My Approach

The approach I took combines a Kivy-based user interface with a multithreaded Amazon scraper. While the scraper runs in the background, the Kivy UI stays fully responsive, showing a rotating loading spinner, a progress bar, and live updates about the scraping process.

The scraper itself runs on a separate thread so it doesn't freeze the interface. It goes through Amazon search results, collects links to product pages, and then visits each one to pull details like the product name, price, shipping cost, seller, and brand. It uses the `requests` library to make HTTP calls and `BeautifulSoup` to parse the HTML. The goal is to gather 50 valid products, saving each one to its own JSON file while also updating a combined summary file with all the data.

To avoid getting blocked by Amazon, I used realistic HTTP headers (like a real User-Agent and Referer) to make the requests look like they're coming from a normal browser. I also added random delays between requests to mimic human browsing behavior and reduce the chances of detection.

If you’re looking for a more detailed breakdown of how the scraper works, especially the logic in `scraper.py`, there’s a full explanation at the bottom of the project’s README file.

the rist challenge i faced is Amazon aggressively defends against bots, and I ran into issues with requests getting blocked or returning incomplete pages. To work around this, I mimicked real browser behavior by setting custom HTTP headers like a common User-Agent and Referer. I also added random delays between each request to make the scraping appear more human-like, and programmed the scraper to simply skip over any pages that failed to load properly instead of crashing or getting stuck.

headers:





sleep to mimc wait:



if soup(the page is scraped is empty):



skipping incomplete products in oreder to get 50 json with at least title in them:



also initially took the shipping price without stripping it checking if its even being shipped, fixed it here:



also realizin i need to use thread for the ui response