."

parseData "fake\_product\_page.html" "Smartphone Galaxy S22 Ultra"