**Project Documentation: Amazon Product Scraper with Flask UI**

**1. Introduction**

This project is a **web scraping and visualization tool** that extracts product details from Amazon and presents them in a **Flask-powered web application**.  
The scraper collects product information such as title, price, rating, reviews, and product images. The Flask application provides a user-friendly interface to search, filter, and display these products in a grid format similar to Amazon’s own UI.

**2. Objectives**

* To develop a scraper that extracts product data from Amazon.
* To handle Amazon’s blocking issues (503/505 errors) by using **rotating headers** and a **mock data fallback system**.
* To store product data in a structured JSON format.
* To build a **Flask-based web interface** to display scraped products.
* To provide features like search filtering and Amazon-style UI design.

**3. System Requirements**

**Hardware:**

* Minimum: 4 GB RAM, 2-core processor
* Recommended: 8 GB RAM, 4-core processor

**Software:**

* Python 3.9+
* pip (Python package manager)
* Flask (for web app)
* BeautifulSoup4 (for HTML parsing)
* Requests (for HTTP requests)
* Git (for version control)
* Browser (Chrome/Edge/Firefox)

**4. Technologies Used**

* **Python** → Core programming language.
* **Flask** → Backend web framework.
* **HTML, CSS** → Frontend UI design.
* **Requests** → For sending HTTP requests to Amazon.
* **BeautifulSoup** → For parsing HTML and extracting product details.
* **JSON** → For storing scraped product data.
* **Git & GitHub** → For version control and collaboration.

**5. System Architecture**

**Step 1: Scraper Module (scraper.py)**

* Sends an HTTP request to Amazon search page.
* Uses rotating User-Agent headers to avoid detection.
* Parses HTML using BeautifulSoup.
* Extracts title, price, rating, reviews, and image for each product.
* Saves data into products.json.
* If Amazon blocks (status ≠ 200), loads **mock product data** instead.

**Step 2: Web Application (appli.py)**

* Flask loads products.json.
* Displays products in an Amazon-like UI (index.html).
* Provides search functionality to filter products by title.
* Additional UI features: header with logo, cart, splash screen, product grid.

## 7. Features

1. **Web Scraping**
   * Extracts Amazon product details (title, price, rating, reviews, image).
   * Handles Amazon blocking (503/505) gracefully with fallback mock data.
2. **Data Storage**
   * Stores results in products.json for reusability.
3. **Flask Web Application**
   * Displays products in an Amazon-style layout.
   * Includes header with logo, cart, and navigation links.
   * Search bar to filter products by keyword.
4. **UI Enhancements**
   * Splash screen before main page.
   * Product cards with images, price, and reviews.
   * Responsive design using CSS grid.

## 8. Challenges Faced

* Amazon frequently blocks scraping requests (status 503/505).
* Dynamic content loading makes scraping inconsistent.
* Solution: used **rotating headers** and **mock fallback data**.
* Maintaining Amazon-like UI with only static JSON data.