StarTech Web Scraper

1. Overview

This project implements a web scraper for the StarTech Bangladesh website, designed to crawl categories and extract product information into a MongoDB Atlas database. The scraper uses Crawl4AI (Playwright-powered) as the primary engine and FireCrawl as a fallback for robustness. It incorporates concurrency control, retry logic, and MongoDB best practices to handle production-scale workloads.

2. Core Components

- asyncio: Manages asynchronous tasks and concurrency.
- crawl4ai.AsyncWebCrawler: Primary async crawler using Playwright for dynamic content.
- firecrawl.Firecrawl: Fallback HTML retriever if Crawl4AI fails.
- BeautifulSoup (bs4): Parses HTML and extracts product/category information.
- pymongo: Interface with MongoDB Atlas for persistence.
- tenacity: Provides retry logic for transient errors.
- dotenv: Loads environment variables (e.g., MongoDB connection string).
- logging: Structured logging for debugging and monitoring.

3. MongoDB Integration

Initialization (init_mongodb)

- Connects to MongoDB Atlas with increased timeouts, retryWrites, majority write concern, and a unique index on the url field to prevent duplicate products.

Data Persistence

- Products are upserted to avoid duplicates.
- Batched inserts improve performance and reduce MongoDB write overhead.

4. Crawling & HTML Fetching

fetch_html(url)

- Adds a delay between requests to respect server load.
- Crawl4AI is attempted first for dynamic content.
- FireCrawl acts as fallback.
- Tenacity retry decorator ensures resilience.

5. Category Discovery

discover_categories(root_url)

- Extracts root categories and first-level subcategories from homepage navigation.
  nav.navbar .nav-item > a.nav-link
  Example: `/desktop`, `/laptop-notebook`, `/component`.

  nav.navbar .dropdown-menu a
  Example: `/desktops/brand-pc`, `/desktops/gaming-pc`.

- Filters to include only paths with 1–2 segments.
- Produces clean, deduplicated category list.

6. Product Scraping

scrape_category(url)

- Extracts product details like URL, category, title, price, and stock status.
- Extracts product cards with flexible selectors:
  .p-item, .product-layout

- Captures:

- **URL** → normalized with `urljoin`.

- **Category** → derived from the first path segment of URL.
- **Title** → `.p-item-name`, `.product-name`, or `h4 a`.
- **Price** → `.p-item-price`, `.price-new`, `.price`.
- **Status** → `.stock-status`, `.status`.

Stores results in a structured dictionary:

```
{
    "url": product_url,
    "category": category,
    "title": title,
    "price": price,
    "status": status,
    "scraped_from": url
}
```

7. Concurrency Model

`process_category_chunk`

- Uses asyncio.Semaphore to limit parallel category scrapes (`MAX_WORKERS = 3`).
- Processes categories in chunks of `MAX_WORKERS * 2` for balance.
- Adds a delay (`REQUEST_DELAY`) between category requests to avoid server overload.

8. Retry, Resilience & Logging

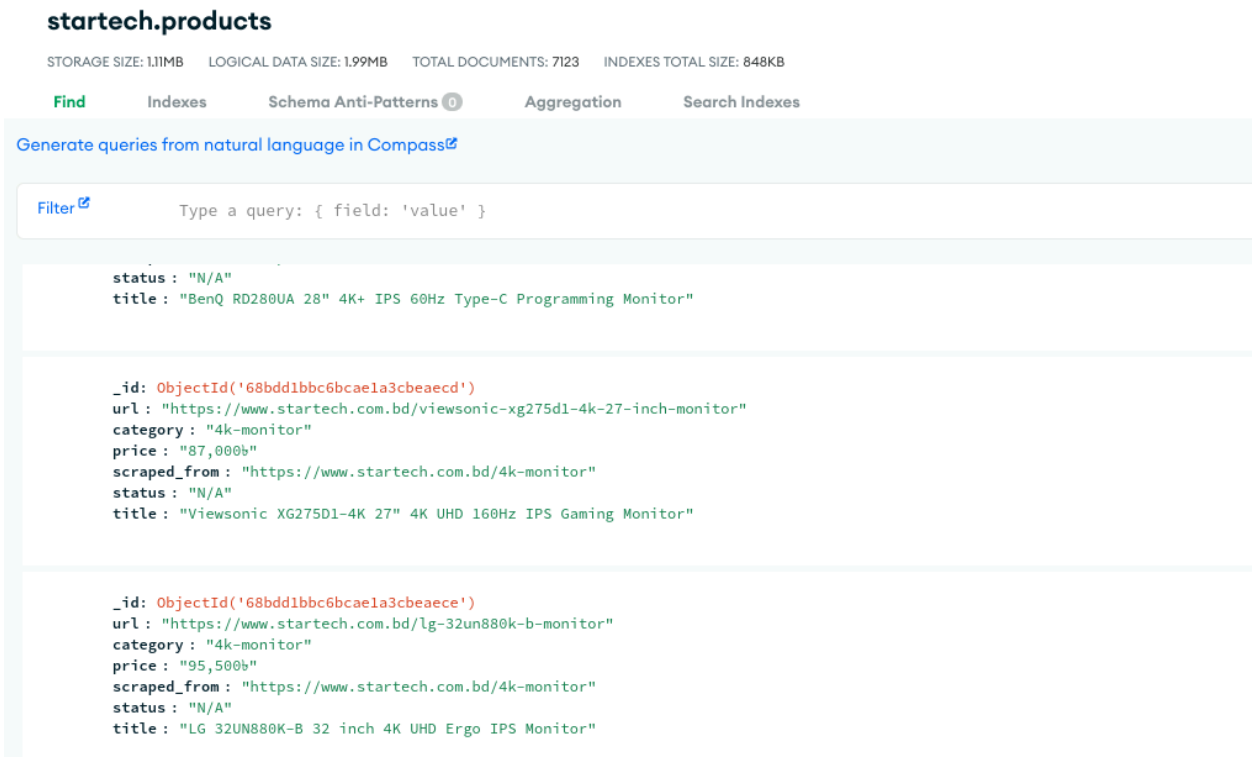
- Retry logic (tenacity) ensures transient errors (timeouts, connection drops) are retried.
- Graceful failure handling: categories or products that fail extraction don’t crash the pipeline.
- Structured logging provides:

1. Connection status (MongoDB)
2. Progress updates (# categories processed, # products inserted)
3. Error traces for debugging

9. Performance & Reliability Features

- Request delays reduce server strain.
- Batch inserts optimize MongoDB performance.
- Concurrency limits prevent overload.
- Retry logic improves resilience.
- Upserts ensure database consistency.
- Indexing on URL speeds queries.

MongoDB :



Terminal Output :