🛒 Product URL Crawler – Documentation

# 📌 Problem Statement

Develop a scalable, efficient, and robust web crawler to discover all product URLs from a list of e-commerce domains. The crawler should intelligently adapt to diverse URL structures used across different websites and produce structured output for each domain.

# 🎯 Objective

- Discover product URLs from e-commerce websites.

- Output a structured mapping from domain → product URLs.

- Ensure performance, scalability, and robustness.

# 🌐 Target Domains (Minimum Required)

* https://www.virgio.com/
* https://www.tatacliq.com/
* https://nykaafashion.com/
* https://www.westside.com/

# 🧠 Approach Overview

## 1. URL Discovery Strategy

Breadth-First Traversal (BFS): Crawl pages using a queue, avoiding revisits.  
Domain Whitelisting: Only follow links that belong to the current domain.  
Pattern Matching: Product URLs are identified using common keyword patterns observed in e-commerce platforms.

## 2. Product URL Identification Heuristics

A predefined set of substrings is used to match potential product URLs:

['/product/', '/products/', '/item/', '/p/', '/details/', '/shop/', '/pd/', '/collections/', 'cart', 'p-mp']

## 3. Page Content Scraping

Selenium (Headless Chrome) is used for rendering JavaScript-heavy websites.  
BeautifulSoup parses the DOM and extracts anchor tags.  
All discovered links are normalized using urljoin() and filtered by domain.

## 4. Data Storage

Discovered product URLs are stored in Excel files using Pandas (output/domain\_name\_products.xlsx).  
Each domain is crawled independently and saved separately.

# ⚙️ Implementation Overview

The core script is crawler.py, which performs:  
- Environment setup using Selenium  
- BFS traversal of pages  
- Filtering & logging of product URLs  
- Saving output as .xlsx  
- Logging to logs/crawler.log (rotating file)

# 🧪 Scalability & Performance

The architecture supports scaling up to hundreds of domains.  
Future improvements could include:  
- Concurrency with asyncio or multithreading  
- URL pattern learning via machine learning  
- Distributed crawling using tools like Scrapy or Frontera

# 🛡️ Robustness Features

- Domain-based link validation ensures no cross-site crawling.  
- Exception handling during driver.get() to handle broken/malformed pages.  
- Duplicate avoidance via visited and found\_products sets.  
- Logging at INFO, DEBUG, and ERROR levels for debugging and monitoring.

# 📁 Output Structure

Each domain will generate an Excel file in the output directory.

Each file contains unique product URLs in the following format:

| product\_url |  
|-------------|  
| https://www.example.com/product/12345 |

# 📹 Loom Walkthrough

🎥 Loom Video Walkthrough Link Here: https://www.loom.com

# 🏁 Conclusion

This crawler provides a modular, scalable framework for discovering product URLs on e-commerce platforms. It can easily be extended to support more domains, more complex product identification heuristics, and faster parallel crawling. Contributions and suggestions are welcome!