

**INTEGRATED PROJECT REPORT**  
**ON**  
**SmartComp**

Submitted in partial fulfilment of the requirement for the  
Course Integrated Project (CS 203) of

**COMPUTER SCIENCE AND ENGINEERING**  
**B.E. BATCH-2019**  
**IN**  
**JUNE-2022**



**Under the guidance of:**

Dr. Deepak Ahlawat  
Assistant Professor | Department of CSE

**Submitted By:**

Aarchie Girdhar  
1910990235  
Paras Chaudhary  
1910991638  
Rashmi  
1910990273

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CHITKARA UNIVERSITY, PUNJAB**

**INTEGRATED PROJECT REPORT**  
**ON**  
**SmartComp**

Submitted in partial fulfilment of the requirement for the  
Course Integrated Project (CS 203) of

**COMPUTER SCIENCE AND ENGINEERING**  
**B.E. BATCH-2019**  
**IN**  
**JUNE-2022**



**Under the guidance of:**

Dr. Deepak Ahlawat  
Assistant Professor | Department of CSE

**Submitted By:**

Aarchie Girdhar  
1910990235  
Paras Chaudhary  
1910991638  
Rashmi  
1910990273

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CHITKARA UNIVERSITY, PUNJAB**

## **CERTIFICATE**

This is to be certified that the project entitled SmartComp has been submitted for the Bachelor of Computer Science Engineering at Chitkara University, Punjab during the academic semester January 2022- May-2022 is a bona fide piece of project work carried out by Aarchie Girdhar (1910990235), Paras Chaudhary (1910991638) and Rashmi (1910990273) towards the partial fulfilment for the award of the course Integrated Project (CS 203) under the guidance of Dr. Deepak Ahlawat and supervision.

**Sign. of Project Guide:**  
**Dr. Deepak Ahlawat**  
**(Associate Professor | CSE)**

## **CANDIDATE’S DECLARATION**

We, **AARCHIE GIRDHAR (1910990235), PARAS CHAUDHARY (1910991638) AND RASHMI (1910990273)** students of the **GROUP - 11**, B.E.-2019 of the Chitkara University, Punjab hereby declare that the Integrated Project Report entitled **SMARTCOMP** is an original work and data provided in the study is authentic to the best of our knowledge. This report has not been submitted to any other Institute for the award of any other course.

**Sign. of Archie Girdhar**

Aarchie Girdhar

ID No. 1910990235

**Sign. of Paras Chaudhary**

Paras Chaudhary

ID No. 1910991638

**Sign. of Rashmi**

Rashmi

ID No. 1910990273

**Place: Chitkara University, Punjab**

**Date: 27 June 2022**

## **ACKNOWLEDGEMENT**

It is our pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behaviour and acts during the course of study.

We express our sincere gratitude to all for providing me an opportunity to undergo Integrated Project as the part of the curriculum.

We are thankful to Dr. Deepak Ahlawat for his support, cooperation, and motivation provided to us during the training for constant inspiration, presence and blessings.

We also extend our sincere appreciation to *Dr. Deepak Ahlawat and Gaurav Goyal* who provided his valuable suggestions and precious time in accomplishing our Integrated project report.

Lastly, We would like to thank the almighty and our parents for their moral support and friends with whom we shared our day-to day experience and received lots of suggestions that improve our quality of work.

**Aarchie Girdhar**

**1910990235**

**Paras Chaudhary**

**19109991638**

**Rashmi**

**1910990273**

## INDEX

1.	Abstract / Keywords	7
2.	Introduction to the Project	8
2.1.	Background	8
2.2.	Problem Statement	9
3.	Software and Hardware Requirement Specification	10
3.1.	Methods	10
3.2.	Programming / Working Environment	12
3.3.	Requirements to run the Application	15
4.	Web Scraping Analysis and Implementation	17
5.	Program's Structure Analysis and GUI constructions	22
6.	Code-Implementation and Scripts Connections	26
7.	System Testing	32
8.	Limitations	35
9.	Conclusion	36
10.	Future Scope	37
11.	Bibliography / References	39

## **ABSTRACT/KEYWORDS**

“SmartComp” is a smart tool to compare and analyse prices and reviews of products across various e-commerce websites. Our application is built on Django – a python backend framework, implementing HTML, CSS, Javascript and JQuery on the front end. We have also used various python libraries for various purposes. Selenium was used by us to scrape the data off of ecommerce websites. Celery – task scheduling library and Redis – message broker were also used to make the tasks of scraping of data asynchronous to boost our application speed. Certain other libraries like bs4, lxml, re, pandas, etc, were also used for developmental and testing purposes.

## INTRODUCTION TO THE PROJECT

### Background:

Every shopper looks for the best deals & discounts before buying any product.

Nowadays before purchasing anything the buyers do some online research of the products on the internet. One of the major factors which lead to purchasing of any product is cost or pricing. The buyers tend to compare prices before purchasing any product.

But since it is very difficult to visit each & every website for price comparison, there needs to be a solution to automate this process. The Price comparison application - SmartComp proposed here gathers information dynamically on product prices from various websites & presents it to the users. The users can then choose to buy from the best options available. Even Ecommerce traders can use this price comparison website to study their competitors and form new strategies accordingly to attract new customers & stay ahead of their competitors.

This price comparison website for products will help to compare the price from various e-commerce websites, This Price comparison site is extremely helpful for frequent online shoppers to check prices on different online stores in one place.

This system will show you the product prices from different retailers to show you where to buy the product at affordable price, any two static websites classes are analysed to get the pricing details, to get the pricing details, the system visits the website based on user's search and downloads the html search page of that specific



website, once prices from both the websites are retrieved, it is displayed on our website in the form of price comparison.

## Problem Statement:

Users access the internet and get the information through different E-commerce websites. This approach to find and compare data from different websites is a time-consuming task.

To ease this process, we will enable the user to use our search engine which launches web crawler that dynamically crawls different e-commerce websites for deals on the searched product, which is presented in an easy to compare format along with the analysis of the reviews provided by the customers.

# **SOFTWARE AND HARDWARE REQUIREMENT SPECIFICATION**

## **Methods:**

### **Step - 1: Identifying the root cause of the problem**

The most initial step that we took to tackle the problem was identifying the root cause of its existence. During our research, we realized that the websites present out there to compare the deals across various ecommerce websites were few and far between; and the ones that were there were rather simple and lacked an informative and intuitive interface. As such, identification of key comparisons between deals could not be successfully done by the user.

Understanding this, we realized that our application required a brand-new interface that not only relayed all the important information needed to compare the deals, but also provided an intuitive and easy to use interface that would make our website highly dynamic and useful for the user.

### **Step - 2: Creating the design of our application**

Instead of jumping straight away into the development cycle, we first designed our application on Figma, deciding all major and minor features that need to be added along with their position in our application. While designing we also were able to

understand and anticipate the data flow in our application while development.

### Step - 3: Writing Web Scraping Scripts

With the expectations of our product set out, we then moved onto writing web scraping scripts to retrieve the key component of our application, data of deals from the ecommerce websites.

### Step - 4: Developing our application

On successful retrieval of data from ecommerce websites, the next obvious step was to create an application wherein the user could search the keywords/deals that they wish to compare. We implemented the designs we made in step-1.

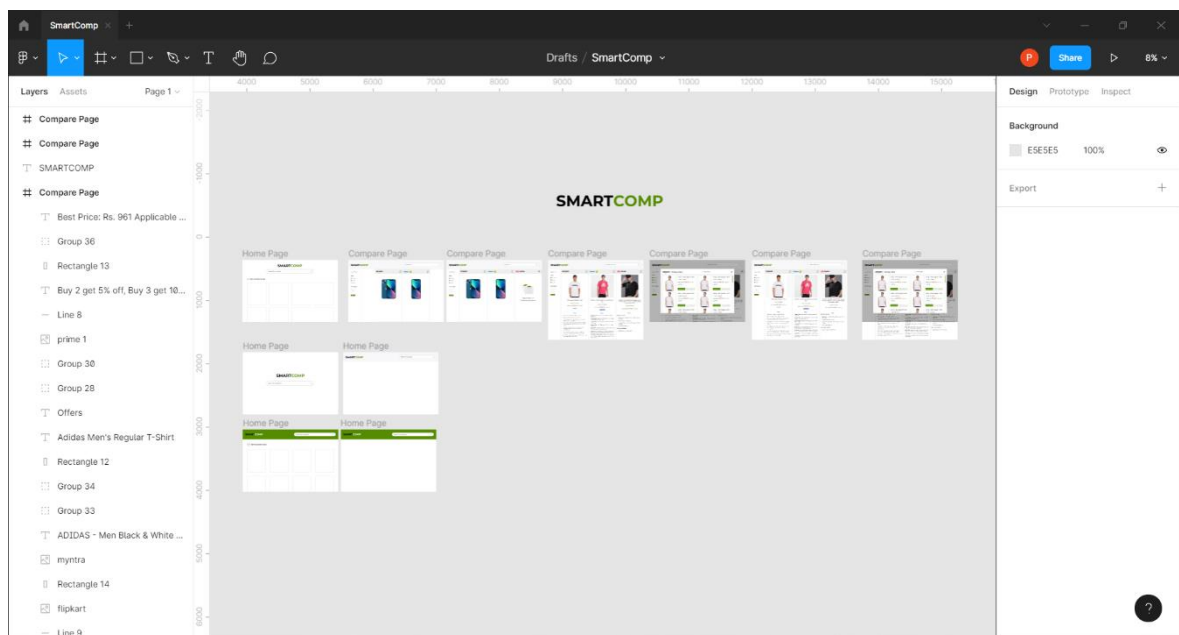
### Step - 5: Connecting our Scripts and our application

Since web scraping is a resource and time intensive task, we had to integrate our scripts with our application asynchronously to reduce the loading time of our website. Doing so required researching and finding other unique solutions for our final application.

## Programming / Working Environments:

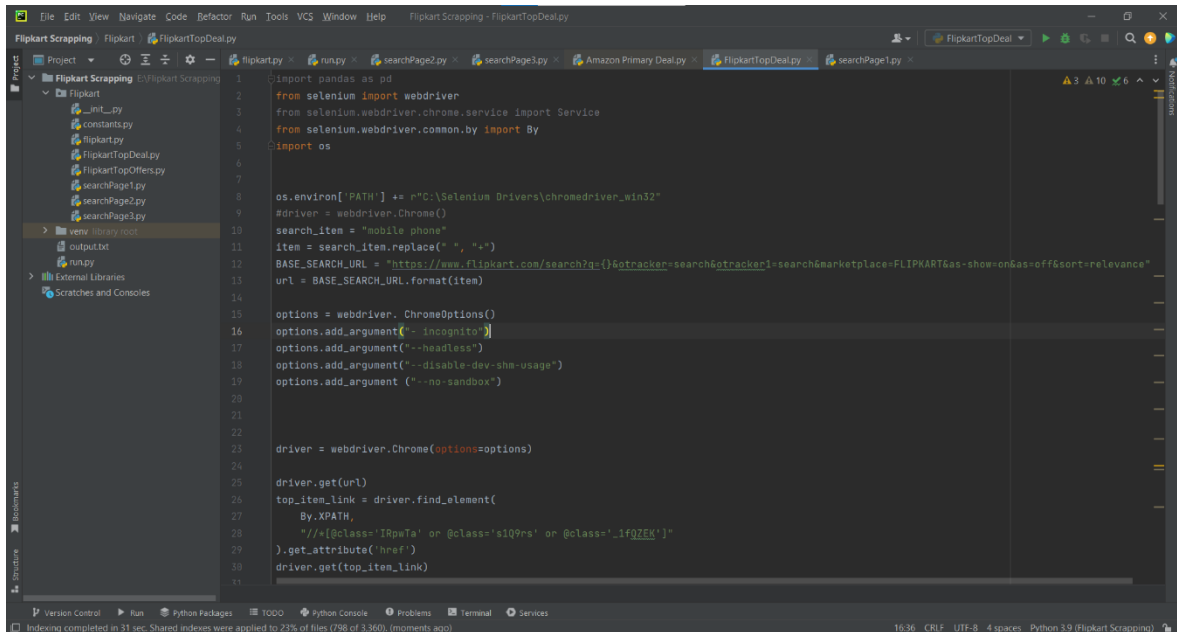
Following are the programming environments we have used to test and build this project:

### 1. FIGMA:



As the first and foremost step, we have created the designs of our application using Figma as per the requirements of the project. Designs were made keeping in mind our future scopes for the project as well.

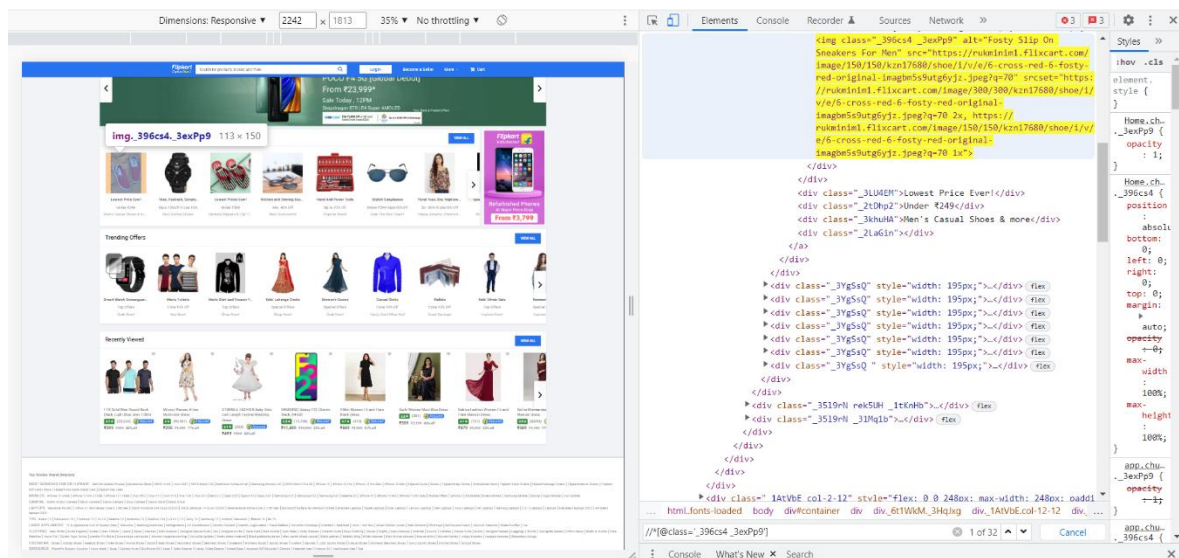
## 2. PYCHARM:



Data mining have been done by using python libraries – Selenium and testing have been done using python library – Beautiful Soup.

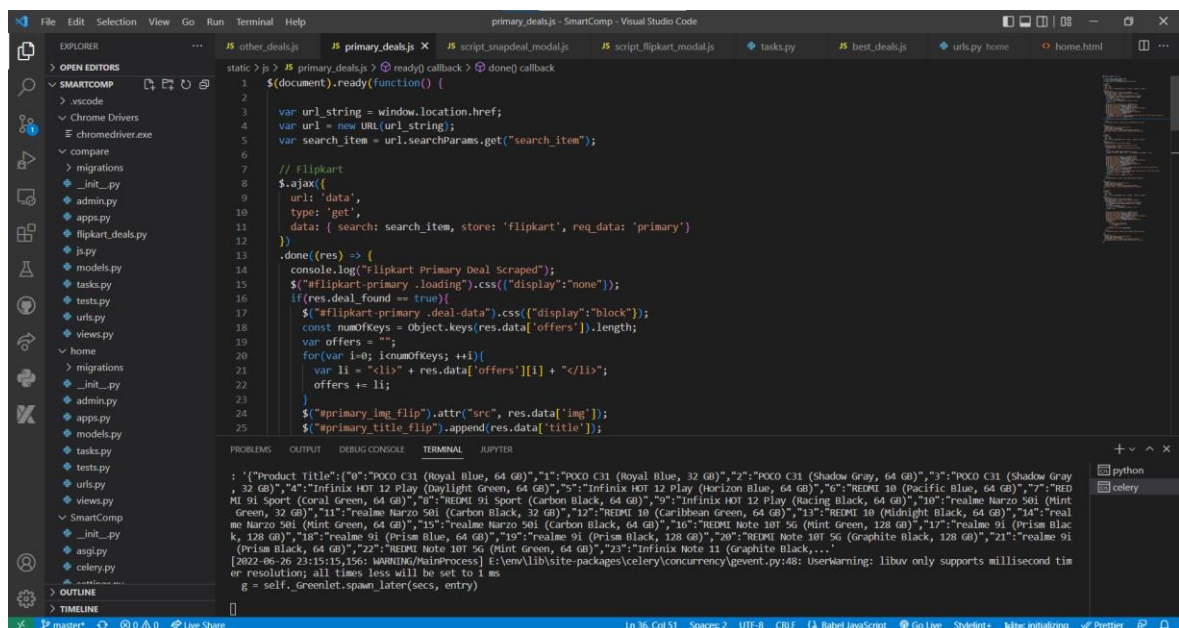
In order to test the retrieving of all the required data had been done on PyCharm (a python IDE)

## 3. BROWSER:



XPath in Selenium (XML path) have been used for navigation through the HTML structure of the page and located elements for data mining in Selenium web drivers.

#### 4. VISUAL STUDIO CODE EDITOR:



In order to integrate the retrieved data and to display it over the Django application,

and making it work along with headless chrome driver VS Code Editor have been used.

## Requirements To Run the Application:

In order to run the application, following is the requirements.txt file for our project.

### requirements.txt:

```

requirements.txt
1  aiohttp==3.8.1
2  aiosignal==1.2.0
3  amqp==5.1.1
4  asgiref==3.5.2
5  async-generator==1.10
6  async-timeout==4.0.2
7  attrs==21.4.0
8  beautifulsoup4==4.11.1
9  billiard==3.6.4.0
10 bs4==0.0.1
11 celery==5.2.7
12 certifi==2022.6.15
13 cffi==1.15.0
14 charset-normalizer==2.0.12
15 click==8.1.3
16 click-didyoumean==0.3.0
17 click-plugins==1.1.1
18 click-repl==0.2.0
19 colorama==0.4.5
20 cryptography==37.0.2

```

```

requirements.txt
21 Deprecated==1.2.13
22 Django==4.0.5
23 django-celery-results==2.3.1
24 frozenlist==1.3.0
25 gevent==21.12.0
26 greenlet==1.1.2
27 h11==0.13.0
28 idna==3.3
29 kombu==5.2.4
30 loader==2017.9.11
31 Logbook==1.5.3
32 multidict==6.0.2
33 outcome==1.2.0
34 packaging==21.3
35 pendulum==2.1.2
36 prompt-toolkit==3.0.29
37 pycparser==2.21
38 pyOpenSSL==22.0.0
39 pyparsing==3.0.9
40 PySocks==1.7.1

```

Create a virtual environment.

Run the following command inside the virtual environment in order to install all the packages needed.

```
pip install -r requirements.txt
```

Turn on the celery and redis servers.

Run the Django server in order to run the application.



## WEB SCRAPPING ANALYSIS AND IMPLEMENTATION

Because of the nature of our problem, we did not need to implement a database in our application. Nonetheless, data is at the core of our application and is instead scraped dynamically when the user enters their search keywords.

This ensures that the user receives the latest updated data from their desired ecommerce websites.

Selenium is the most important library that has been used for web scraping in our application. Although, certain other libraries like bs4, lxml, re, pandas, etc, were also used for developmental and testing purposes.

The scripts for web scraping were first implemented in procedural coding style in separate scripts and later integrated into our application after fail proof retrieval of data from the various ecommerce websites. Before looking into the outputs of from these scripts, first it is important to know all the different types of data that is being retrieved from each ecommerce website.

According to our application design, we have two pages - Home Page, which displays the best deals from the 3 ecommerce websites that are Amazon, Flipkart and Snapdeal. The second page is the Compare page, which is where the user is able to compare the deals that they wish to. To fulfil the requirements of Compare Page, 2 sets of data are required – data of the deal that the user wishes to compare (Primary Deal) and other similar deals that are available according to the searched keyword (Other Deals). We will see ahead the webpage (Amazon for example) and the script scraping it for side-by-side analysis.



## Amazon Other Deals Scrapped:

```

1 from selenium import webdriver
2 from selenium.webdriver.chrome.service import Service
3 from selenium.webdriver.common.by import By
4 from webdriver_manager.chrome import ChromeDriverManager
5 from bs4 import BeautifulSoup

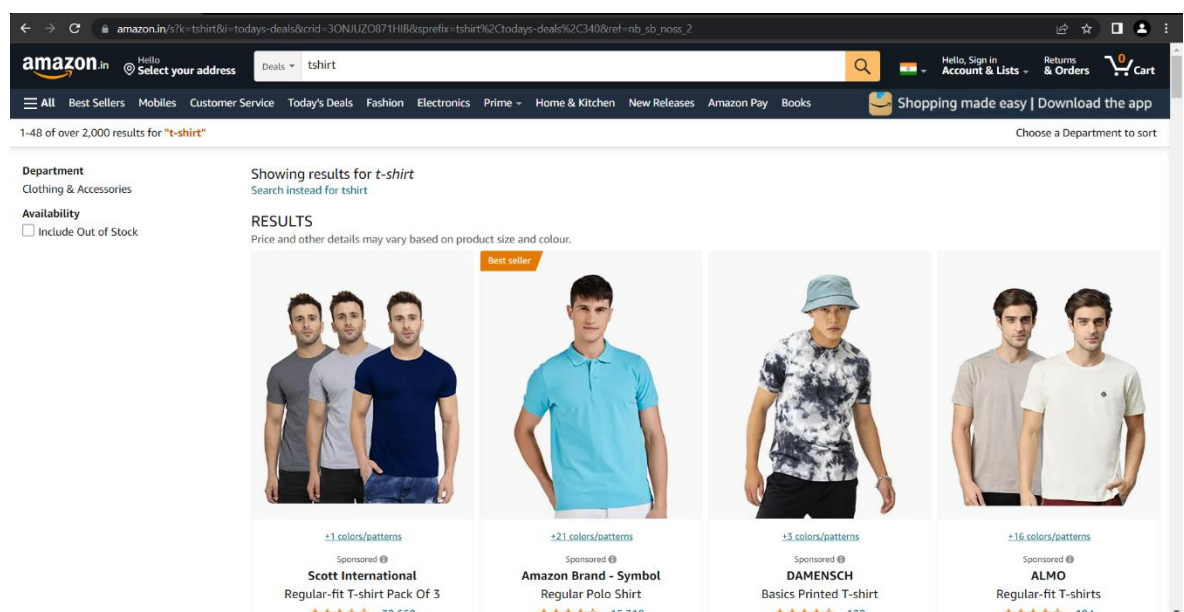
['Scott International', 'HINGE', 'Generic', 'Pangolin', 'Ben Martin', 'WDIRARA', 'BLIVE', 'DAMENSCH', 'THE BLAZZE', 'RodZen', 'LEOTUDE', 'Scott Intern
['https://m.media-amazon.com/images/I/71vp8Lec9JL.AC_UL320_.jpg', 'https://m.media-amazon.com/images/I/71wBkkT6z6L.AC_UL320_.jpg', 'https://m.media-
['₹2,397', '₹1,299', '₹1,299', '₹1,500', '₹1,999', '₹14,697', '₹1,999', '₹599', '₹999', '₹1,499', '₹1,099', '₹2,397', '₹1,099', '₹1,099', '₹1,099', '₹
['₹471', '₹799', '₹335', '₹1,425', '₹299', '₹4,397', '₹251', '₹395', '₹599', '₹349', '₹299', '₹471', '₹420', '₹319', '₹324', '₹399', '₹263', '₹369', '
['(80% off)', '(38% off)', '(74% off)', '(5% off)', '(85% off)', '(70% off)', '(87% off)', '(34% off)', '(40% off)', '(77% off)', '(73% off)', '(80% o
['3.7 out of 5 stars', None, '3.2 out of 5 stars', None, '4.5 out of 5 stars', '4.2 out of 5 stars', '3.1 out of 5 stars', '4.3 out of 5 stars', '3.2
['30,660', None, '3', None, '69', '13', '132', '186', '737', '4', '30,660', '3,006', '15', '548', '5', '5,100', '334', '692', '49', '120', '407
[False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
[False, False, False, False, False, False, True, False, False, False, False, False, False, False, False, False, False, False, True, False, False
['https://www.amazon.in/gp/s/redirect/picassoRedirect.html/ref=pa_sp_atf_aps_sr_pg1_1?ie=UTF8&adId=A0443853MEL6NH682WDG&url=%2FScott-International-Reg

Converting to DataFrame and printing it.

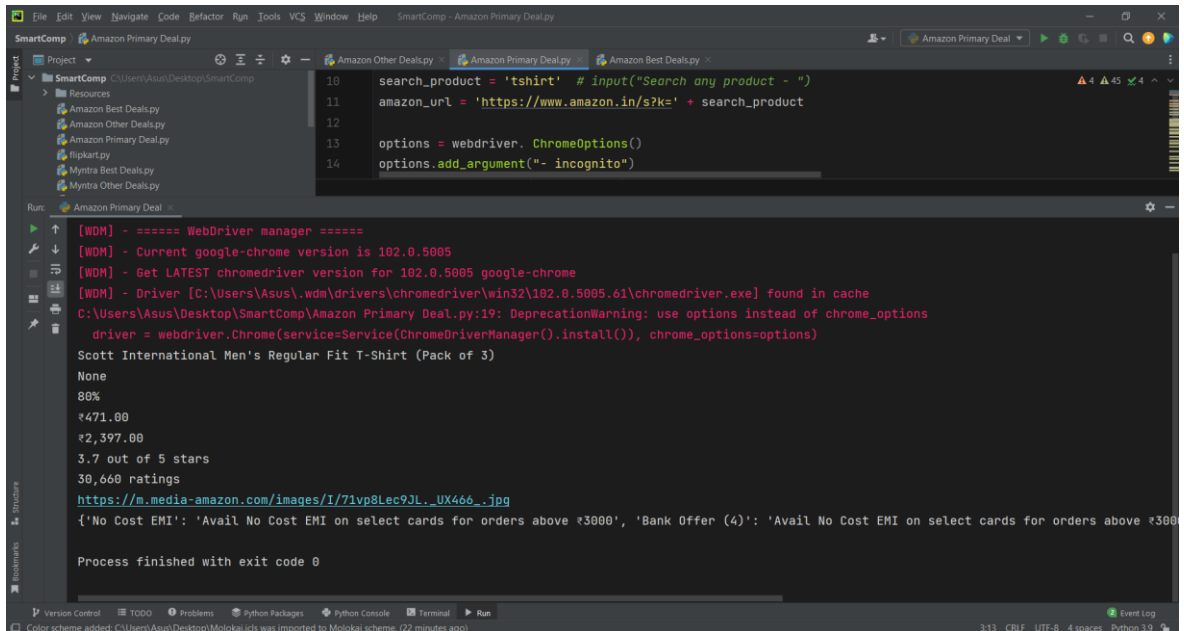
Process finished with exit code 0

```

## Amazon Other Deals Webpage



## Amazon Primary Deal Script



```

10 search_product = 'tshirt' # input("Search any product - ")
11 amazon_url = 'https://www.amazon.in/s?k=' + search_product
12
13 options = webdriver.ChromeOptions()
14 options.add_argument("- incognito")

```

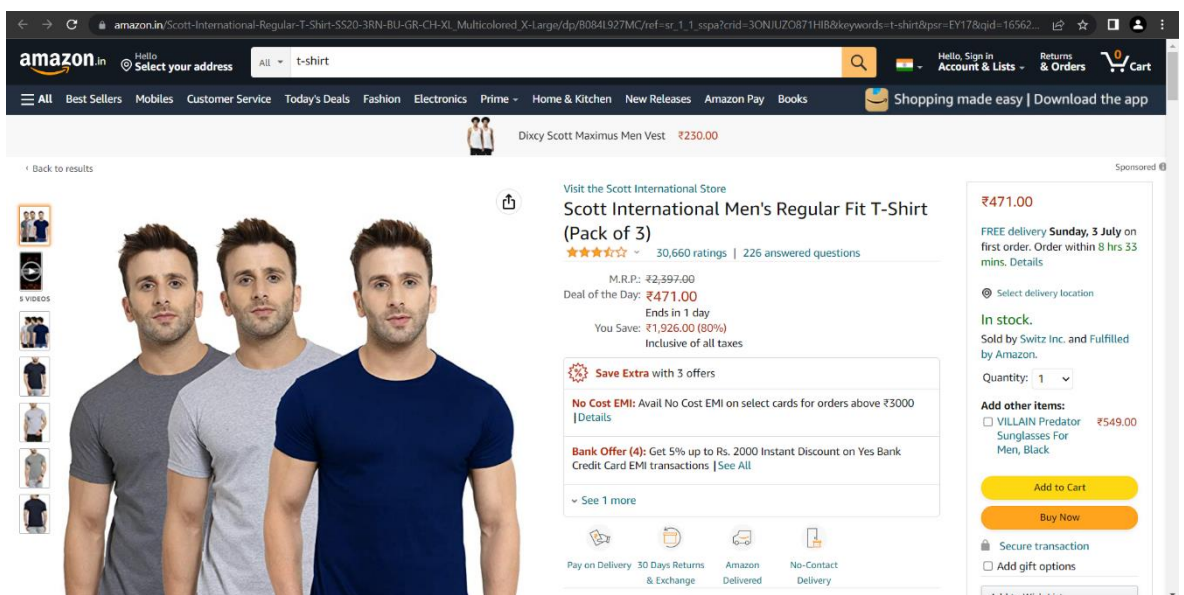
Run: Amazon Primary Deal

```

[WDM] - ===== WebDriver manager =====
[WDM] - Current google-chrome version is 102.0.5005
[WDM] - Get LATEST chromedriver version for 102.0.5005 google-chrome
[WDM] - Driver [C:\Users\Asus\wdm\drivers\chromedriver\win32\102.0.5005.61\chromedriver.exe] found in cache
C:\Users\Asus\Desktop\SmartComp\Amazon Primary Deal.py:19: DeprecationWarning: use options instead of chrome_options
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), chrome_options=options)
Scott International Men's Regular Fit T-Shirt (Pack of 3)
None
80%
₹471.00
₹2,397.00
3.7 out of 5 stars
30,660 ratings
https://m.media-amazon.com/images/I/71vp8Lec9JL_UX466_.jpg
{'No Cost EMI': 'Avail No Cost EMI on select cards for orders above ₹3000', 'Bank Offer (4)': 'Avail No Cost EMI on select cards for orders above ₹3000'}
Process finished with exit code 0

```

## Amazon Primary Deals Webpage

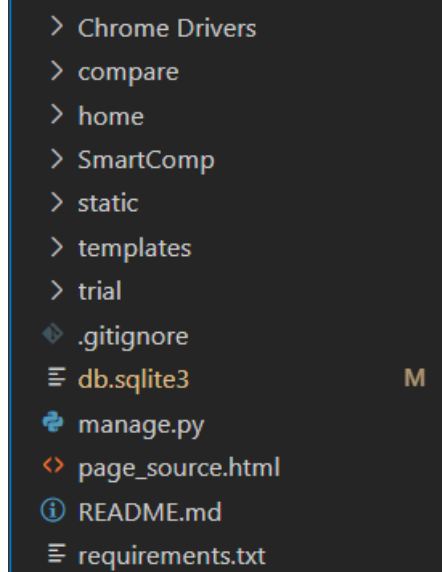


Similar scripts were also made for the other two ecommerce websites that have been included for comparing in our application - Flipkart and Snapdeal.

## PROGRAM STRUCTURE ANALYSIS AND GUI CONSTRUCTION

Following are some screenshots of the application structure we used for our Django application.

### File Structure



### Installed\_apps

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'compare',
    'home',
    'django_celery_results',
]
```

## urls.py

```

16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('', include('home.urls')),
21     path('compare/', include('compare.urls')),
22     path('trial/', include('trial.urls')),
23     path('admin/', admin.site.urls),
24 ]
25

```

```

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-.\env\Scripts\activate: https://aka.ms/pscore6
>> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 27, 2022 - 05:45:32
Django version 4.0.5, using settings 'SmartComp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

```

## Settings for static files such as CSS, JS, Images

```

STATIC_URL = 'static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
STATIC_ROOT = os.path.join(BASE_DIR, 'assets')

```

## Settings for template files created using HTML, CSS, Bootstrap

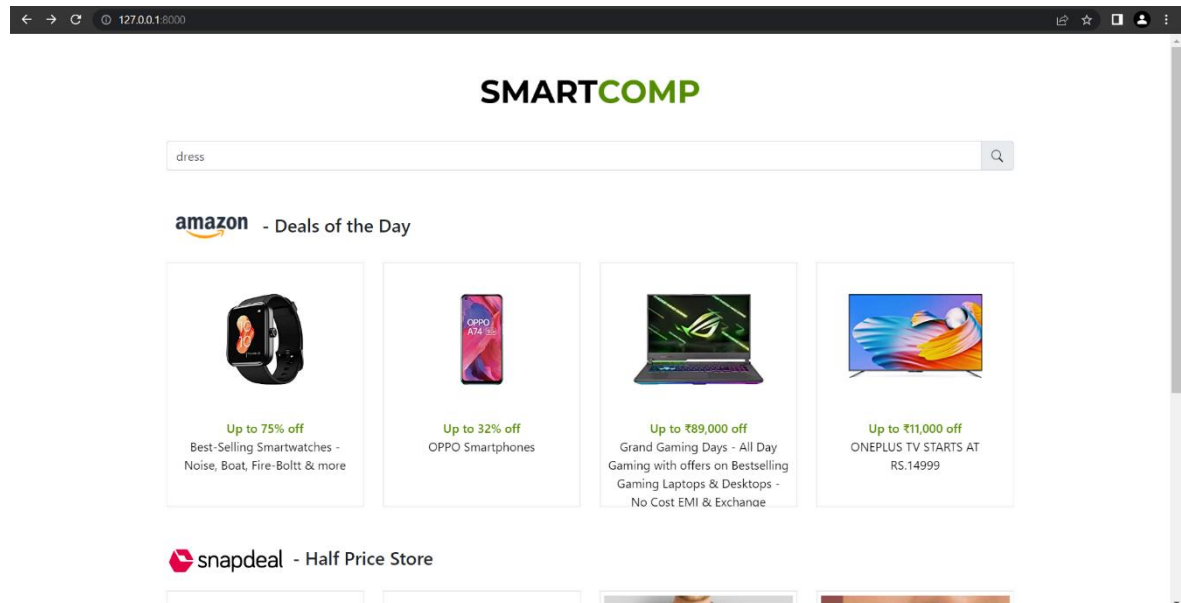
```

55 ROOT_URLCONF = 'SmartComp.urls'
56
57
58 TEMPLATES = [
59     {
60         'BACKEND': 'django.template.backends.django.DjangoTemplates',
61         'DIRS': [os.path.join(BASE_DIR, 'templates')],
62         'APP_DIRS': True,
63         'OPTIONS': {
64             'context_processors': [
65                 'django.template.context_processors.debug',
66                 'django.template.context_processors.request',
67                 'django.contrib.auth.context_processors.auth',
68                 'django.contrib.messages.context_processors.messages',
69             ],
70         },
71     ],
72 ]
73

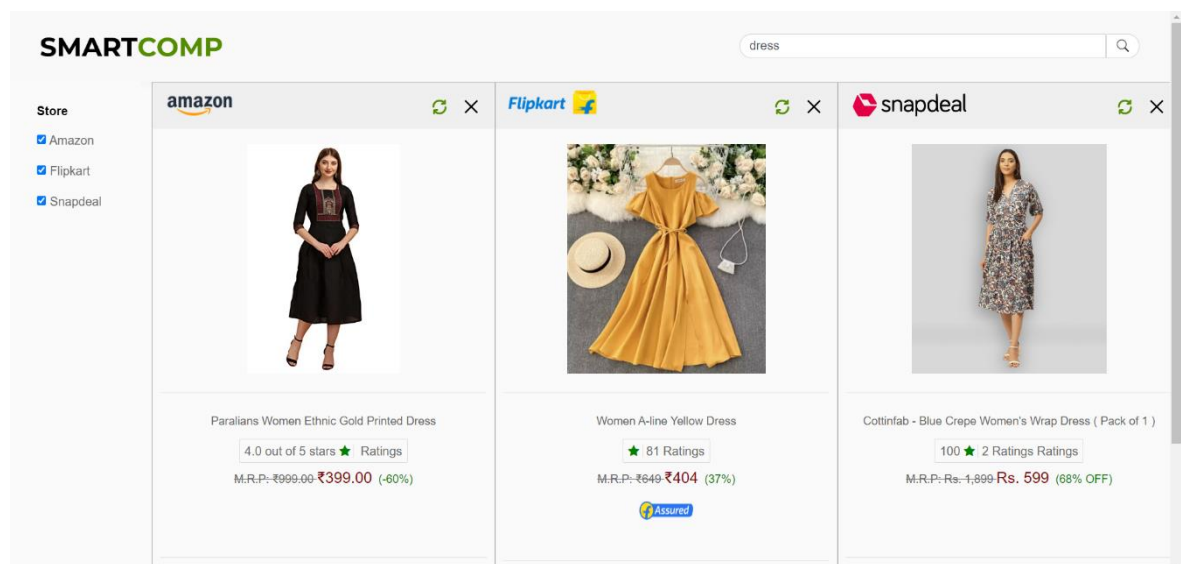
```

The front end of our application has been built using HTML, CSS, Javascript, JQuery and Bootstrap. Following are the images of our web application in various states.

## Home page (Best deals from Amazon, Flipkart and Snapdeal)



## Primary deals



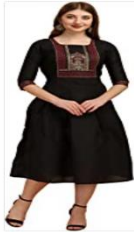


## Other deals

**SMARTCOMP** dress Q

Store  
☒ Amazon  
☒ Flipkart  
☒ Snapdeal

**amazon** - Change Deal Search the URL X




Women Ethnic Gold Printed Dress

4.0 out of 5 stars ★ 1 Ratings

M.R.P.: ₹999 **₹399** ((60% off))

[Compare Deal](#) [View on Amazon](#)




Western Dresses for Women|Stylish Latest Dresses|Skirts|Kurta with Palazzo Set|Long Kurtis|Stylish Tops|Western Tops for Girls|Gown|Maxi Dress Crop top|Party Dress

4.0 out of 5 stars ★ 493 Ratings


M.R.P.: ₹2,338 **₹568** ((76% off))

**prime**

[Compare Deal](#) [View on Amazon](#)



Multi Colored Floral Aline Printed Dress




Women's Maxi Dress.

**SMARTCOMP** dress Q

Store  
☒ Amazon  
☐ Flipkart  
☒ Snapdeal

**amazon** Q X




Paralians Women Ethnic Gold Printed Dress

4.0 out of 5 stars ★ Ratings

M.R.P.: ₹999.00 **₹399.00** (~60%)

**snapdeal** Q X



Cottinfab - Blue Crepe Women's Wrap Dress ( Pack of 1 )

100 ★ 2 Ratings Ratings

M.R.P.: Rs. 1,899 **Rs. 599** (68% OFF)

## CODE-IMPLEMENTATION AND SCRIPTS CONNECTIONS

As we discussed before, web scraping is a time and resource intensive and doing so one at a time is not really feasible as increased waiting time lowers user retention rate. As such, we had to find a way to perform our various web scraping tasks asynchronously while building measures to ensure scalability in the future. Thus, we decided to use Celery - an open-source asynchronous task queue or job queue which is based on distributed message passing. As a requirement of Celery, we also had to use Redis - an in-memory data structure store, used as a distributed, in-memory key-value database, cache and message broker, with optional durability.

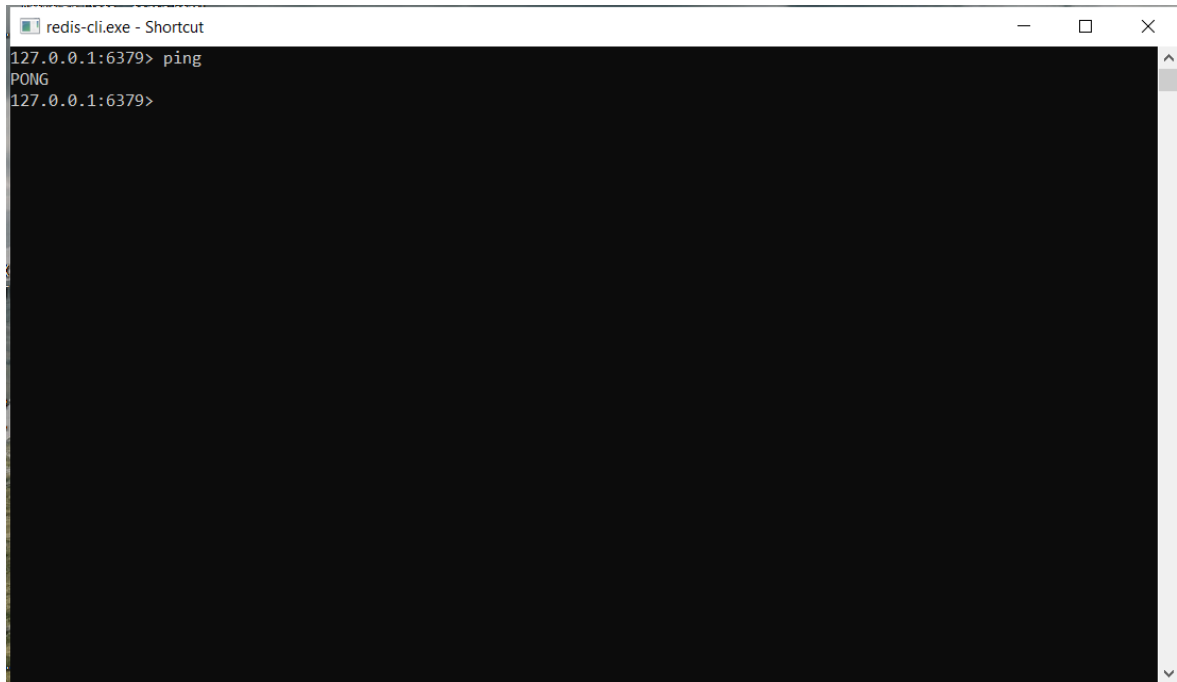
We divided our web scraping scripts into different tasks in celery that could be performed simultaneously when needed. Using a special view and a new url path calling it, we provided a url for our frontend architecture to make API calls to using Ajax.

Ahead, you will be viewing the images of code of some of the key components and their settings that help provide a continuous connection between our website's frontend and backend.

### Celery settings in Django application

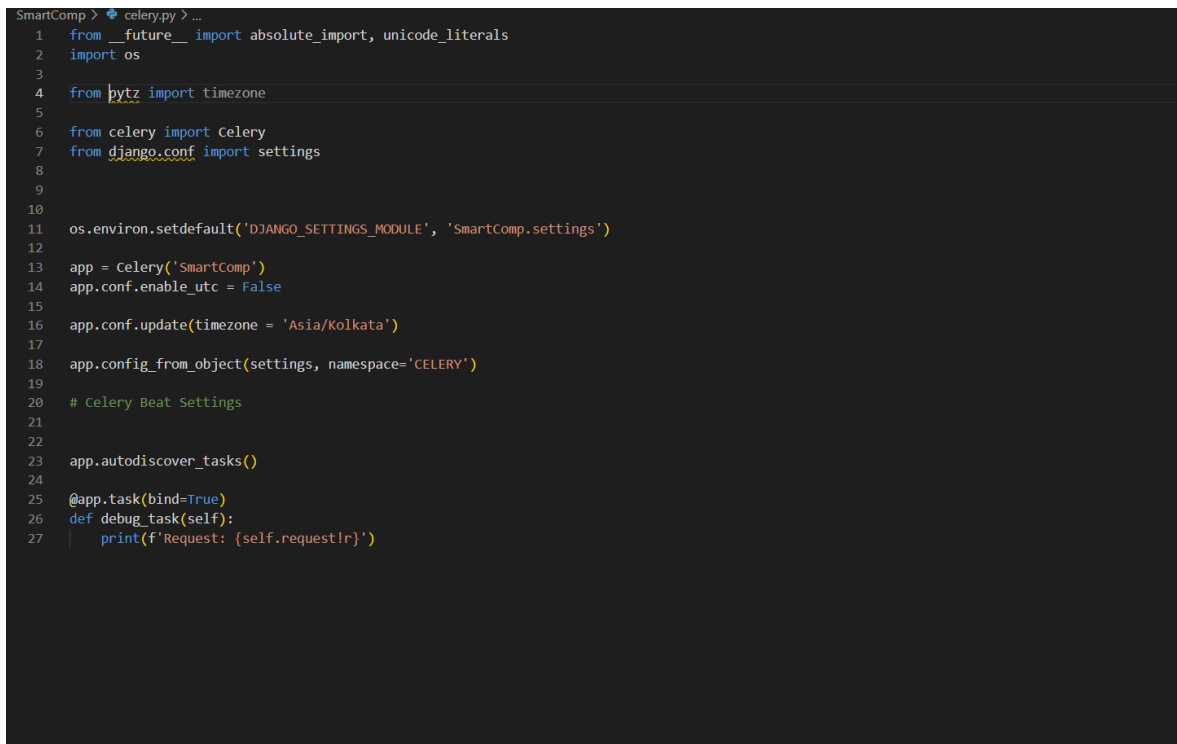
```
138 # CELERY SETTINGS
139
140 CELERY_BROKER_URL = 'redis://127.0.0.1:6379'
141 CELERY_ACCEPT_CONTENT = ['application/json']
142 CELERY_RESULT_SERIALIZER = 'json'
143 CELERY_TASK_SERIALIZER = 'json'
144 CELERY_TIMEZONE = 'Asia/Kolkata'
145
146 CELERY_RESULT_BACKEND = 'django-db'
```

## Verifying redis connection availability



```
redis-cli.exe - Shortcut
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

## Celery.py



```
SmartComp > celery.py > ...
1 from __future__ import absolute_import, unicode_literals
2 import os
3
4 from pytz import timezone
5
6 from celery import Celery
7 from django.conf import settings
8
9
10
11 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'SmartComp.settings')
12
13 app = Celery('SmartComp')
14 app.conf.enable_utc = False
15
16 app.conf.update(timezone = 'Asia/Kolkata')
17
18 app.config_from_object(settings, namespace='CELERY')
19
20 # Celery Beat Settings
21
22
23 app.autodiscover_tasks()
24
25 @app.task(bind=True)
26 def debug_task(self):
27     print(f'Request: {self.request!r}')
```

## Best deals scrapping tasks

```
home > tasks.py > ...
1 import time
2 import pandas as pd
3 from celery import shared_task
4 from selenium.common.exceptions import WebDriverException
5 from selenium import webdriver
6 from selenium.webdriver.common.by import By
7
8 options = webdriver.ChromeOptions()
9 options.add_argument("--incognito")
10 options.add_argument("--headless")
11 options.add_argument("--disable-dev-shm-usage")
12 options.add_argument("--no-sandbox")
13
14 @shared_task(bind=True)
15 def amazon_best_deals(self):
16
17     driver = webdriver.Chrome(executable_path=r'E:\SmartComp\Chrome Drivers\chromedriver.exe', options=options)
18     driver.maximize_window()
19
20     driver.get('https://www.amazon.in/gp/goldbox')
21
22     all_deals = driver.find_elements(by=By.XPATH, value='//div[@aria-label="Deals grid"]//div[@data-testid="deal-card"]')
23     if len(all_deals) > 15:
24         all_deals = all_deals[0:15]
25
26     title = []
27     img = []
28     discount = []
29     deal_url = []
30     for deal in all_deals:
31         try:
32             deal_url.append(deal.find_element(by=By.XPATH, value='//a').get_attribute('href'))
33         except:
34             continue
35         try:
36             img.append(deal.find_element(by=By.XPATH, value='//img').get_attribute('src'))
37         except:
```

## Other deals and primary deals scrapping tasks

```
compare > tasks.py > ...
1 import re
2 from celery import shared_task
3 from selenium import webdriver
4 from selenium.webdriver.common.by import By
5 import pandas as pd
6 from bs4 import BeautifulSoup
7
8 options = webdriver.ChromeOptions()
9 options.add_argument("--incognito")
10 options.add_argument("--headless")
11 options.add_argument("--disable-dev-shm-usage")
12 options.add_argument("--no-sandbox")
13
14 #driver = webdriver.Chrome(executable_path=r'E:\SmartComp\Chrome Drivers\chromedriver.exe', options=options)
15
16 @shared_task(bind=True)
17 def get_search_url(self, store, search):
18     if store == 'flipkart':
19         item = search.replace(" ", "+")
20         BASE_SEARCH_URL = "https://www.flipkart.com/search?q={}&otracker=search&otracker1=search&marketplace=FLIPKART&as-show=on&as-off=off&sort="
21         url = BASE_SEARCH_URL.format(item)
22         return url
23     elif store == 'amazon':
24         item = search.strip()
25         url = 'https://www.amazon.in/s?k=' + item
26         return url
27     elif store == 'snapdeal':
28         item = search.strip()
29         url = 'https://www.snapdeal.com/search?keyword=' + item
30         return url
31
32 @shared_task(bind=True)
33 def flipkart_primary_deal(self, search):
34     driver = webdriver.Chrome(executable_path=r'E:\SmartComp\Chrome Drivers\chromedriver.exe', options=options)
35     url = get_search_url('flipkart', search)
36     driver.get(url)
37
```

## Data url path

```
urls.py x
home > urls.py > [e] urlpatterns
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.home, name='home'),
6      path('data', views.AjaxHandlerView.as_view())
7  ]
```

## Ajax handler view

```
12  class AjaxHandlerView(View):
13      def get(self, request):
14          store = request.GET.get('store')
15          req_data = request.GET.get('req_data')
16
17          if req_data == 'best-deals':
18              if store == 'flipkart':
19                  data = flipkart_best_deals.delay()
20              elif store == 'amazon':
21                  data = amazon_best_deals.delay()
22              elif store == 'snapdeal':
23                  data = snapdeal_best_deals.delay()
24
25          return JsonResponse({'data': data.get()}, status=200)
26
27
```

## Sample ajax call

```

JS best_deals.js X
static > js > JS best_deals.js > ready() callback > done() callback
1  $(document).ready(function(){
2      // Amazon Scrapping
3      $.ajax({
4          url: 'data',
5          type: 'get',
6          data: { req_data: 'best-deals', store: 'amazon'}
7      })
8      .done((res) => {
9          $("#loading").remove()
10         console.log('Amazon Scraped')
11         var data = JSON.parse(res.data);
12         console.log(data);
13         var numOfKeys = Object.keys(data['Deal Link']).length;
14         console.log(numOfKeys);
15
16         for(var i=0; i<numOfKeys; ++i){
17             var item = `
18                 <div class="deal-container" style="height:320px">
19                     <a height="100%" target="_blank" href="${data['Deal Link'][i]} + "`
20                     <center>
21                         <div style="height:180px;">
22                             
24                 <br/>

```

## Celery connected to redis

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
depreacted.warn(description=f'The {setting!r} setting',
[2022-06-27 06:32:32,837: WARNING/MainProcess] E:\env\lib\site-packages\celery\app\utils.py:204: CDeprecationWarning:
    The 'CELERY_ACCEPT_CONTENT' setting is deprecated and scheduled for removal in
    version 6.0.0. Use the accept_content instead
depreacted.warn(description=f'The {setting!r} setting',
[2022-06-27 06:32:32,839: WARNING/MainProcess] E:\env\lib\site-packages\celery\app\utils.py:204: CDeprecationWarning:
    The 'CELERY_TASK_SERIALIZER' setting is deprecated and scheduled for removal in
    version 6.0.0. Use the task_serializer instead
depreacted.warn(description=f'The {setting!r} setting',
[2022-06-27 06:32:32,841: WARNING/MainProcess] Please run `celery upgrade settings path/to/settings.py` to avoid thes
e warnings and to allow a smoother upgrade to Celery 6.0.
[2022-06-27 06:32:32,867: INFO/MainProcess] Connected to redis://127.0.0.1:6379//
[2022-06-27 06:32:32,881: INFO/MainProcess] mingle: searching for neighbors
[2022-06-27 06:32:33,922: INFO/MainProcess] mingle: all alone
[2022-06-27 06:32:33,952: INFO/MainProcess] pidbox: Connected to redis://127.0.0.1:6379//.
[2022-06-27 06:32:33,962: WARNING/MainProcess] E:\env\lib\site-packages\celery\fixups\django.py:203: UserWarning: Usi
ng settings.DEBUG leads to a memory
leak, never use this setting in production environments!
warnings.warn('Using settings.DEBUG leads to a memory
[2022-06-27 06:32:33,963: INFO/MainProcess] celery@LAPTOP-8QRBNOET ready.

```

## Django server connection established

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
[27/Jun/2022 06:16:02] "GET /compare/?search_item=mobile HTTP/1.1" 200 21813
[27/Jun/2022 06:16:22] "GET /compare/data?search=mobile&store=snapdeal&req_data=primary HTTP/1.1" 200 482
[27/Jun/2022 06:16:22] "GET /compare?search_item=dal HTTP/1.1" 301 0
[27/Jun/2022 06:16:22] "GET /compare/?search_item=dal HTTP/1.1" 200 21813
[27/Jun/2022 06:16:25] "GET /compare/data?search=mobile&store=amazon&req_data=other HTTP/1.1" 200 7901
[27/Jun/2022 06:16:27] "GET /compare/data?search=mobile&store=flipkart&req_data=primary HTTP/1.1" 200 1178
[27/Jun/2022 06:16:32] "GET /compare/data?search=mobile&store=amazon&req_data=primary HTTP/1.1" 200 951
[27/Jun/2022 06:16:33] "GET /compare/data?search=mobile&store=snapdeal&req_data=other HTTP/1.1" 200 17470
[27/Jun/2022 06:16:48] "GET /compare/data?search=dal&store=flipkart&req_data=primary HTTP/1.1" 200 930
[27/Jun/2022 06:16:49] "GET /compare/data?search=dal&store=amazon&req_data=primary HTTP/1.1" 200 875
[27/Jun/2022 06:16:50] "GET /compare/data?search=dal&store=snapdeal&req_data=primary HTTP/1.1" 200 500
[27/Jun/2022 06:16:54] "GET /compare/data?search=dal&store=amazon&req_data=other HTTP/1.1" 200 7246
[27/Jun/2022 06:16:54] "GET /static/images/amazon%20fresh.png HTTP/1.1" 200 3785
[27/Jun/2022 06:17:00] "GET /compare/data?search=mobile&store=flipkart&req_data=other HTTP/1.1" 200 21902
[27/Jun/2022 06:17:06] "GET /compare/data?search=dal&store=snapdeal&req_data=other HTTP/1.1" 200 8751
[27/Jun/2022 06:17:18] "GET /compare/data?search=dal&store=flipkart&req_data=other HTTP/1.1" 200 29329
(env) PS E:\SmartComp> ..\env\Scripts\activate
>> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 27, 2022 - 06:35:11
Django version 4.0.5, using settings 'SmartComp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

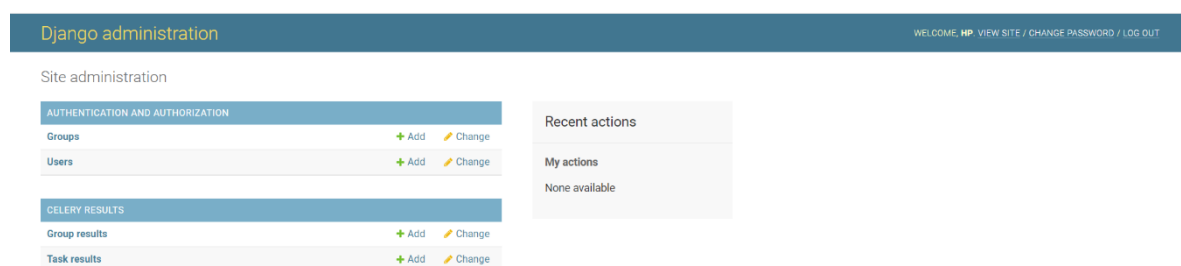
```

## SYSTEM TESTING

At each phase of integrating our Django application to the data retrieving dynamically as per the demand of users, there was a need to perform a lot of system testing. Used the following methods to test our application:

The tasks generated by multiple users are being taken care off by celery which we used for task scheduling. For every task there was a need to test how much time that task was taking in order to give the result and was the result of the task accurate or not.

For that purpose, Django ORM is used.



There is a table named “tasks” created inside Django-ORM (Django – object relational modal) in order to store the record of each and every request along with its result.



Django administration

WELCOME, HP VIEW SITE / CHANGE PASSWORD / LOG OUT

Home Celery Results Task results

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

CELERY RESULTS

Group results + Add

Task results + Add

Select task result to change

Q [ ] Search

2022 June 20 June 21 June 22 June 23 June 24 June 25 June 26

Action: [ ] Go 0 of 100 selected

TASK ID	PERIODIC TASK NAME	TASK NAME	COMPLETED DATETIME	TASK STATE	WORKER
4ddc9914-098a-42bf-ba69-fb5b4f91816a	-	home.tasks.snapdeal_best_deals	June 26, 2022, 11:38 p.m.	SUCCESS	celery@LAP-4HMORNLM
48bcff03-3c96-44de-ac31-bacfeeee3a42	-	home.tasks.amazon_best_deals	June 26, 2022, 11:38 p.m.	SUCCESS	celery@LAP-4HMORNLM
d4270cbe-26d3-4407-8913-9c07c7eb1b08	-	home.tasks.amazon_best_deals	June 26, 2022, 11:38 p.m.	SUCCESS	celery@LAP-4HMORNLM
d29a2c0b-9d5d-4c70-a5be-98362559eb8d	-	home.tasks.flipkart_best_deals	June 26, 2022, 11:38 p.m.	SUCCESS	celery@LAP-4HMORNLM
1d120460-2ce6-4a83-88cc-69bfc8fc584e	-	compare.tasks.flipkart_other_deals	June 26, 2022, 9:52 p.m.	SUCCESS	celery@LAP-4HMORNLM

By Task State

All

FAILURE

PENDING

RECEIVED

RETRY

REVOKED

STARTED

SUCCESS

By Completed DateTime

Any date

Today

Past 7 days

This month

This year

By Periodic Task Name

All

-

By Task Name

All

ADD TASK RESULT +

Here, are the task record of the requests made by the users.

Django administration

WELCOME, HP VIEW SITE / CHANGE PASSWORD / LOG OUT

Home Celery Results Task results <Task: 4ddc9914-098a-42bf-ba69-fb5b4f91816a (SUCCESS)>

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

CELERY RESULTS

Group results + Add

Task results + Add

Change task result

HISTORY

<Task: 4ddc9914-098a-42bf-ba69-fb5b4f91816a (SUCCESS)>

Task ID: 4ddc9914-098a-42bf-ba69-fb5b4f91816a  
Celery ID for the Task that was run

Task Name: home.tasks.snapdeal\_best\_deals  
Name of the Task which was run

Task State: SUCCESS  
Current state of the task being run

Worker: celery@LAPTOP-4HMORNLM  
Worker that executes the task

Result Content Type: application/json  
Content type of the result data

Result Encoding: utf-8  
The encoding used to save the task result data

Parameters

Task Positional Arguments: "0"  
JSON representation of the positional arguments used with the task

«

As a result, we are dynamically getting the data the user asked for.

## LIMITATIONS

### 1. Data caching:

As a humongous number of resources are required to run this application. Hence, it generates a large amount of cache memory due to which it sometimes takes 20 – 30 seconds to give the required results to be displayed. And this limitation of the project may lead to a bad user experience sometimes.

### 2. Blocked instances of web scrapping:

As all the data is being scrapped from the various e-commerce websites like Amazon, Flipkart and Snapdeal and is being displayed dynamically, i.e. depending upon the user's requirement, sometimes when, a large number of users are accessing those websites they are displayed as blocked for some users so that they can handle the crowd of users sending multiple requests to the server again and again.

In such a case no data can be mined from those. Hence, at that time there will be no deals available for the user to compare over our application.

## CONCLUSIONS

We try to spend as much time as we can, to check whether or not we are buying a right thing? And the most important; whether we are buying it at a reasonable rate or not? Providing the ability to compare products is at the heart of what comparison sites are trying to achieve. However, those sites are achieving this with varying levels of success. My target of SmartComp is simply log on to this web application, learn about the products, compare their prices from various e-commerce websites, search and browse the options, compare the number of reviews and ratings those products have got and shop with satisfaction of trust and all these at your door steps. This will be saving a lot of time of the users, which they spend in going to different websites and comparing prices and then again comparing.

Hence, SmartComp is a smart tool to compare and analyse prices and reviews of products across various e-commerce websites.

## **FUTURE SCOPE**

### **1. Optimizing data caching:**

Data Caching is to be optimized as due to a humungous number of requirements, this application is generating lot of cache memory, due to which its speed for giving results slow down sometimes depending upon the internet and cache generated it takes 20-30 seconds to give required results, which may lead to a bad user experience. That needs to be optimized as out future aspects.

### **2. Subscription Modal:**

While our web application offers various features they are still not as attractive as is required to gain a substantial user base to capitalize upon. As such, we believe that we can add a subscription model to our application in future, offering the following features:

- Special discount coupons for our subscribers.
- Tier system to classify subscribers with different purchasing frequency and reward them accordingly with exclusive deal offers.

### **3. Advanced Filters:**

We all find it really convenient to just apply filters as per our requirements and getting the range of choices short down from a huge number to a smaller one. On e-commerce websites like Amazon, Flipkart, Myntra, Snapdeal etc, filters like price

range, colour, material, company and a lot more are available.

As are future aspect we will be automating that feature on the server side so that data can be displayed dynamically using the filter choices of the users as well in our website.

In this way, this project can be scaled to a good level, by gaining more and more users.

## BIBLIOGRAPHY / REFERENCE

<https://www.selenium.dev/documentation/>

<https://developer.mozilla.org/en-US/docs/Web/XPath>

<https://docs.celeryq.dev/en/stable/>

<https://chromedriver.chromium.org/>

<https://www.selenium.dev/documentation/webdriver/>

<https://redis.io/docs/>

<https://docs.djangoproject.com/en/4.0/>

<https://docs.djangoproject.com/en/4.0/ref/contrib/admin/>

<https://developer.mozilla.org/en-US/docs/Glossary/jQuery>

<https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>

<https://docs.python.org/3/>

<https://testdriven.io/blog/django-and-celery/>

<https://www.w3schools.com/html/>

<https://www.w3schools.com/w3css/defaultT.asp>

<https://www.w3schools.com/js/>

<https://www.w3schools.com/bootstrap4/>

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>

<https://stackoverflow.com/>

<http://www.gevent.org/>

<https://www.jetbrains.com/pycharm/>