Project Report

on

Amazon & YouTube No code Scraper

at

Finstein Advizory

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology

in

COMPUTER ENGINEERING (ARTIFICIAL INTELLIGENCE)

by

V V N S Yujit Kumar(22012532005)

Under the Guidance of

Internal Guide:	**External Guide:**
Prof. Sheetal Kumar Dixit	Mr. Shaktivel Mageshwaran

Semester VIII

Submitted to

Department of Computer Engineering

Faculty of Engineering & Technology

Ganpat University, Ganpat Vidyanagar-384012, Gujarat

May 2025

U.V. PATEL COLLEGE OF ENGINEERING

Ganpat Vidyanagar-384012, Mehsana-Gandhinagar Highway,
District-Mehsana, Gujarat, INDIA

CERTIFICATE

This is to certify that **Mr.**V V N S Yujit Kumar student of **B.Tech. Semester VIII (Computer Engineering-Artificial Intelligence)** has completed his/her full semester project work titled "**Amazon & YouTube No code Scraper**" satisfactorily in partial fulfillment of the requirement of Bachelor of Technology degree of Computer Engineering of Ganpat University, Ganpat Vidyanagar in the year 2024-2025.

Prof. Sheetal Kumar Dixit
Internal Guide

Dr. Paresh Solanki
Head of the Department
Computer Engineering

Stamp

Date of Examination: 10/05/2025

ABSTRACT

In the age of data-driven decision-making, access to real-time and structured information has become crucial for individuals, researchers, and businesses. E-commerce platforms like Amazon and content-sharing sites like YouTube host enormous volumes of publicly accessible data, which, if extracted efficiently, can offer immense insights. However, the manual process of data extraction is tedious, inconsistent, and often unscalable. This project presents a comprehensive solution in the form of a robust Web Scraping Application that automates the process of data collection from Amazon and YouTube, ensuring speed, consistency, and accuracy in data handling.

The system is developed using Python as the core language, with Flask serving as the backend framework to create a dynamic web interface. It supports user authentication via a secure login/signup system using SQLite and hashed credentials. Once authenticated, users can interact with a clean and responsive dashboard that allows them to initiate data scraping tasks from either Amazon or YouTube.

The Amazon Scraper is engineered to perform multi-page scraping of search results while handling a wide array of product metadata such as product title, price, rating, number of reviews, delivery options, URLs, deal status, and more. To ensure ethical scraping, anti-blocking measures such as user-agent rotation and random delays are integrated, alongside rate-limiting functionality. Scraped data can be exported in three widely used formats: CSV, Excel, and JSON.

The YouTube Scraper focuses on channel video data extraction. Users can retrieve video titles, view counts, and upload dates from any public channel. Pagination support enables scraping of multiple videos in a single session. The platform is particularly useful for analysts studying content engagement or video publishing trends.

The system's architecture emphasizes security, performance, and scalability. It implements CSRF protection, session-based authentication, and input validation. The use of Selenium WebDriver enables accurate scraping of dynamic content, while BeautifulSoup handles HTML parsing. Pandas powers the data processing, cleaning, and formatting layers.

In addition, the project implements efficient data management by automatically deleting old downloads and organizing files per user. A blog section is integrated into the UI to educate users about scraping ethics, legality, and best practices.

Overall, this project represents a professional-grade, modular, and secure scraping framework capable of automating data extraction from two of the most data-rich platforms on the internet. It can be extended easily to support other sources, making it ideal for researchers, developers, and digital marketers.

INDEX

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The digital era has witnessed an unprecedented surge in the amount of data generated online, particularly on e-commerce platforms like Amazon and video-sharing platforms like YouTube. These platforms offer valuable insights ranging from consumer preferences to content engagement metrics. However, accessing this data manually is time-consuming and prone to inconsistency. This project addresses the need for an automated, scalable, and secure system to extract structured data from Amazon and YouTube using web scraping technologies.

The developed system is a web-based scraping platform built using Python and Flask that allows users to extract and download real-time data from Amazon and YouTube with ease. With a user-friendly dashboard, robust backend, secure authentication, and multi-format export support, the application serves as a powerful data collection tool. It supports automation of tedious tasks, accurate parsing of complex web structures, and adherence to ethical scraping standards.

1.1 Background and Motivation

In the age of information, data is the new oil. Businesses, researchers, and analysts heavily rely on data-driven decision-making, and the demand for efficient methods of acquiring relevant data has skyrocketed. However, despite the abundance of data available on the web, much of it remains locked behind complex website interfaces. Web scraping has emerged as a vital technique to bridge this gap by automating the extraction of structured information from unstructured web pages.

E-commerce platforms like Amazon and content platforms like YouTube are two of the most valuable sources of data in today's digital economy. From market trends and product reviews to consumer behavior and content analysis, the insights derived from these platforms are essential across multiple industries. However, scraping such dynamic, interactive websites involves overcoming numerous technical and ethical challenges, such as rate limiting, CAPTCHA verification, JavaScript-loaded content, and anti-bot mechanisms.

The motivation behind this project is to design and develop a robust, scalable, and secure web scraping application that can extract valuable data from Amazon and YouTube, process it, and make it available to end-users through a user-friendly web interface. The project not only serves as a functional data collection tool but also stands as an academic demonstration of integrating backend web development, browser automation, data engineering, and security best practices.

1.2 Objectives of the project

The primary objectives of this project are as follows:

- To develop a secure web-based application that can scrape structured data from Amazon and YouTube.
- To implement a responsive and intuitive user interface for end-users to initiate scraping tasks.
- To ensure that data is extracted in real-time from dynamic websites using Selenium and BeautifulSoup4.
- To allow users to download data in multiple formats: CSV, Excel, and JSON.
- To integrate a login and registration system with secure session management.
- To implement best practices in web scraping ethics, data validation, and performance optimization.

1.3 Problem Statement

Although tools and libraries exist for scraping web data, most are either command-line based or require technical expertise to operate. There is a lack of unified web-based platforms that allow users, particularly non-technical users, to collect data from Amazon and YouTube in a secure, flexible, and customizable manner. Furthermore, scraping from these platforms presents additional complexities due to their dynamic nature and the presence of anti-bot mechanisms.

This project aims to solve this problem by developing a complete, end-to-end web scraping platform that:

- Is easy to use through a web browser
- Handles authentication and user-specific sessions
- Supports real-time scraping with multi-page handling
- Protects against detection and rate limiting
- Provides instant downloadable outputs in user-chosen formats

1.4 Overview of the System

The system is a Flask-based web application with a backend designed to handle routing, form submissions, scraping job executions, data processing, and user session management. The frontend is developed using HTML and CSS to ensure a clean and responsive user experience.

The application consists of the following modules:

1. **Amazon Scraper Module**
    - Scrapes product data including title, price, rating, reviews, delivery time, and URL.
    - Supports multiple result pages and incorporates delay mechanisms to mimic human behavior.
2. **YouTube Scraper Module**

- Scrapes data from public YouTube channels including video title, view count, and upload date.
- Automates scrolling to capture all content from channels with large video lists.

3. **Authentication System**
    - Users can register and log in securely.
    - Passwords are hashed using Werkzeug's hashing utilities.
    - Sessions are managed through Flask's secure session mechanism.
4. **Export & Download System**
    - Users can select their desired format before scraping.
    - Upon completion, files are made available for download from the dashboard.
5. **User Interface**
    - The dashboard is designed for maximum usability.
    - Clear feedback is provided for each action including progress and error messages.

1.5 Significance of the Project

This project plays a pivotal role in demonstrating how modern technologies can be orchestrated to solve real-world problems. From a technical standpoint, the application showcases:

- How browser automation tools like Selenium can be used to interact with JavaScript-heavy websites
- How Flask can be used to create a scalable web application
- How user input can be safely handled and used to dynamically trigger backend processes
- How to protect a web app using password hashing, session management, and input validation

From a practical standpoint, the project has direct applications in:

- Market research and e-commerce analytics
- Content tracking and trend analysis on YouTube
- Academic and industrial research
- Digital marketing and SEO

1.6 Target Audience

The system is designed to be useful for a wide range of users:

- Students and researchers looking for data on trends, prices, or content performance
- Digital marketers analyzing competitor products or content strategies
- E-commerce analysts evaluating product demand and customer reviews
- Content creators who wish to keep track of performance metrics

1.7 Key Technologies Used

- **Frontend**: HTML, CSS
- **Backend**: Flask (Python)
- **Scraping**: Selenium, BeautifulSoup4
- **Data Export**: Pandas
- **Security**: Werkzeug, Flask Sessions
- **Browser Automation**: Chrome WebDriver (headless mode)
- **Database**: SQLite (for authentication)

1.8 Summary

This project represents a well-rounded fusion of web development, automation, data processing, and security. It addresses a real-world need for accessible and efficient data collection from complex websites. By building this application, not only have we created a useful tool, but also explored the depths of modern web scraping and its integration into full-stack web applications. It stands as an example of how open-source tools and intelligent design can be combined to create impactful, real-world applications.

2. PROJECT SCOPE

The project scope defines the boundaries, functionalities, and extent of capabilities offered by the web scraping application. This project focuses solely on extracting structured data from Amazon (an e-commerce platform) and YouTube (a video content platform). While many scraping tools in the market provide broad, general-purpose capabilities, this project emphasizes a focused, robust, and ethically developed solution for these two high-traffic websites.

The scope ensures clarity on what the application will accomplish, what it will not, how it operates, and for whom it is intended. It also outlines the constraints, technological boundaries, and possibilities for future expansion.

2.1 Functional Scope

The primary functionalities that fall within the project scope include:

2.1.1 Amazon Data Scraping Module

The Amazon scraping module is responsible for fetching structured product-related information directly from Amazon search result pages. It is capable of:

- **Multi-page scraping**: Users can fetch product data from multiple search results pages.
- **Product Information Extraction**:
    - Product Title
    - Price (if available)
    - Rating
    - Number of Reviews
    - Delivery Information
    - Product URL
    - Sponsored Status (whether the result is a sponsored ad)
    - Deal or Offer Information
- **Rate Limiting and Anti-blocking**:
    - Random delays between requests
    - Headless browser automation using Selenium
    - User-agent rotation (to avoid detection)

- **Data Export Options**:
    - CSV (.csv)
    - Excel (.xlsx)
    - JSON (.json)
- **Error Handling**:
    - Graceful fallback when elements are missing
    - Logging system to record scraping success/failures

2.1.2 Youtube Channel Scraping Module

This module focuses on extracting video content data from public YouTube channels. It includes:

- **Channel-Wide Data Collection**:
    - Video Title
    - View Count
    - Upload Date
    - Video URL
- **Pagination / Scroll Handling**:
    - Automated scrolling to load all videos from a channel
- **Data Export Functionality**:
    - CSV
    - Excel
    - JSON
- **Dynamic Element Handling**:
    - Detect and parse content rendered via JavaScript

2.1.3 Authentication and Security

- **User Authentication System:**
    - Signup with secure password hashing (Werkzeug)

        - Login system with session management
        - Logout functionality
- **Session Control:**
    - Prevents unauthorized access to the dashboard and scraping modules
- **Input Validation:**
    - All forms are protected against invalid or malicious input.

2.1.4 User Interface and Dashboard

- Responsive Web UI using HTML and CSS
- **Functional Dashboard:**
    - Buttons to trigger YouTube or Amazon scrapers
    - Dropdown to select export format
    - Display progress messages (e.g., "Scraping in progress…")
    - Error/Success feedback messages
- Educational Blog Section (Static content)
- Download Management System:
- Automatically enables download options after scraping is complete.

2.2 Non-Functional Scope

The project also covers several non-functional requirements to ensure quality and reliability.

2.2.1 Performance

• The application is optimized to handle multiple pages of content efficiently.

• Memory management is ensured through use of Pandas and controlled scraping loops.

2.2.2 Reliability

• Uses structured exception handling to prevent crashes.

• Logs errors and status updates during scraping.

2.2.3 Usability

• Clean and modern dashboard UI

• Tooltips, error prompts, and feedback messages improve user experience.

2.2.4 Security

• Passwords are securely stored using hashing

• Prevents direct URL access to restricted pages without login

• CSRF prevention through Flask’s secure session handling

2.3 Out of Scope Features

It is equally important to outline what is not included in the current project:

- Scraping from websites other than YouTube and Amazon
- Real-time analytics or visualization of scraped data within the dashboard
- Database storage of scraped content data is saved as downloadable files, not persisted in a database
- Mobile application version this project is web-based only
- Video content download YouTube scraper only fetches metadata, not actual video content
- Login via OAuth or third-party authentication (Google/Facebook, etc.)
- Scraping from private YouTube content or Amazon user accounts

2.4 Assumptions

This project was developed under the following assumptions:

- The user is scraping publicly available data and not violating terms of service.
- The user has appropriate permissions to run scraping operations for academic or personal use.
- The system will be deployed and run in a controlled environment (e.g., localhost or private server).
- Chrome WebDriver is installed and operational on the system where the application runs.

2.5 Deliverables

The project delivers the following components:

- A Flask-based web application with Amazon and YouTube scraping capabilities.
- Two standalone scraping modules under a scrapers/ directory.
- A responsive dashboard with scraping control and download management.
- Authentication system with signup/login functionality.
- Organized file exports in the format selected by the user.
- Logging mechanism to track scraping operations and errors.

2.6 Future Scope (Within Amazon and Youtube)

Though this project currently focuses on specific data points, the architecture allows for future expansion within Amazon and YouTube, such as:

2.6.1 For Amazon

• Scraping customer reviews and ratings distribution

• Tracking historical price changes (price history)

• Category-level or brand-level data collection

• Integrating live alerts for price drops or stock changes

2.6.2 For Youtube

- Collecting engagement metrics (likes/dislikes/comments)
- Video description and tag scraping
- Channel metadata like subscriber count
- Scraping playlists and video categories
- Generating time-series reports of channel activity

2.7 Summary

In conclusion, the project’s scope is centered on building a feature-rich, reliable, and secure web scraping application that supports two of the world’s most significant platforms: Amazon and YouTube. It provides not just scraping capabilities, but also integrates robust backend operations, session management, and a sleek frontend dashboard, all while conforming to ethical standards and modern security practices.

The design encourages modularity and extensibility while being constrained to the platforms mentioned to maintain a high degree of performance, control, and compliance.

3. FEASIBILITY ANALYSIS

Feasibility analysis is a vital phase in the software development lifecycle as it evaluates whether a proposed system is technically, economically, operationally, and practically viable. This section provides an in-depth examination of various aspects of feasibility for the Web Scraping Application for YouTube and Amazon, ensuring that the project can be successfully implemented, maintained, and utilized.

3.1 Technical Feasibility

Technical feasibility assesses whether the current hardware, software, and technical resources are sufficient to support the development, deployment, and maintenance of the project.

3.1.1 Technology Stack

• **Backend Framework:** Python with Flask a lightweight, fast, and ideal for modular backend systems.

• **Frontend:** HTML, CSS ensures a clean and responsive user interface.

• **Web Scraping Tools:** Selenium (for dynamic scraping), BeautifulSoup (for parsing HTML), Pandas (for data manipulation).

• **Browser Automation:** Chrome WebDriver running in headless mode enables scraping without opening a visual browser.

• **Database:** SQLite for authentication a lightweight and file-based, making it ideal for small to mid-sized applications.

• **Data Export Formats:** CSV, Excel (XLSX), and JSON covering the most commonly used data formats.

3.1.2 Compatibility

- Compatible with all modern operating systems (Windows, macOS, Linux).
- Works with the latest version of Google Chrome and Chromium-based browsers.
- Python-based platform-independent and highly portable.
- Frontend tested for responsiveness and adaptability across various screen sizes.

3.1.3 Resource Availability

• All required tools and libraries are open-source and well-documented.

• Requires minimal system resources a standard computer with Python and Google Chrome is sufficient.

• Easy-to-install dependencies via requirements.txt.

3.1.4 Security Technologies

• Passwords are hashed using Werkzeug.

• Session-based authentication.

• Protection against CSRF and XSS using Flask's secure mechanisms.

• Input validation and secure file operations.

3.2 Time Feasibility

Time schedule feasibility ensures that the project can be completed within a practical and acceptable timeframe. This project has been carefully planned and executed over a period of 3 months (12 weeks). Below is a detailed breakdown of the project timeline:

- Week 1: Requirements Gathering & Initial Setup
- Week 2: System Design
- Week 3: Backend Setup
- Week 4: Frontend Integration
- Week 5: Amazon Scraper – Basic Extraction
- Week 6: Amazon Scraper – Advanced Features
- Week 7: YouTube Scraper – Core Development
- Week 8: Dashboard Connectivity
- Week 9: Testing and Error Handling
- Week 10: Optimization and Logging
- Week 11: Documentation
- Week 12: Final Integration

3.3 Operational Feasibility

Operational feasibility determines whether the system will function as intended in a real-world environment and whether users will be able to interact with it effectively.

3.3.1 User Interaction

• Intuitive user interface with dashboard-style controls.

• Clear instructions for data scraping operations.

• Download management and format selection are streamlined.

• No need for the user to write code fully no-code usage.

3.3.2 Error Handling Feedback

• The system provides real-time error messages for issues like invalid URLs, network timeouts, or browser issues.

• Logs are maintained to help troubleshoot and improve scraping strategies.

3.3.3 Deployment Environment

• Designed to run locally or on a private server.

• Minimal dependencies and environment setup needed.

3.3.4 User Acceptance

- Designed with educational and research users in mind.
- Use cases tested include:
    - Academic research on e-commerce trends
    - Content analysis of YouTube videos
    - Market tracking for small businesses

3.4 Implementation Feasibility

Implementation feasibility assesses how effectively the system can be built, integrated, and deployed in the existing environment.

3.4.1 Ease of Development

• Modular architecture for separation of concerns.

• Independent scraping scripts placed under /scrapers/.

• Unified interface through Flask routing.

3.4.2 Maintainability

• Well-documented codebase.

• Clear folder structure.

• Easy to extend functionality by adding new scrapers under the existing framework.

3.4.3 Deployment Readiness

• Can be deployed using simple Flask run commands.

• Only dependency is Python, along with standard packages listed in requirements.txt.

3.4.4 Compatibility With Existing Tools

Outputs (CSV, Excel, JSON) are compatible with most data analysis and visualization tools like:

- Excel
- Google Sheets
- Power BI
- Tableau
- Pandas/Jupyter for Python-based analysis

3.5 Economic Feasibility

Economic feasibility assesses whether the project is cost-effective, both in terms of development and operational expenses.

3.5.1 Cost of Development

• Zero licensing fees: All tools and libraries used are open-source (Flask, Selenium, Pandas, etc.).

• No third-party APIs required, eliminating recurring costs.

• Development carried out on existing systems without need for cloud infrastructure.

3.5.2 Cost of Maintenance

• Minimal maintenance needed due to the lightweight architecture.

• No external hosting costs unless the application is deployed on a live server.

• No costs associated with third-party integrations or databases beyond SQLite.

3.5.3 Time as Cost

• Time investment was kept minimal through agile development and modular scripting.

• Reusability of code reduces long-term development costs.

3.5.4 Return of Investment (ROI)

High ROI in terms of:

- Data collection for research
- Automation of repetitive data gathering tasks
- Education and demonstration of real-world data extraction

3.6 Summary

The comprehensive feasibility study of this web scraping project confirms that it is technically sound, operationally robust, implementable, economically viable, and achievable within a realistic time frame. The well-thought-out architecture, open-source tools, and user-centric design contribute to a solution that is both sustainable and scalable for future needs within the domains of YouTube and Amazon data extraction.

4. SOFTWARE AND HARDWARE REQUIREMENT

The successful development, deployment, and execution of this web scraping project for Amazon and YouTube heavily rely on specific software and hardware configurations. This section outlines the minimum and recommended requirements for running the system effectively, along with the tools and technologies employed throughout the project lifecycle.

4.1 Hardware Requirements

Hardware plays a crucial role in the performance of the system, especially when dealing with web scraping tasks that involve browser automation, heavy data processing, and export operations. While the application is designed to be lightweight and optimized, below are the hardware requirements:

Minimum Hardware Requirements

- **Processor:** Intel Core i3 (7th Gen or higher) / AMD Ryzen 3 or equivalent
- **RAM:** 4 GB DDR4
- **Storage:** 100 GB HDD/SSD (minimum 10 GB free space required)
- **Graphics:** Integrated GPU
- **Display:** 13" HD (1366 x 768) or higher resolution
- **Internet:** Stable broadband connection (minimum 2 Mbps) – Required for scraping live data
- **USB Port:** Optional – for backup/export using USB drives
- **Others:** Basic input devices (keyboard, mouse), speakers (for UI feedback if implemented)

Recommended Hardware Requirements

- **Processor:** Intel Core i5 or i7 (10th Gen or newer) / AMD Ryzen 5/7 or equivalent
- **RAM:** 8 GB DDR4 or higher
- **Storage:** 256 GB SSD (for faster read/write operations)
- **Display:** 15.6" Full HD (1920 x 1080)
- **Internet:** Stable high-speed connection (above 10 Mbps)
- **Backup:** Cloud storage or external HDD for regular backups

4.2 Software Requirements

The project leverages a combination of modern development tools, libraries, and frameworks to deliver a robust and scalable solution. Below are the software requirements categorized by their purpose:

Operating System

- **Minimum:** Windows 10 / Linux Ubuntu 20.04+
- **Recommended:** Windows 11 / Linux Ubuntu 22.04 LTS

Development Environment

- **IDE/Editor:**
    - Visual Studio Code *(with Python extension installed)*
    - PyCharm Community Edition *(optional)*

Backend Technologies

- **Language:** Python 3.8 or higher
- **Framework:** Flask (Web framework)
- **Database:** SQLite3 (Lightweight RDBMS used for user authentication)
- **Libraries & Modules:**
    - Flask-Login (User session management)
    - Flask-WTF (CSRF protection, form handling)
    - Werkzeug (Password hashing and security)
    - pandas (Data manipulation and processing)
    - openpyxl (Excel file handling)
    - json, csv (Built-in Python modules for data export)

Web Scraping Tools

- **Browser Automation:** Selenium WebDriver
- **Browser Driver:** ChromeDriver (compatible with installed Chrome version)
- **HTML Parsing:** BeautifulSoup4
- **Additional Tools:**
    - time, random (for rate-limiting)
    - user-agent-rotator / Custom user-agent pool

    - logging module (for debugging and monitoring)

Frontend Technologies

- **HTML5** – Structure of the web application pages
- **CSS3** – Styling and responsive layout
- **Jinja2 Templating Engine** – Dynamic HTML generation from Flask

Browser Requirements

- **Minimum:** Google Chrome (v90+)
- **Recommended:** Google Chrome (v115+ with developer tools enabled)
- **Extensions (optional for debugging):**
    - XPath Helper
    - JSON Viewer
    - Live Server

4.3 Tools & Technology Stack Overview

Category	**Tools / Technology**	**Purpose**
Programming Language	Python 3.8+	Backend logic and scraping
Framework	Flask	Web application framework
Scraping Library	Selenium, BeautifulSoup4	Dynamic content scraping and parsing
Data Handling	Pandas, openpyxl, CSV, JSON	Structuring, processing and exporting data
UI Development	HTML, CSS, Bootstrap	Frontend design and responsiveness
Authentication	Flask-Login, Werkzeug	Secure user access and password handling
Database	SQLite3	Lightweight data storage (auth system)
IDE	Visual Studio Code	Writing and debugging code
Version Control	Git, GitHub	Code management
Testing	Manual Testing, Logging	Validating scraping and UI behavior
Deployment (optional)	Localhost or WSGI Server (Gunicorn)	Local or production deployment

Table 1: Technology stack overview

4.4 Assumptions

- The system will be deployed on a local machine, not on a remote server.
- Chrome browser and matching ChromeDriver are pre-installed.
- Python and required libraries are properly set up using pip.
- Internet connection is available throughout usage for real-time data scraping.

4.5 In Summary

The hardware and software requirements listed above ensure that the application functions smoothly and efficiently. Since the application relies on real-time scraping from Amazon and YouTube, it is designed to be lightweight, cross-platform compatible, and scalable for future extensions. With minimal system requirements, even users with standard configurations can run this application without performance bottlenecks.

5. PROCESS MODEL

A software process model provides a structured approach for planning, executing, and managing software development. For this project focused on web scraping and data extraction from Amazon and YouTube—an effective and structured model is necessary to manage complexity, maintain modularity, and ensure reliability.

Given the need for rapid prototyping, constant validation, and incremental improvement, the most appropriate model for this project is the Incremental Process Model combined with elements of Prototyping and Agile practices.

5.1 Incremental Process Model Overview

The Incremental Model breaks the project into smaller, manageable modules that are developed and tested independently. Each increment builds upon the previous one, adding functionality and refining features. This model is particularly effective for projects like ours where components such as scrapers (Amazon, YouTube), authentication, dashboard UI, and export features are distinct yet interconnected.

Key Features of Incremental Model in Our Project

- Modular development of Amazon and YouTube scrapers.
- Independent testing of components.
- Early delivery of a working product (basic MVP).
- Flexible integration of user feedback and debugging.
- Parallel development of frontend and backend modules.

5.2 Increments Used in the project

Increment	Module/Component	Description
1	**User Authentication System**	Signup/Login functionality using Flask, SQLite, hashed passwords.
2	**Amazon Scraper**	Multi-page scraping with Selenium, data extraction, export.
3	**YouTube Scraper**	Video metadata extraction, pagination, export capability.
4	**Frontend Dashboard**	Flask-based UI, Jinja2 templates, navigation bar, status messages.
5	**Data Export & Download System**	CSV, Excel, and JSON export functionality with organized file storage.

6	**Logging and Error Handling**	Comprehensive backend error logging, rate-limiting, anti-blocking features.
7	**Educational Blog**	Informational content for user learning

Table 2: Increments Used in the project

5.3 Prototyping for UI and Workflow

Before the actual development, UI mockups were created to visualize:

- Dashboard layout
- Button placement for scrapers
- Export download options
- Login/Signup flow
- Scraping progress indicator

5.4 Agile Elements Applied

Although not formally using Agile sprints, some Agile principles were naturally incorporated:

- Continuous Integration: Regular merging of code after each increment.
- Frequent Testing: Each scraper was tested after small changes.
- Iterative Feedback: UI and scraping logic were refined after practical testing.

5.5 Reason for Choosing This Model

Criteria	**Justification**
Project Complexity	Dividing tasks into increments simplified development.
Independent Functional Modules	Amazon and YouTube scrapers were ideal candidates for isolated development.
Early Testing & Validation	Prototyping enabled real-time verification of scraping results.
User Experience Focus	Agile-style iterations helped in refining UI and UX.
Risk Reduction	Bugs were isolated within modules instead of crashing the whole system.

Performance Optimization	Modular testing allowed targeted performance improvements.

Table 3: Reason For Choosing Model

5.6 Summary of Development Flow

1. **Requirement Analysis**
   - Identify target platforms (Amazon, YouTube)
   - Define user roles, scraping targets, and export expectations
2. **Planning**
   - Breakdown of work into increments
   - Timeline mapping (3-month plan)
3. **Prototype Design**
   - UI sketches for dashboard, login, export, error screens
4. **Incremental Implementation**
   - Build modules one by one
   - Integrate after unit testing
5. **Validation and Testing**
   - Perform functional, UI, and export testing
   - Apply rate-limiting and anti-blocking strategies
6. **Deployment and User Testing**
   - Run on localhost
   - Final polish of UI and error messages
7. **Documentation and Wrap-up**
   - Prepare user manual, report, and code cleanup

This structured Incremental + Prototyping approach not only ensured systematic development but also improved code maintainability, reusability, and overall robustness of the web scraping system.

6. PROJECT PLAN

A project plan is essential to ensure the timely and efficient execution of a software development project. It outlines the sequence of major activities, their estimated durations, and dependencies. Below is a comprehensive week-by-week plan tailored to your web scraping project using Python, Flask, Selenium, and supporting technologies.

Week 1: Requirement Gathering & Research

- Understand the requirements and constraints of the project.
- Research Amazon and YouTube website structures, scraping policies, and potential challenges.
- Study existing scraping tools and methods.
- Define project scope and target features.
- Prepare the initial Software Requirements Specification (SRS) document.

Week 2: Environment Setup & Tool Installation

- Install Python, Flask, Selenium, Chrome WebDriver.
- Set up version control (Git).
- Create project folder structure.
- Install supporting libraries: BeautifulSoup4, Pandas, openpyxl, etc.
- Initialize Flask app with routes and templates.

Week 3: Design UI Prototypes

- Create UI mockups for:
    - Login/Signup pages
    - Dashboard
    - Scraper buttons
    - Blog/news layout
- Finalize HTML template structure using Jinja2.
- Style UI with basic CSS (static/styles.css).
- Get feedback and refine design.

Week 4: Develop Authentication System

- Implement signup and login using Flask and SQLite.
- Apply password hashing with Werkzeug.
- Integrate session management and access control.
- Validate input and implement error handling.

Week 5: Build Amazon Scraper – Core Logic

- Implement Selenium-based scraping for:
    - Title
    - Price
    - Rating
    - Review count
    - Delivery info
    - Deal status and sponsorship
- Add multi-page support and pagination handling.

Week 6: Amazon Scraper – Export & Rate Limiting

- Add export functionality (CSV, Excel, JSON).
- Organize file output in structured directories.
- Implement random delays, user-agent rotation, and basic anti-detection mechanisms.
- Write test cases for scraping reliability.

Week 7: Build YouTube Scraper

- Develop Selenium script for scraping:
    - Video titles
    - Views
    - Upload dates
- Handle pagination through scrolling or page navigation.
- Ensure data consistency and export support.

Week 8: Backend Integration of Scrapers

- Link scrapers to Flask backend routes.
- Create buttons for each scraper on the dashboard.
- Connect front-end buttons to trigger Python scripts securely.
- Display "scraping in progress" indicators.

Week 9: Implement Download & Progress System

- Store scraped files in temporary locations.
- Show download links after scraping completion.
- Display progress bar or message.
- Handle scraping errors and display user-friendly messages.

Week 10: Add Blog & Logging

- Add optional blog page or educational content (HTML only).
- Add structured logging for scraper results, errors, and status.

Week 11: Testing & Optimization

- Perform end-to-end testing of:
    - Authentication
    - Scraping
    - Export and UI
- Conduct performance testing with multiple pages/videos.
- Fix bugs, validate edge cases, and fine-tune selectors.
- Optimize memory and CPU usage.

Week 12-14: Documentation

- Prepare final project report.
- Final testing and polish.

7. SYSTEM DESIGN

7.1 Diagrams

7.1.1 Use Case Diagram

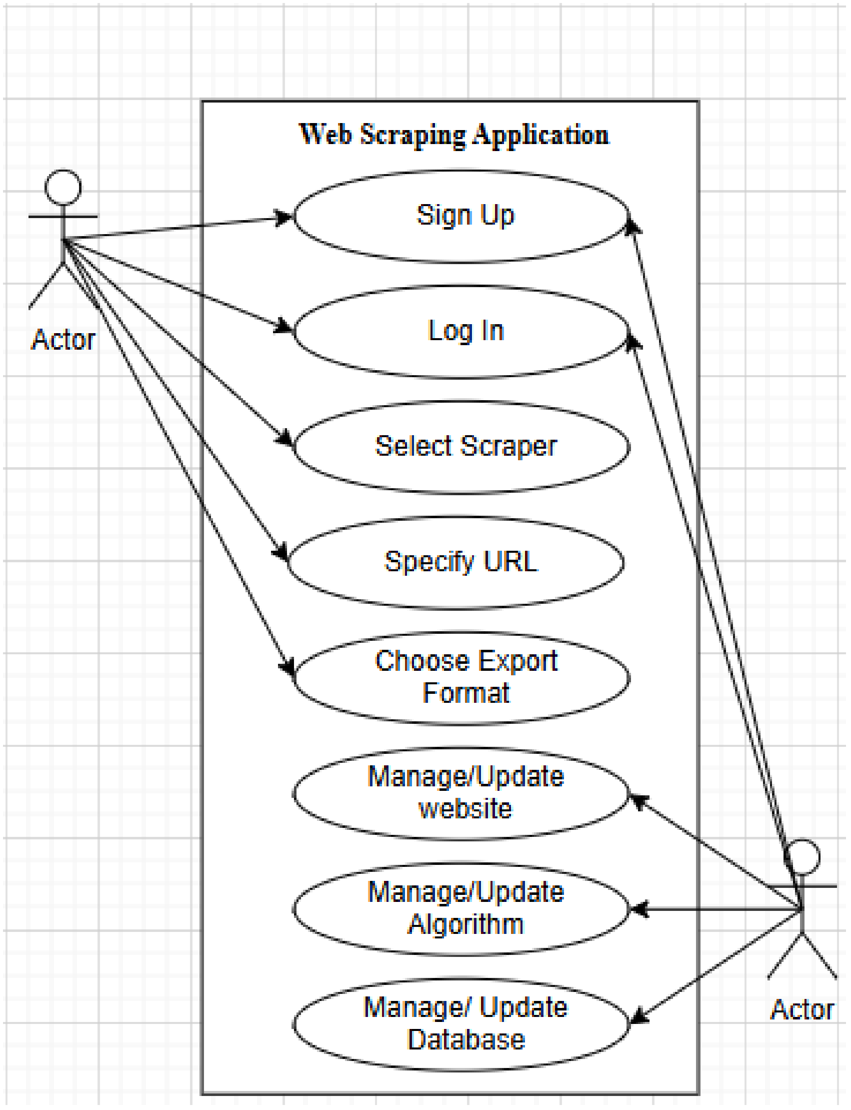


Figure 1: Use Case Diagram

7.1.2 Class Diagram

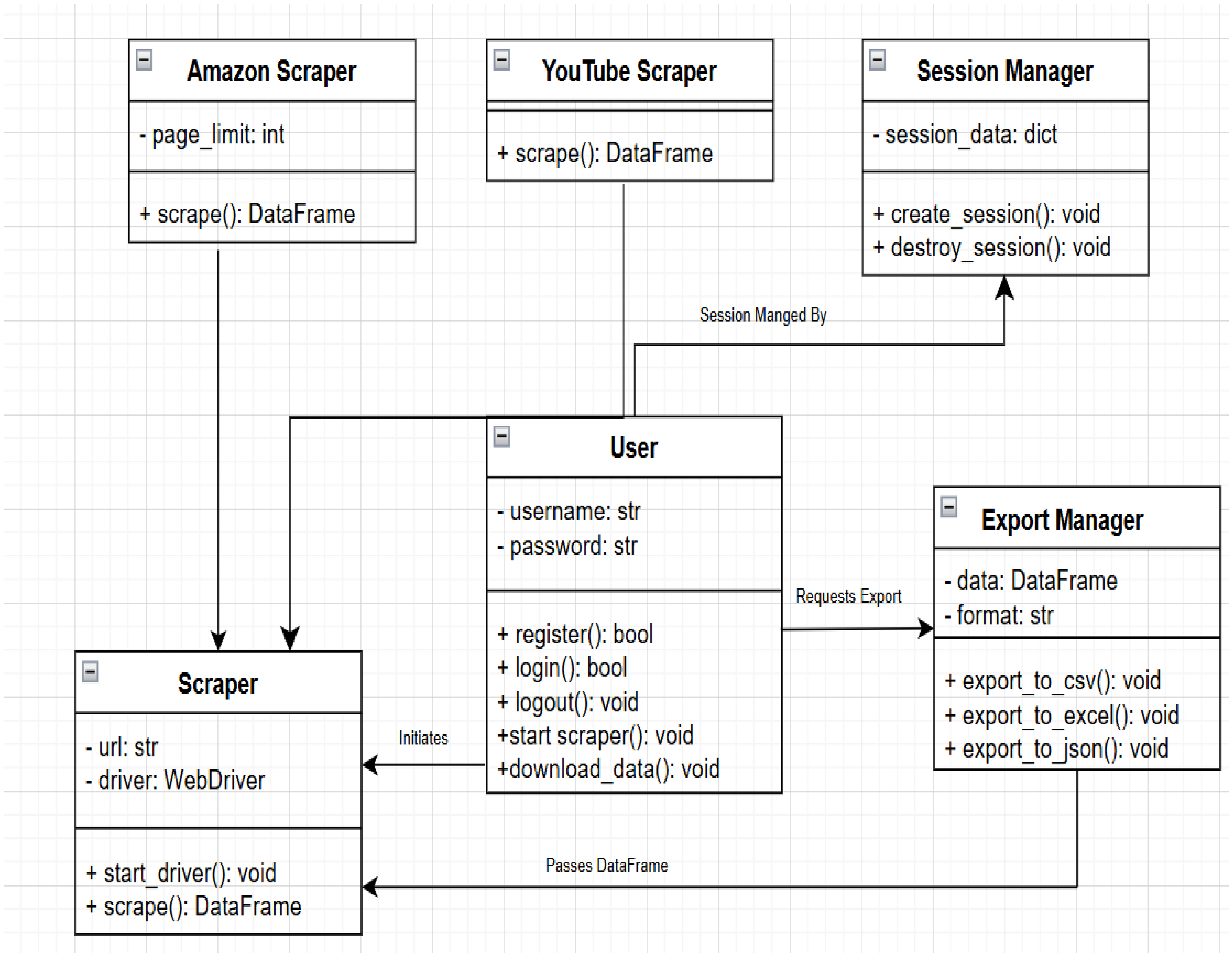


Figure 2: Class Diagram

7.1.3 Activity Diagram

7.1.3.1 User Activity Diagram

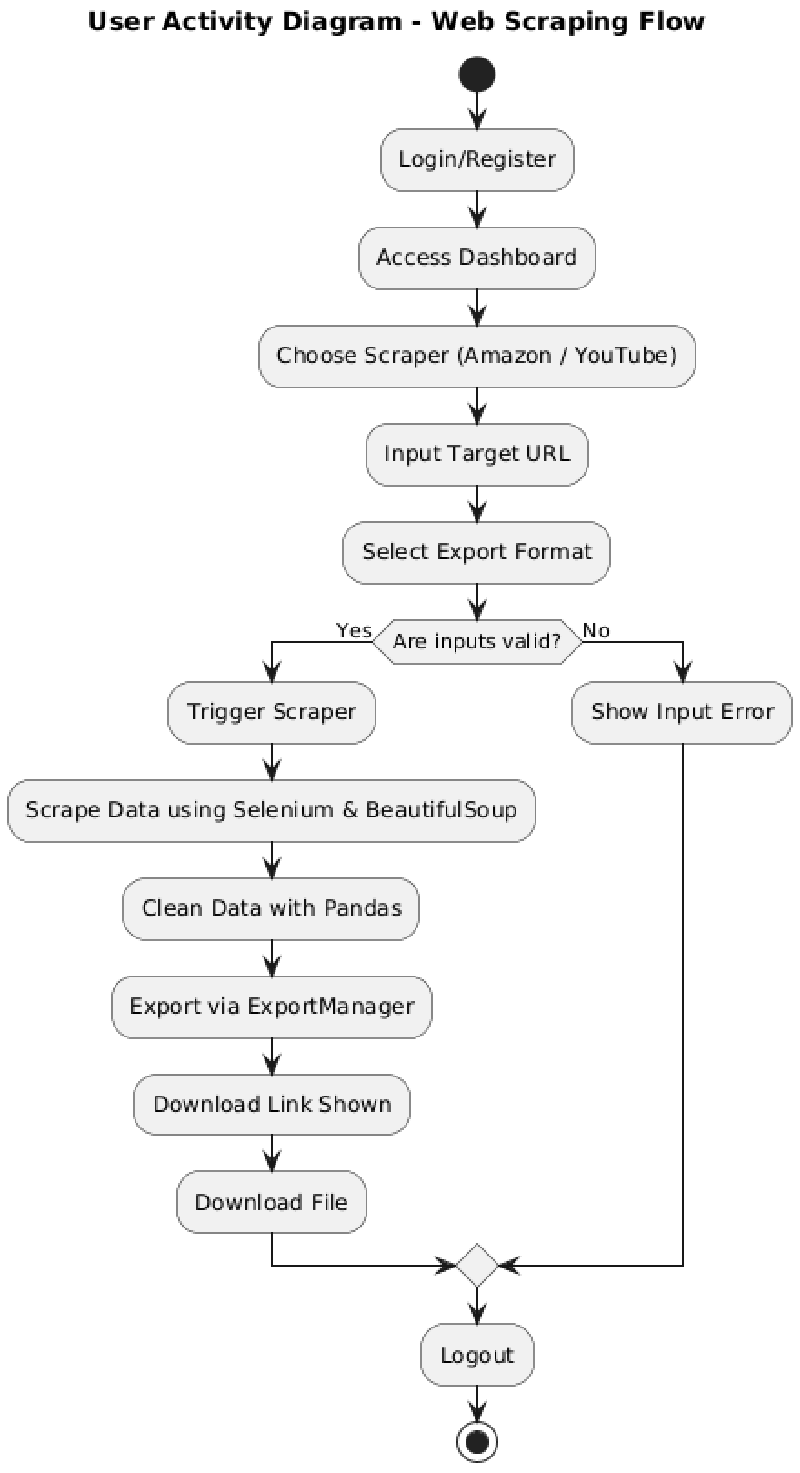
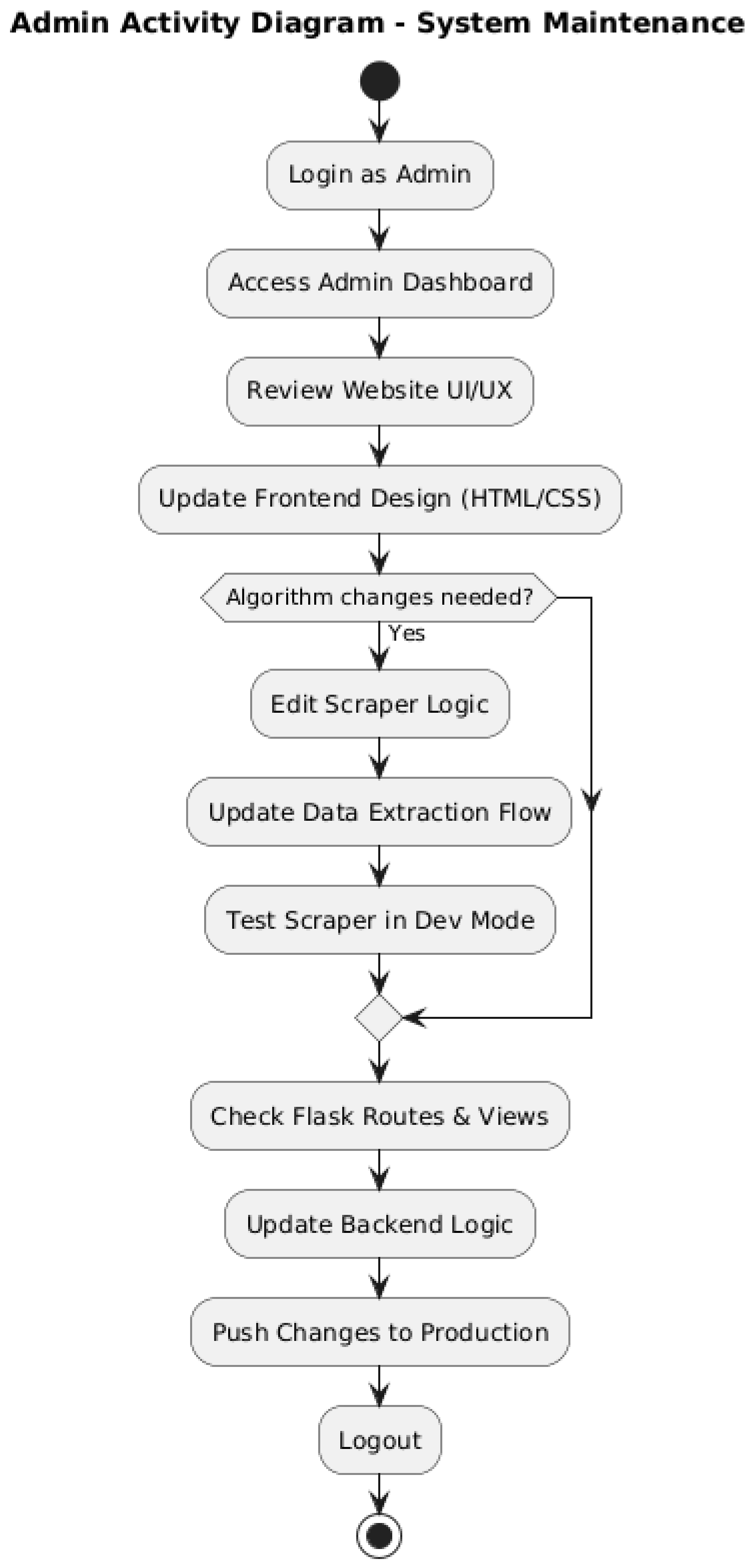


Figure 3: User Activity Diagram

7.1.3.2 Admin Activity



Admin Activity Diagram - System Maintenance
Login as Admin
Access Admin Dashboard
Review Website UI/UX
Update Frontend Design (HTML/CSS)
Algorithm changes needed?
Yes
Edit Scraper Logic
Update Data Extraction Flow
Test Scraper in Dev Mode
Check Flask Routes & Views
Update Backend Logic
Push Changes to Production
Logout

Figure 4: Admin Activity Diagram

7.1.4 Sequence Diagram

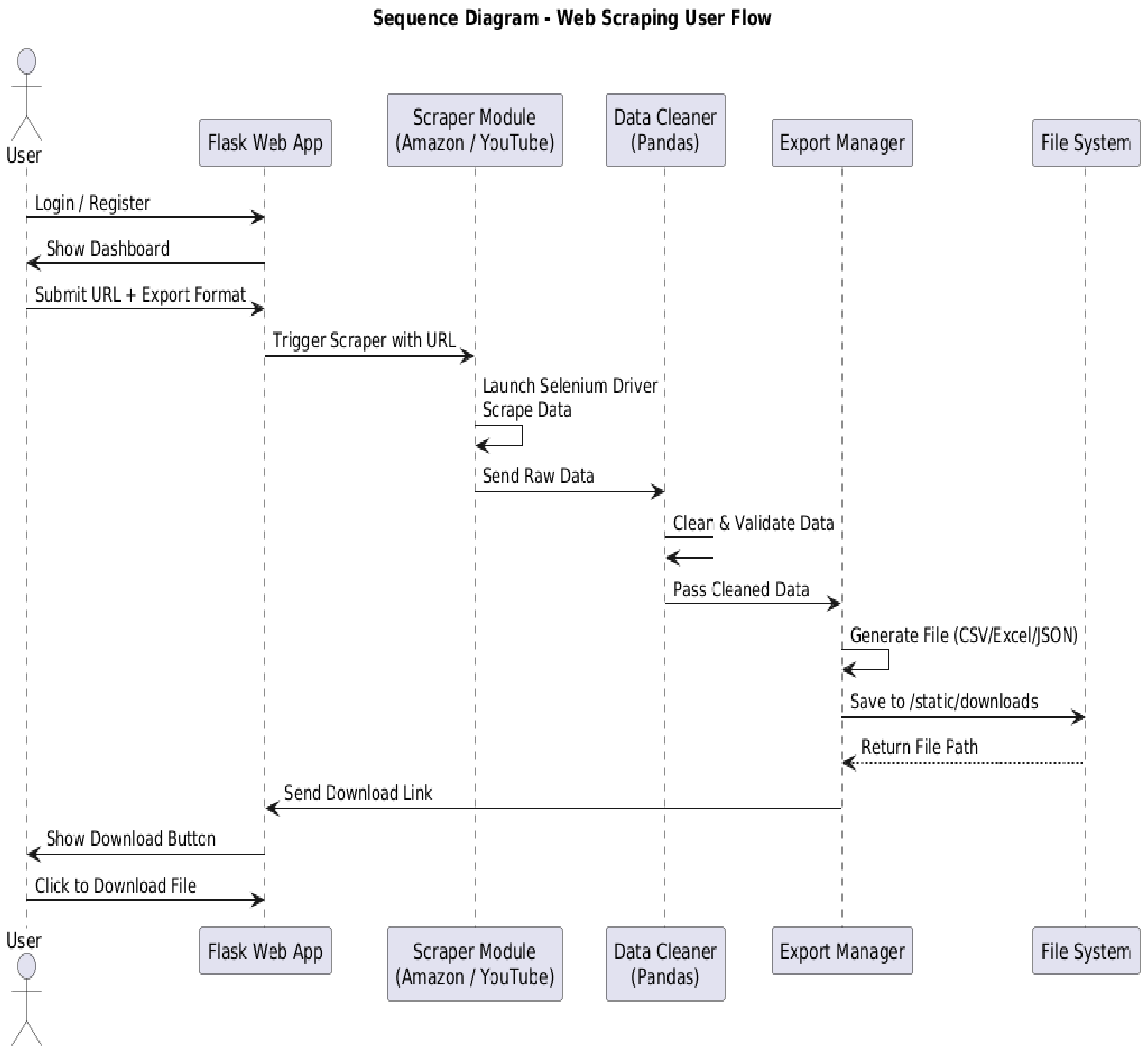


Figure 5: Sequence Diagram

7.1.5 Data Flow Diagrams

7.1.5.1 DFD Level 0

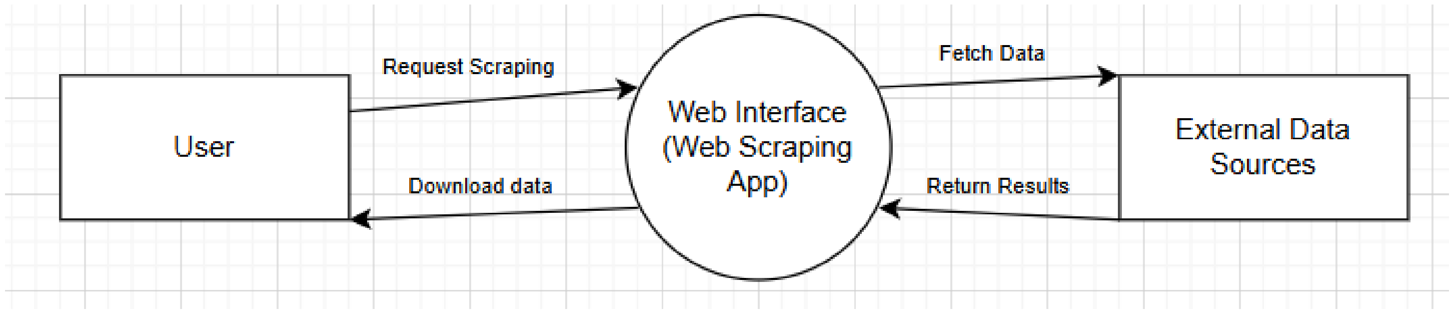


Figure 6: DFD Level 0

7.1.5.2 DFD (Level 1)

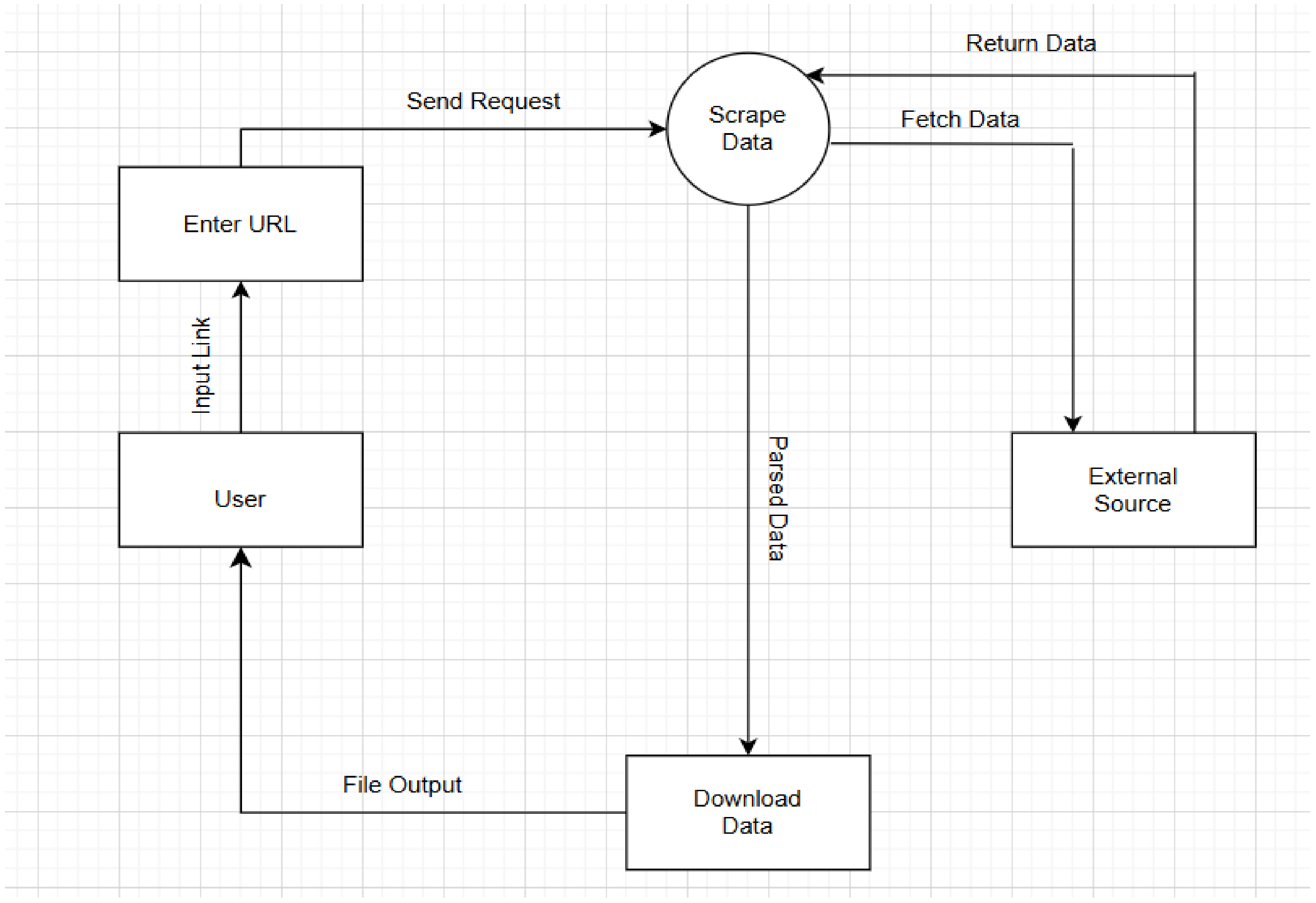


Figure 7: DFD Level 1

7.1.5.3 DFD Level 2

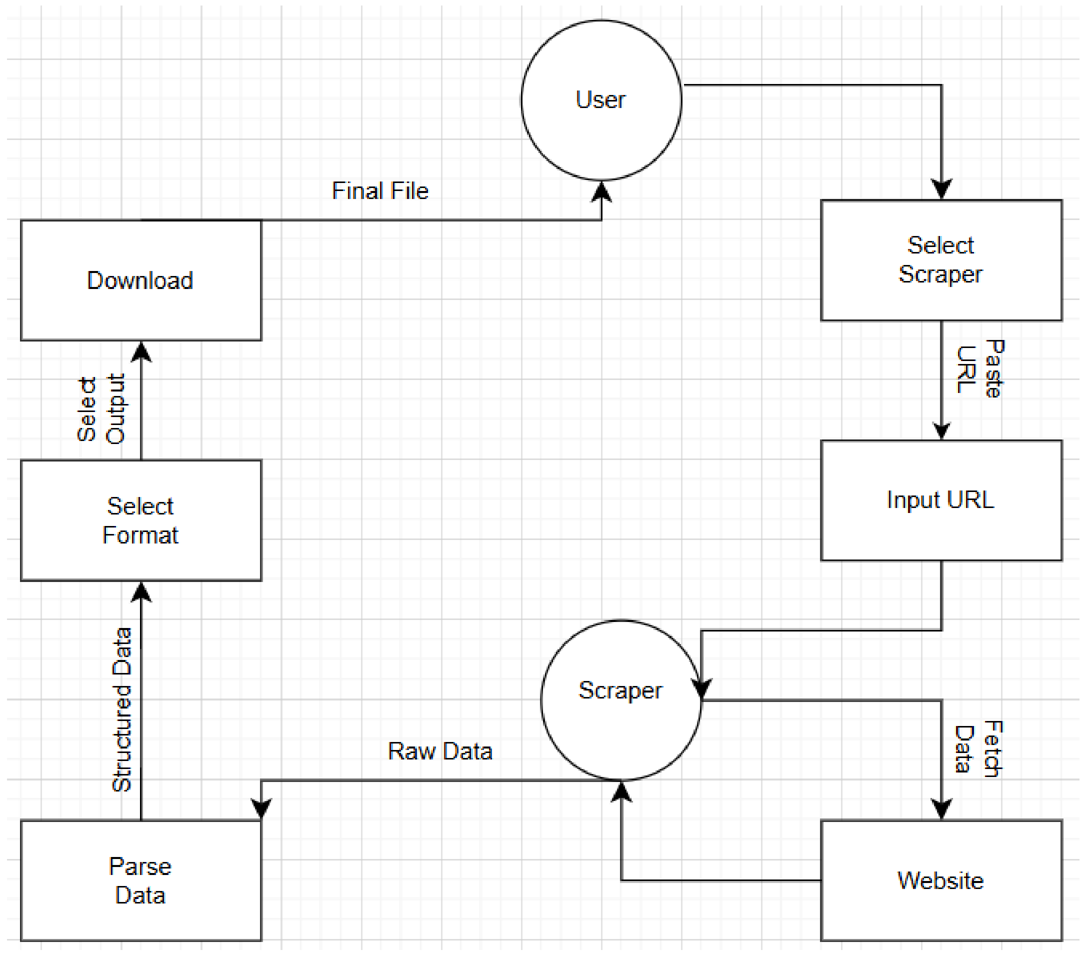


Figure 8: DFD Level 2

7.2 Tables & Data Dictionary

7.2.1 List Of Tables

Table Name	Description
Users	Stores user registration and authentication data
scraping_tasks	Stores metadata about each scraping task submitted
Scraped_data	Stores reference to scraped data files and their metadata
Logs	Stores system logs and scraping activity logs

Table 4: List of Tables

7.2.2 Users Table

Column Name	Data Type	Constraints	Description
user_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for the user
username	TEXT	UNIQUE, NOT NULL	User's chosen username
password_hash	TEXT	NOT NULL	Hashed password using Werkzeug
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Account creation timestamp

Table 5: Users Table

7.2.3 Scraping Tasks Table

Column Name	Data Type	Constraints	Description
task_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique task identifier
user_id	INTEGER	FOREIGN KEY (users.user_id)	User who created the task
platform	TEXT	NOT NULL	Either 'Amazon' or 'YouTube'
input_url	TEXT	NOT NULL	The URL provided by the user
output_format	TEXT	NOT NULL	Chosen format: CSV, Excel, or JSON
status	TEXT	NOT NULL, DEFAULT 'Pending'	Task status (Pending, In Progress, Completed, Failed)
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Task creation time
completed_at	DATETIME	NULL	Task completion time

Table 6: Scraping Tasks Table

7.2.4 Scraped Data Table

Column Name	Data Type	Constraints	Description
data_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier
task_id	INTEGER	FOREIGN KEY (scraping_tasks.task_id)	Related scraping task
file_path	TEXT	NOT NULL	Path to the exported data file
file_format	TEXT	NOT NULL	Format of the file (CSV, Excel, JSON)
generated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	File generation timestamp

Table 7: Scraped Data Table

7.2.5 Logs Table

Column Name	Data Type	Constraints	Description
log_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each log
user_id	INTEGER	FOREIGN KEY (users.user_id)	Related user (if applicable)
action	TEXT	NOT NULL	Action description (Login, Scraping Started, etc.)
timestamp	DATETIME	DEFAULT CURRENT_TIMESTAMP	Time of log entry
details	TEXT	NULL	Optional detailed log message

Table 8: Logs Table

7.2.6 Data Dictionary

Table Name	Column Name	Description	Example
Users	user_id	Unique identifier of a user	1
Users	username	Username chosen by the user	johndoe
Users	password_hash	Hashed password string	pbkdf2:sha256$…
scraping_tasks	platform	Platform to scrape from	Amazon
scraping_tasks	input_url	URL to be scraped	https://amazon.in/s?k=laptops
scraping_tasks	output_format	Desired output file format	CSV
scraping_tasks	status	Task status	Completed
scraped_data	file_path	Location of stored scraped file	/outputs/task1_data.csv
Logs	action	Recorded user/system action	Scraping Started

Table 9: Data Dictionary

7.3 Design Strategy

7.3.1 Overview

For the development of our Web Scraping Application for Amazon and YouTube, we have adopted a 3-tier architecture design strategy. This architecture pattern separates the application into three distinct layers Presentation Layer, Application (Logic) Layer, and Data Layer each responsible for handling specific tasks. By following this modular structure, we ensure high scalability, better maintainability, clear separation of concerns, and enhanced security.

The three-tier architecture perfectly aligns with the complexity and extensibility required by our system, as it integrates:

- A web-based front-end interface for user interaction.
- A backend business logic layer to handle scraping tasks, user sessions, and processing.
- A data management layer to manage authentication, scraping task metadata, and file storage.

7.3.2 Layers of 3 -Tier Architecture

Presentation Layer (User Interface)

Responsibilities:

- Manages all interactions between the end user and the application.
- Accepts user inputs such as URLs to scrape, platform selections, and export format choices.
- Displays task progress, error messages, scraped data download links, and other relevant feedback.

Technologies used:

- **HTML, CSS** → Layout, styling, and responsive design.
- **Jinja2 Templates** (Flask templating engine) → Dynamic rendering of web pages.
- **Bootstrap (optional)** → For responsive and clean UI components.
- **JavaScript** (optional) → For improved interactivity (loading indicators, form validation, etc.).

Key Components:

- Login & Signup Pages
- Dashboard (Task Management Interface)
- Blog Section (Educational Content)
- Scraping Progress & Result Pages

Application Layer (Business Logic Layer)

Responsibilities:

- Acts as the core of the system, processing all the business rules and logic.
- Handles user authentication and session management.
- Manages initiation and monitoring of scraping tasks (Amazon or YouTube).
- Facilitates secure data export and file management.
- Communicates between the UI and the database.

Technologies used:

- **Python (Flask)** → Web framework for handling routing, sessions, and server-side processing.
- **Selenium WebDriver** → For dynamic content scraping.
- **BeautifulSoup4** → For parsing HTML and extracting data.

- **Werkzeug** → For password hashing and secure session handling.
- **Pandas** → For data cleaning, processing, and exporting in multiple formats.
- **Logging** (Python built-in) → For system activity and error logging.

Key Components:

- Scraper Modules (amazon_scraper.py, youtube_scraper.py)
- Authentication System (app.py)
- Task management and data export handlers
- Error handling and security utilities

Data Layer (Data Management Layer)

Responsibilities:

- Handles all data-related operations including storage, retrieval, and updates.
- Manages user authentication data, scraping task metadata, and file paths for exported data.
- Ensures data integrity, persistence, and secure access control.

Technologies used:

- **SQLite3 Database** → Lightweight RDBMS embedded within the application.
- **File System** → For storing scraped data files (CSV, Excel, JSON) in organized folders.
- **SQLAlchemy ORM (optional)** → For easier database manipulation in Python (if implemented).

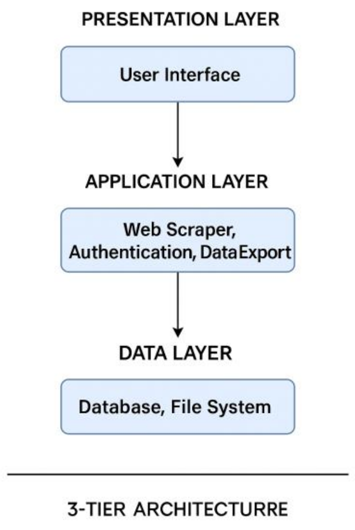


Figure 9: 3 – Tier Design Strategy Architecture

Key Components:

- users table (user accounts and hashed passwords)
- scraping_tasks table (task metadata)
- scraped_data table (scraped data file references)
- logs table (activity logs)

7.3.3 Benefits of Using 3 – Tier Architecture

Advantage	Explanation
Scalability	Each layer can be independently scaled to handle more users, tasks, or data.
Maintainability	Clear separation of code improves debugging, testing, and feature enhancements.
Security	Sensitive operations (like authentication and file management) are handled server-side, limiting client exposure.
Reusability	Logic components and UI templates can be reused across multiple parts of the application.
Flexibility	Easy to switch technologies within layers (e.g., migrate from SQLite to PostgreSQL).

Table 10: Benefits of Using 3 – Tier Architeture

8. IMPLEMENTATION DETAILS

This section elaborates on the comprehensive implementation strategy used in building the proposed web scraping application system. The project is designed to offer a robust, scalable, and secure solution for automated data extraction from two popular platforms Amazon and YouTube. The system integrates powerful backend scraping modules with a modern web interface, providing users with a seamless experience from login to data retrieval and download. Below, we describe in detail the various algorithms, implementation approaches, and workflows that underpin the system.

8.1 Algorithms

8.1.1 Overall System Workflow Algorithm

The overall workflow of the system is methodically structured to ensure smooth interaction between the user interface and backend processing modules. The system starts with user authentication, followed by task selection, scraping operations, data processing, and finally, data export and download.

1. The system initiates when the user accesses the web application hosted on the Flask server.
2. The user is prompted to register or log into the system using their credentials.
3. Upon successful login, the user is redirected to the dashboard, where they can choose between scraping data from Amazon or YouTube.
4. The user selects the desired scraping module and inputs the corresponding URL an Amazon search result link or a YouTube channel URL.
5. The user specifies the preferred export format for the scraped data, choosing from CSV, Excel, or JSON.
6. The system performs input validation to ensure the URLs and formats are correct.
7. The relevant scraping module (Amazon or YouTube) is triggered.
8. The scraper launches a headless Selenium WebDriver and navigates through the respective website to collect the required data.
9. The scraped data is structured and cleaned using Pandas to ensure consistency and accuracy.
10. The cleaned data is passed to the Export Manager, which handles data export in the user-selected format.

11. Upon successful export, the download link is generated and presented to the user.
12. The user downloads the output file.
13. The system performs housekeeping activities, cleaning up session data and removing old files to optimize storage.
14. The session can be terminated when the user logs out or exits the application.

8.1.2 Amazon Scraper Algorithm

The Amazon Scraper module is engineered to traverse multiple pages of Amazon search results while extracting a rich set of product attributes. The scraper integrates anti-blocking mechanisms such as randomized delays and user-agent rotation to minimize the risk of detection by Amazon's security systems.

1. The module accepts the Amazon search URL and the number of pages to scrape.
2. It initializes the Selenium WebDriver in headless mode to run without rendering a visible browser window.
3. A random user-agent string is applied, and randomized time delays between actions are introduced to mimic human-like interaction.
4. For each page in the defined page range:
    - The search results page is loaded.
    - The HTML content of the page is parsed using BeautifulSoup.
    - Each product block is located, and the following attributes are extracted:
        - Product Title
        - Price
        - Rating
        - Number of Reviews
        - Delivery Information
        - Product URL
        - Sponsored Status (whether the product is an ad)

        - Deal Status (if a discount or special deal is applied)

    - The scraper checks if a "Next" button is available and clicks it to proceed to the next page.

5. All extracted product details are consolidated into a Pandas DataFrame.
6. The DataFrame is returned to the Export Manager for further processing.

8.1.3 Youtube Scraper Algorithm

The YouTube Scraper module is designed to capture video metadata from a specified YouTube channel. Since YouTube employs infinite scrolling to load video content dynamically, the scraper handles scrolling operations and pagination intelligently.

1. The module accepts the YouTube channel URL as input.
2. The Selenium WebDriver is initialized in headless mode.
3. The scraper navigates to the channel's "Videos" tab.
4. A scroll loop is executed to dynamically load more videos by scrolling the page multiple times until no additional content appears.
5. The fully loaded page source is parsed using BeautifulSoup.
6. For each video element on the page, the following details are extracted:
    - Video Title
    - View Count
    - Upload Date
    - Video URL
7. The extracted video metadata is stored in a Pandas DataFrame.
8. The DataFrame is forwarded to the Export Manager for export.

8.1.4 Data Export Algorithm

The system provides flexible data export capabilities to cater to diverse user requirements. The Export Manager handles conversion of the processed DataFrame into user-specified formats efficiently.

1. The module receives the DataFrame and the selected export format from the user.
2. Based on the selected format:
    - For CSV: The Pandas to_csv() method is used to generate a .csv file.
    - For Excel: The Pandas to_excel() method creates an Excel spreadsheet (.xlsx).
    - For JSON: The Pandas to_json() method is invoked to output a JSON file.
3. The generated file is saved according to your browser download settings.
4. The path to the generated file is returned to the Flask application.
5. A download link is rendered on the dashboard, allowing the user to download the file.

8.1.5 User Authentication and Session Management Algorithm

Ensuring secure user authentication and reliable session management is critical to safeguarding user data and controlling access to the application's functionalities.

1. During registration:
    - The user submits a username, email, and password.
    - The password is hashed using Werkzeug's security module.
    - The hashed password, along with the username and email, is stored in the SQLite database.
2. During login:
    - The system verifies if the entered username exists.
    - The entered password is hashed and compared with the stored hash.
    - Upon successful verification, session variables are created to track the logged-in user.
3. During logout:
    - Session variables are cleared.
    - The user is redirected to the login page.

8.2 Flowcharts

8.2.1 User Authentication

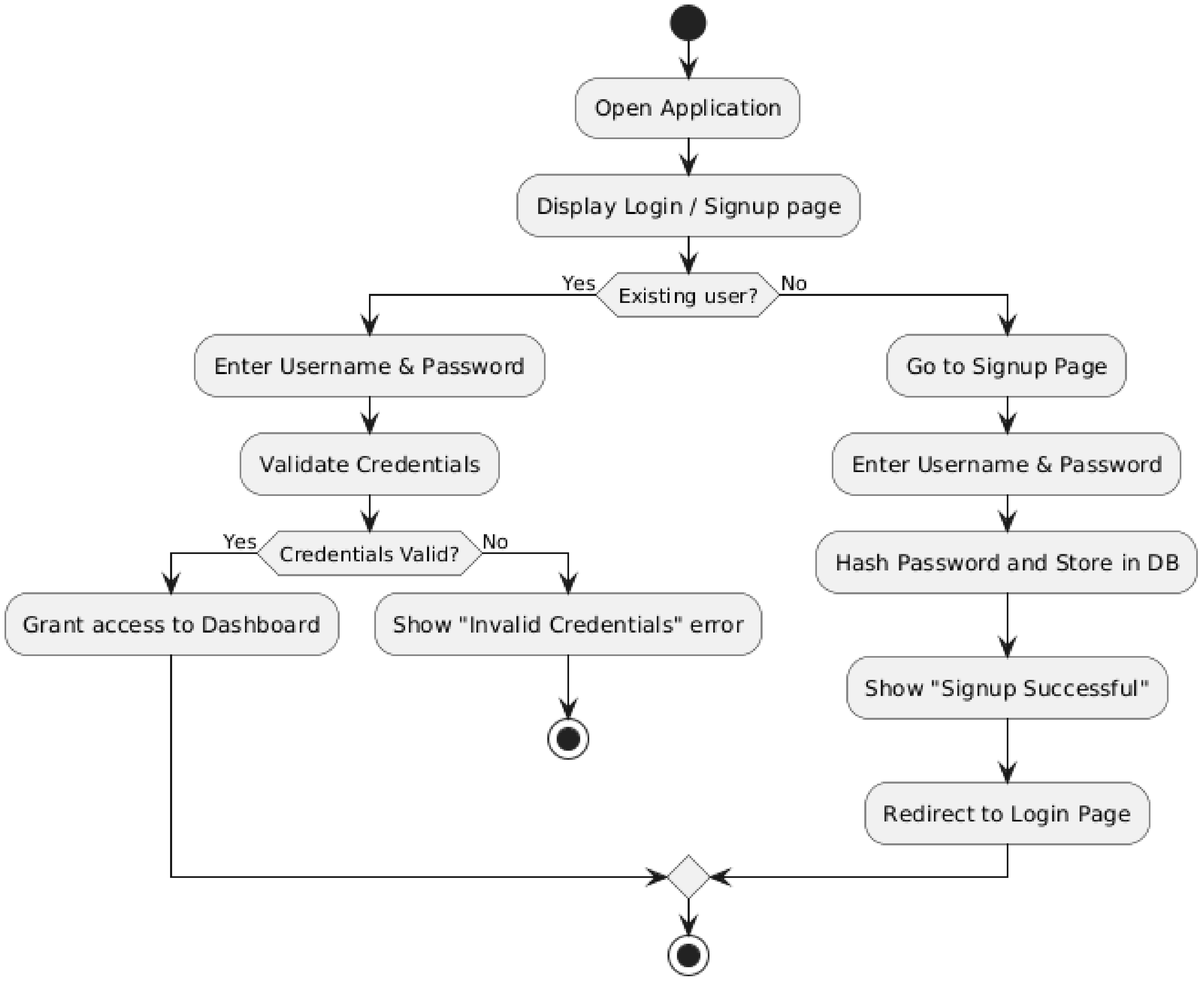

Open Application
Display Login / Signup page
Existing user?
Yes
No
Enter Username & Password
Validate Credentials
Credentials Valid?
Yes
No
Grant access to Dashboard
Show "Invalid Credentials" error
Go to Signup Page
Enter Username & Password
Hash Password and Store in DB
Show "Signup Successful"
Redirect to Login Page

Figure 10: User Authentication Flow chart

8.2.2 Amazon Scraper

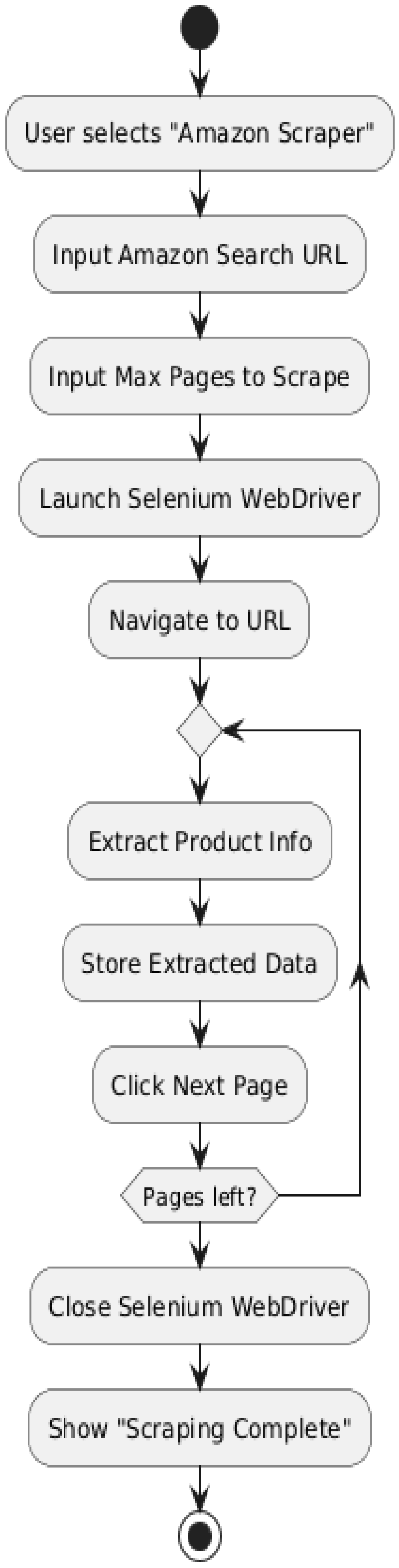


Figure 11: Amazon Scraper Flowchart

8.2.3 Youtube Scraper

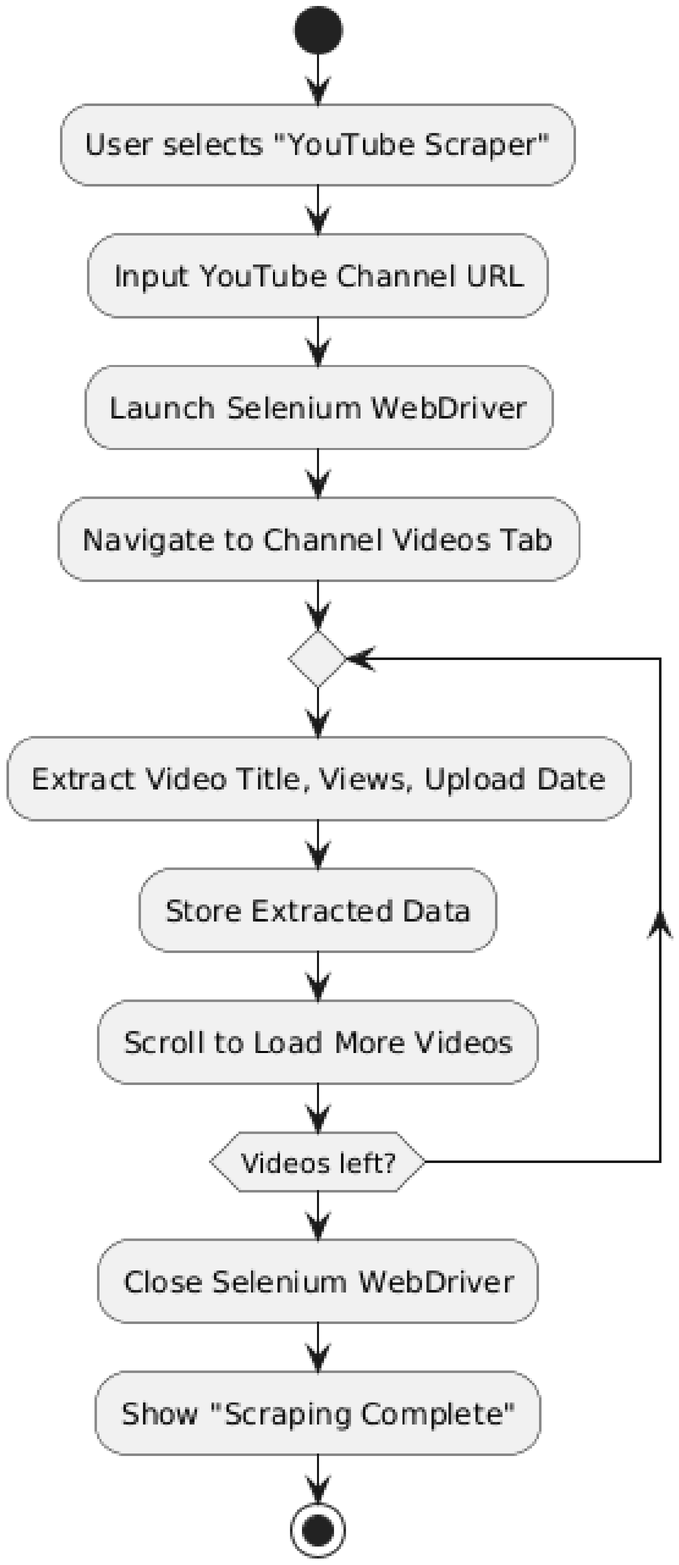

User selects "YouTube Scraper"
Input YouTube Channel URL
Launch Selenium WebDriver
Navigate to Channel Videos Tab
Extract Video Title, Views, Upload Date
Store Extracted Data
Scroll to Load More Videos
Videos left?
Close Selenium WebDriver
Show "Scraping Complete"

Figure 12: YouTube Scraper Flow Chart

8.2.4 Data Export

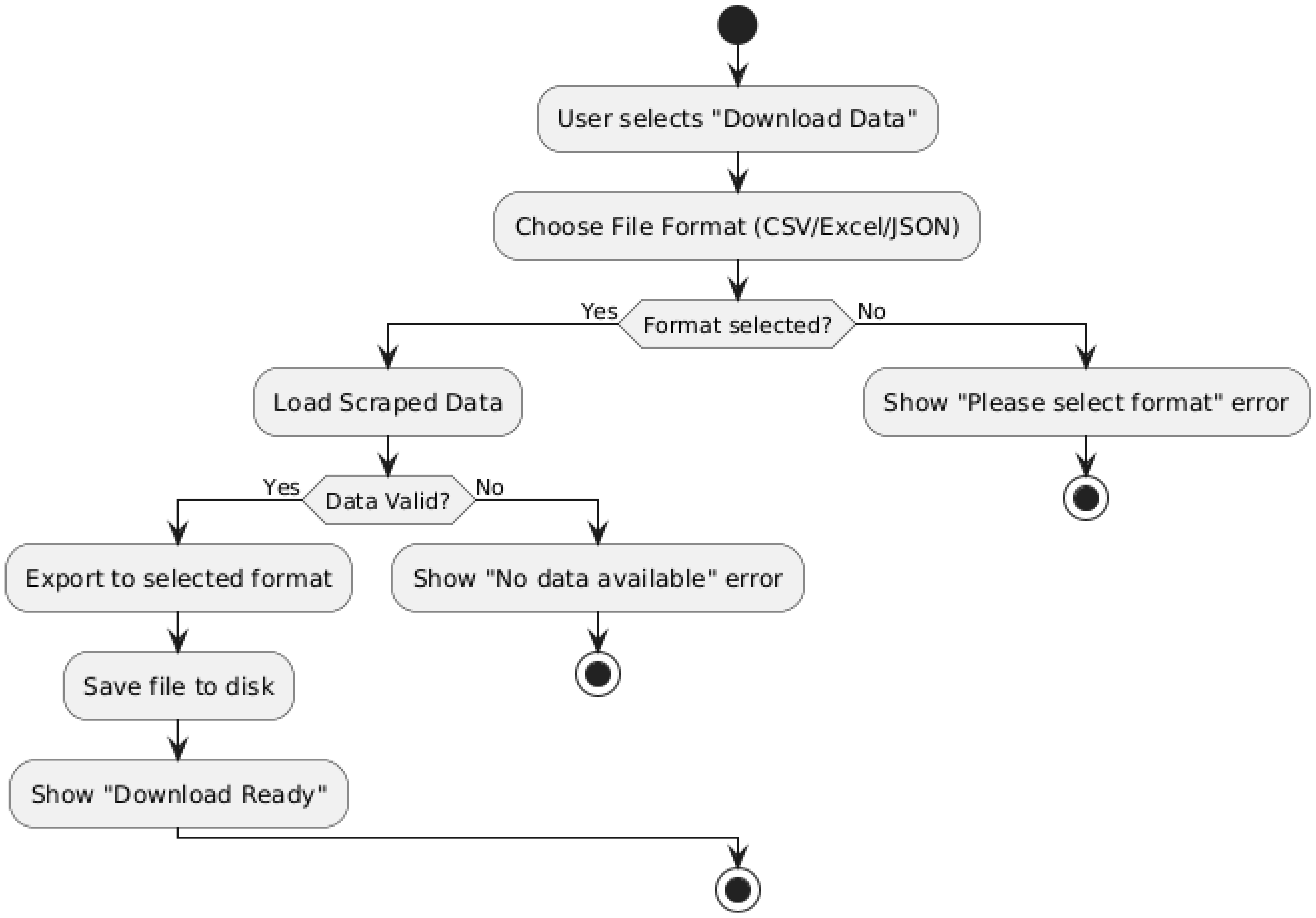

User selects "Download Data"
Choose File Format (CSV/Excel/JSON)
Format selected?
Yes
No
Load Scraped Data
Show "Please select format" error
Data Valid?
Yes
No
Export to selected format
Show "No data available" error
Save file to disk
Show "Download Ready"

Figure 13: Data Export Flowchart

8.3 Program Codes

8.3.1 Amazon Scraper

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from bs4 import BeautifulSoup
import time
import random
from urllib.parse import urljoin, urlparse
import logging
from dataclasses import dataclass
from typing import List, Optional, Tuple
import csv
import os
import pandas as pd
import json

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)
```

```
@dataclass
class Product:
    title: str
    price: str
    rating: Optional[str]
    reviews: Optional[str]
    delivery: Optional[str]
    url: str
    sponsored: bool = False
    deal: Optional[str] = None

class AmazonScraper:
    def __init__(self, visible_browser=False):
        self.base_url = "https://www.amazon.in"
        self.visible_browser = visible_browser
        self.driver = self._init_driver()
        self.delay_range = (1, 3)

    def _init_driver(self):
        options = Options()
        if not self.visible_browser:
        options.add_argument("--headless")
        options.add_argument("--disable-gpu")
        options.add_argument("--no-sandbox")
        options.add_argument("--window-size=1920,1080")
        options.add_argument("--log-level=3")  # Suppress WebGL warnings
        options.add_argument("--silent")
        options.add_experimental_option('excludeSwitches', ['enable-logging'])
        options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
        driver = webdriver.Chrome(options=options)
        return driver

    def _random_delay(self):
        delay = random.uniform(*self.delay_range)
        time.sleep(delay)

    def _validate_amazon_url(self, url: str) -> bool:
        parsed = urlparse(url)
        return parsed.netloc.endswith('amazon.in') or parsed.netloc.endswith('amazon.com')

    def _get_page(self, url: str) -> Optional[str]:
        try:
            self._random_delay()
            self.driver.get(url)

            WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located((By.CSS_SELECTOR, "div.s-result-item")))

            if "api-services-support@amazon.com" in self.driver.page_source:
                raise Exception("CAPTCHA detected - Amazon is blocking requests")

            return self.driver.page_source
        except Exception as e:
            logger.error(f"Error loading page {url}: {e}")
            return None
    def _extract_product_data(self, item) -> Optional[Product]:
        try:
            link = item.select_one('h2 a.a-link-normal, a.a-link-normal.s-line-clamp-2, a.a-link-normal.s-line-
clamp-3')
            if not link:
                return None

            raw_url = link.get('href', '')
            product_url = urljoin(self.base_url, raw_url.split('?')[0])
            title = link.get_text(strip=True)

            price_whole = item.select_one('span.a-price-whole')
            price = price_whole.get_text(strip=True).replace(',', '') if price_whole else None

            rating = item.select_one('span.a-icon-alt')
            rating = rating.get_text(strip=True).split()[0] if rating else None

            reviews = item.select_one('span.a-size-base[aria-label], span.a-size-base.s-underline-text')
            reviews = reviews.get_text(strip=True) if reviews else None

            delivery = item.select_one('span.a-color-base.a-text-bold')
            delivery = delivery.get_text(strip=True) if delivery else None

            deal = item.select_one('span.a-badge-text')
            deal = deal.get_text(strip=True) if deal else None

            sponsored = bool(item.select_one('span.a-color-secondary:contains("Sponsored"),
span:contains("Sponsored Ad")'))

            return Product(
                title=title,
                price=price,
                rating=rating,
                reviews=reviews,
                delivery=delivery,
                url=product_url,
                sponsored=sponsored,
                deal=deal
            )
        except Exception as e:
            logger.error(f"Error extracting product data: {e}")
            return None

    def _get_next_page_url(self, soup) -> Optional[str]:
        next_button = soup.select_one('a.s-pagination-next')
        if next_button and not 'a-disabled' in next_button.get('class', []):
            return urljoin(self.base_url, next_button['href'])
        return None

    def scrape_amazon(self, search_query: str, max_pages: int = 1) -> List[Product]:
        if not search_query:
            return []

        all_products = []
        current_page = 1
        current_url = search_query  # Use the full URL provided

        try:
            while current_url and current_page <= max_pages:
```

```
        logger.info(f"Scraping page {current_page}")
        html = self._get_page(current_url)
        if not html:
            break

        soup = BeautifulSoup(html, 'html.parser')
        items = soup.select('div.s-result-item[data-component-type="s-search-result"]')

        page_products = []
        for item in items:
            product = self._extract_product_data(item)
            if product:
                page_products.append(product)

        all_products.extend(page_products)
        logger.info(f"Found {len(page_products)} products on page {current_page}")

        # Get next page URL
        current_url = self._get_next_page_url(soup)
        current_page += 1

        if not current_url:
            break

        self._random_delay()

    # Save the results
    if all_products:
        # Ensure output directory exists
```

```
            os.makedirs('output', exist_ok=True)

            # Save as CSV
            df = pd.DataFrame([p.__dict__ for p in all_products])
            df.to_csv('output/amazon_products.csv', index=False)
            logger.info(f"Saved {len(all_products)} products to CSV")

            # Save as Excel
            df.to_excel('output/amazon_products.xlsx', index=False)
            logger.info(f"Saved {len(all_products)} products to Excel")

            # Save as JSON
            with open('output/amazon_products.json', 'w', encoding='utf-8') as f:
                json.dump([p.__dict__ for p in all_products], f, indent=2)
            logger.info(f"Saved {len(all_products)} products to JSON")

        return all_products

    except Exception as e:
        logger.error(f"Error during scraping: {e}")
        return []
    finally:
        self.close()

def close(self):
    self.driver.quit()
```

8.3.2 Youtube Scraper

```
import time
import random
import pandas as pd
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import (
    TimeoutException,
    NoSuchElementException,
    WebDriverException
)


def setup_driver():
    options = Options()
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--window-size=1920,1080')
    options.add_argument('--user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36')
    driver = webdriver.Chrome(options=options)
    return driver


def random_wait():
    wait_time = random.uniform(2, 5)
    time.sleep(wait_time)


def scrape_youtube_videos(url):
    if not url:
        return []


    driver = setup_driver()
    driver.get(url)
    random_wait()


    video_data = []
    scraped_videos = set()


    try:
        WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "contents")))


        while True:
            videos = driver.find_elements(By.XPATH, "//ytd-rich-item-renderer")


            for video in videos:
                try:
                    title_element = video.find_element(By.XPATH, ".//yt-formatted-string[@id='video-title']")
                    title = title_element.text
                    if title in scraped_videos:
                        continue


                    views = video.find_element(By.XPATH, ".//span[@class='inline-metadata-item style-scope ytd-
video-meta-block']").text
```

```
                upload_date = video.find_elements(By.XPATH, ".//span[@class='inline-metadata-item style-
scope ytd-video-meta-block']")[1].text

                video_info = {
                    "title": title,
                    "views": views,
                    "upload_date": upload_date
                }
                video_data.append(video_info)
                scraped_videos.add(title)
            except NoSuchElementException:
                continue

        last_height = driver.execute_script("return document.documentElement.scrollHeight")
        driver.execute_script("window.scrollTo(0, document.documentElement.scrollHeight);")
        random_wait()
        new_height = driver.execute_script("return document.documentElement.scrollHeight")

        if new_height == last_height:
            break

    except TimeoutException:
        print("Timed out waiting for page elements to load")
    finally:
        driver.quit()

    # Ensure output directory exists
    os.makedirs('output', exist_ok=True)
```

```
# Save data to Excel, CSV, and JSON
if video_data:
    df = pd.DataFrame(video_data)
    df.to_excel("output/youtube_videos.xlsx", index=False)
    df.to_csv("output/youtube_videos.csv", index=False)

return video_data
```

8.3.3 App.py

```
from flask import Flask, render_template, request, redirect, url_for, flash, session, send_file
import os
from werkzeug.security import generate_password_hash, check_password_hash
from scrapers.youtube_scraper import scrape_youtube_videos
from scrapers.amazon_scraper import AmazonScraper
import csv
import sqlite3
import functools
import json
import pandas as pd
import shutil

app = Flask(__name__)
app.secret_key = os.urandom(24)

# Database initialization
def init_db():
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (username TEXT PRIMARY KEY, password TEXT)''')
    conn.commit()
    conn.close()

init_db()

def clear_output_directory():
    """Clear all files in the output directory"""
    if os.path.exists('output'):
        for filename in os.listdir('output'):
            file_path = os.path.join('output', filename)
            try:
                if os.path.isfile(file_path):
                    os.unlink(file_path)
            except Exception as e:
                print(f"Error deleting {file_path}: {e}")

def check_file_has_data(file_path):
    if not os.path.exists(file_path):
        return False

    try:
        if file_path.endswith('.csv'):
            with open(file_path, 'r', encoding='utf-8') as f:
                csv_reader = csv.reader(f)
                # Skip header row
                next(csv_reader, None)
                # Check if there's at least one data row
                return bool(next(csv_reader, None))
        elif file_path.endswith('.xlsx'):
            df = pd.read_excel(file_path)
            return len(df) > 0
        elif file_path.endswith('.json'):
            with open(file_path, 'r', encoding='utf-8') as f:
                data = json.load(f)
                return bool(data)
    except Exception as e:
        print(f"Error checking file {file_path}: {str(e)}")
        return False
    return False

# Login required decorator
def login_required(f):
    @functools.wraps(f)
    def decorated_function(*args, **kwargs):
        if 'username' not in session:
            clear_output_directory()  # Clear data when session expires
            flash('Please login first', 'error')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/')
def index():
    if 'username' in session:
        return redirect(url_for('dashboard'))
    clear_output_directory()  # Clear data when accessing root without session
    return redirect(url_for('login'))

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        confirm_password = request.form['confirm_password']
        if password != confirm_password:
            flash('Passwords do not match', 'error')
            return redirect(url_for('signup'))

        conn = sqlite3.connect('users.db')
        c = conn.cursor()

        c.execute('SELECT username FROM users WHERE username = ?', (username,))
        if c.fetchone() is not None:
            conn.close()
            flash('Username already exists', 'error')
            return redirect(url_for('signup'))

        hashed_password = generate_password_hash(password)
        c.execute('INSERT INTO users (username, password) VALUES (?, ?)',
                 (username, hashed_password))
        conn.commit()
        conn.close()

        flash('Account created successfully', 'success')
        return redirect(url_for('login'))

    return render_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    # Clear any existing data when accessing login page
    clear_output_directory()
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = sqlite3.connect('users.db')
        c = conn.cursor()
        c.execute('SELECT password FROM users WHERE username = ?', (username,))
        user = c.fetchone()
        conn.close()

        if user and check_password_hash(user[0], password):
            session['username'] = username
            flash('Logged in successfully', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password', 'error')

    return render_template('login.html')

@app.route('/logout')
def logout():
    clear_output_directory()  # Clear data on logout
    session.pop('username', None)
    flash('Logged out successfully', 'success')
    return redirect(url_for('login'))

@app.route('/dashboard')
@login_required
```

```
def dashboard():
    # Ensure output directory exists
    os.makedirs('output', exist_ok=True)

    youtube_files = {
        'csv': check_file_has_data('output/youtube_videos.csv'),
        'excel': check_file_has_data('output/youtube_videos.xlsx'),
        'json': check_file_has_data('output/youtube_videos.json')
    }
    amazon_files = {
        'csv': check_file_has_data('output/amazon_products.csv'),
        'excel': check_file_has_data('output/amazon_products.xlsx'),
        'json': check_file_has_data('output/amazon_products.json')
    }
    return render_template('dashboard.html', youtube_files=youtube_files, amazon_files=amazon_files)

@app.route('/blog')
@login_required
def blog():
    return render_template('blog.html')

@app.route('/download/<scraper>/<format>')
@login_required
def download_file(scraper, format):
    if scraper not in ['youtube', 'amazon'] or format not in ['csv', 'excel', 'json']:
        flash('Invalid download request', 'error')
        return redirect(url_for('dashboard'))

    filename = f"{scraper}_{'videos' if scraper == 'youtube' else 'products'}.{format}"
    if format == 'excel':
        filename = filename.replace('excel', 'xlsx')

    file_path = os.path.join('output', filename)

    if not os.path.exists(file_path) or not check_file_has_data(file_path):
        flash('No data available for download', 'error')
        return redirect(url_for('dashboard'))

    return send_file(file_path, as_attachment=True)

@app.route('/clear/<scraper>', methods=['POST'])
@login_required
def clear_data(scraper):
    if scraper not in ['youtube', 'amazon']:
        flash('Invalid scraper specified', 'error')
        return redirect(url_for('dashboard'))

    try:
        # Clear files for the specified scraper
        for ext in ['csv', 'xlsx', 'json']:
            file_path = f'output/{scraper}_{"videos" if scraper == "youtube" else "products"}.{ext}'
            if os.path.exists(file_path):
                os.remove(file_path)

        flash(f'{scraper.title()} data cleared successfully', 'success')
    except Exception as e:
        flash(f'Error clearing {scraper} data: {str(e)}', 'error')
    return redirect(url_for('dashboard'))


@app.route('/scrape/youtube', methods=['POST'])
@login_required
def scrape_youtube():
    query = request.form.get('query', '')
    try:
        # Clear previous data
        for ext in ['csv', 'xlsx', 'json']:
            file_path = f'output/youtube_videos.{ext}'
            if os.path.exists(file_path):
                os.remove(file_path)

        videos = scrape_youtube_videos(query)

        if not videos:
            flash('No videos found', 'error')
            return redirect(url_for('dashboard'))

        # Ensure output directory exists
        os.makedirs('output', exist_ok=True)

        # Save as CSV
        with open('output/youtube_videos.csv', 'w', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=['title', 'views', 'upload_date'])
            writer.writeheader()
            writer.writerows(videos)

        # Save as Excel
        df = pd.DataFrame(videos)
        df.to_excel('output/youtube_videos.xlsx', index=False)

        # Save as JSON
        with open('output/youtube_videos.json', 'w', encoding='utf-8') as f:
            json.dump(videos, f, indent=2)

        flash('YouTube data scraped successfully', 'success')
    except Exception as e:
        flash(f'Error scraping YouTube data: {str(e)}', 'error')
    return redirect(url_for('dashboard'))

@app.route('/scrape/amazon', methods=['POST'])
@login_required
def scrape_amazon():
    query = request.form.get('query', '')
    max_pages = min(int(request.form.get('max_pages', 1)), 20)  # Limit to 20 pages

    try:
        # Clear previous data
        for ext in ['csv', 'xlsx', 'json']:
            file_path = f'output/amazon_products.{ext}'
            if os.path.exists(file_path):
                os.remove(file_path)

        scraper = AmazonScraper()
        products = scraper.scrape_amazon(query, max_pages)

        if not products:
            flash('No products found', 'error')
            return redirect(url_for('dashboard'))

        # Save as CSV
        df = pd.DataFrame([p.__dict__ for p in products])
        df.to_csv('output/amazon_products.csv', index=False)

        # Save as Excel
        df.to_excel('output/amazon_products.xlsx', index=False)

        # Save as JSON
        with open('output/amazon_products.json', 'w', encoding='utf-8') as f:
            json.dump([p.__dict__ for p in products], f, indent=2)

        flash(f'Amazon data scraped successfully. Found {len(products)} products.', 'success')
    except Exception as e:
        flash(f'Error scraping Amazon data: {str(e)}', 'error')
    return redirect(url_for('dashboard'))

if __name__ == '__main__':
    # Ensure output directory exists
    os.makedirs('output', exist_ok=True)
    app.run(debug=True)
```

8.3.4 Dashboard

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dashboard - Web Scraper</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <nav class="nav">
    <div class="nav-container">
      <a href="{{ url_for('dashboard') }}" class="nav-brand">Web Scraper</a>
      <div class="nav-links">
        <a href="{{ url_for('dashboard') }}" class="nav-link">Dashboard</a>
        <a href="{{ url_for('blog') }}" class="nav-link">Blog</a>
        <a href="{{ url_for('logout') }}" class="nav-link">Logout</a>
      </div>
    </div>
  </nav>

  <div class="container">
    <div class="dashboard">
      <div class="dashboard-header">
        <h1 class="dashboard-title">Welcome, {{ session['username'] }}</h1>
      </div>

      {% with messages = get_flashed_messages(with_categories=true) %}
```

```
            {% if messages %}
                {% for category, message in messages %}
                    <div class="flash-message flash-{{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}

        <div class="scraper-options">
            <div class="scraper-card">
                <h2 class="scraper-title">YouTube Channel Scraper</h2>
                <form action="{{ url_for('scrape_youtube') }}" method="post" class="scraper-form">
                    <div class="form-group">
                        <label for="youtube_query">YouTube Channel URL</label>
                        <input type="text" id="youtube_query" name="query" required
placeholder="https://www.youtube.com/@ChannelName/videos">
                        <small class="form-text">Enter the channel's videos page URL</small>
                    </div>
                    <div class="button-group">
                        <button type="submit" class="btn" data-loading-text="Scraping YouTube...">Scrape
YouTube</button>
                        {% if youtube_files.csv or youtube_files.excel or youtube_files.json %}
                            <form action="{{ url_for('clear_data', scraper='youtube') }}" method="post"
style="display: inline;">
                                <button type="submit" class="btn btn-clear">Clear Data</button>
                            </form>
                        {% endif %}
                    </div>
                    <div class="progress-message"></div>
                </form>
                {% if youtube_files.csv or youtube_files.excel or youtube_files.json %}
```

```
                    <div class="download-section">
                        <h3>Download YouTube Data</h3>
                        <div class="download-buttons">
                            {% if youtube_files.csv %}
                                <a href="{{ url_for('download_file', scraper='youtube', format='csv') }}" class="btn btn-download">Download CSV</a>
                            {% endif %}
                            {% if youtube_files.excel %}
                                <a href="{{ url_for('download_file', scraper='youtube', format='excel') }}" class="btn btn-download">Download Excel</a>
                            {% endif %}
                            {% if youtube_files.json %}
                                <a href="{{ url_for('download_file', scraper='youtube', format='json') }}" class="btn btn-download">Download JSON</a>
                            {% endif %}
                        </div>
                    </div>
                {% endif %}
            </div>


            <div class="scraper-card">
                <h2 class="scraper-title">Amazon Scraper</h2>
                <form action="{{ url_for('scrape_amazon') }}" method="post" class="scraper-form">
                    <div class="form-group">
                        <label for="amazon_query">Search Query</label>
                        <input type="text" id="amazon_query" name="query" required placeholder="https://www.amazon.in/s?k=laptop&crid=3BMA2X9SQNAZT&sprefix=laptop%2Caps%2C289&ref=nb_sb_noss_2">
                        <small class="form-text">Enter the Amazon search results URL</small>
                    </div>
                    <div class="form-group">
                        <label for="amazon_pages">Number of Pages to Scrape</label>
                        <input type="number" id="amazon_pages" name="max_pages" min="1" max="20"
    value="1">
                        <small class="form-text">Maximum 20 pages allowed</small>
                    </div>
                    <div class="button-group">
                        <button type="submit" class="btn" data-loading-text="Scraping Amazon...">Scrape
    Amazon</button>
                        {% if amazon_files.csv or amazon_files.excel or amazon_files.json %}
                            <form action="{{ url_for('clear_data', scraper='amazon') }}" method="post"
    style="display: inline;">
                                <button type="submit" class="btn btn-clear">Clear Data</button>
                            </form>
                        {% endif %}
                    </div>
                    <div class="progress-message"></div>
                </form>
                {% if amazon_files.csv or amazon_files.excel or amazon_files.json %}
                    <div class="download-section">
                        <h3>Download Amazon Data</h3>
                        <div class="download-buttons">
                            {% if amazon_files.csv %}
                                <a href="{{ url_for('download_file', scraper='amazon', format='csv') }}" class="btn
    btn-download">Download CSV</a>
                            {% endif %}
                            {% if amazon_files.excel %}
                                <a href="{{ url_for('download_file', scraper='amazon', format='excel') }}" class="btn
    btn-download">Download Excel</a>
                            {% endif %}
                            {% if amazon_files.json %}
                                <a href="{{ url_for('download_file', scraper='amazon', format='json') }}" class="btn
    btn-download">Download JSON</a>
                            {% endif %}
                        </div>
                    </div>
                {% endif %}
            </div>
        </div>
      </div>
    </div>

    <script>
      document.addEventListener('DOMContentLoaded', function() {
        const forms = document.querySelectorAll('.scraper-form');

        forms.forEach(form => {
          form.addEventListener('submit', function(e) {
                const button = this.querySelector('.btn');
                const loadingText = button.getAttribute('data-loading-text');
                const progressMessage = this.querySelector('.progress-message');

                button.setAttribute('data-original-text', button.textContent);
                button.disabled = true;
                button.textContent = loadingText;

                progressMessage.textContent = 'Scraping in progress... This may take a few minutes.';
                progressMessage.style.color = '#666';
                progressMessage.style.marginTop = '10px';
                progressMessage.style.fontSize = '14px';
```

```
                });
            });
        });
    </script>
</body>
</html>
```

9. TESTING

9.1 Testing Model Used

For this project, a combination of Black Box Testing, Functional Testing, and User Acceptance Testing (UAT) was used. These testing models ensured that both the backend functionality and user interface work as expected across the major modules:

- Amazon Scraper
- YouTube Scraper
- User Authentication
- Data Export
- Dashboard Navigation

1. Black Box Testing

- Focused on testing application functionality without knowing internal code logic.
- Verified inputs and outputs (scraper URLs, data exports, login inputs).

2. Functional Testing

- Tested individual functions and their expected outcomes.
- Ensured that modules like login, scraping, exporting, and file download operate correctly.

3. User Acceptance Testing (UAT)

- Ensured the system meets user expectations and is usable for non-technical users.
- Tested workflows from the user's point of view (end-to-end flow).

4. Regression Testing

- After changes or bug fixes, older features were re-tested to ensure nothing was broken.

9.2 Test Cases and Results

S.No	Test Scenario	Test Steps	Expected Result	Actual Result	Status (Pass/Fail)
1	User Signup	1. Open Signup Page 2. Enter username, password 3. Submit	User registered, redirected to login	Works as expected	Pass
2	User Login	1. Open Login Page 2. Enter valid credentials 3. Submit	Dashboard loads	Dashboard loads	Pass
3	Invalid Login	1. Open Login Page 2. Enter invalid credentials 3. Submit	Error "Invalid credentials " shown	Error shown	Pass
4	Amazon Scraper - URL Input	1. Login 2. Select Amazon Scraper 3. Enter valid Amazon search URL 4. Click Scrape	Scraper starts and extracts product data	Scraper works	Pass
5	Amazon Scraper - Pagination	1. Input URL with multiple pages 2. Set page limit =7 3. Click Scrape	Data from 3 pages extracted	Extracts correct data	Pass
6	YouTube Scraper - URL Input	1. Login 2. Select YouTube Scraper 3. Enter valid	Video data extracted	Works as expected	Pass

		YouTube Channel URL 4. Click Scrape			
7	Data Export - CSV	1. Scrape Amazon data 2. Select "Export as CSV" 3. Click Download	CSV file download ed	File downloads correctly	Pass
8	Data Export - JSON	1. Scrape YouTube data 2. Select "Export as JSON" 3. Click Download	JSON file download ed	File downloads correctly	Pass
9	Dashboard Navigation	1. Login 2. Navigate between scrapers 3. Log out	All pages open correctly	Navigatio n works	Pass
10	Session Management	1. Login 2. Close browser without logout 3. Reopen app	Session expired, forced login	Works as expected	Pass

Table 11: Test cases and status

10.USER MANUAL

10.1 User Manual Steps

Amazon & YouTube Data Scraper Web Application

1. Features

- Scrapes data from Amazon and YouTube
- Exports data in CSV, JSON, or Excel format
- Simple login/signup system
- Progress indicator during scraping
- Easy download of results after scraping

2. System Requirements

- Windows 10/11
- Google Chrome browser
- Python 3.10 or higher
- Minimum 4 GB RAM
- 500 MB free disk space

3. Installation Steps

- Extract project folder from project.zip
- Open cmd and navigate to project folder
- Run: python app.py
- Open browser and go to: http://127.0.0.1:5000

4. How to Use

- Sign Up / Login with username & password

Amazon Scraper

- Select Amazon Scraper
- Enter valid Amazon search URL
- Select output format (CSV/JSON/Excel)
- Click Scrape → Wait → Click Download File

YouTube Scraper

- Select YouTube Scraper

- Enter valid YouTube Channel URL
- Select output format (CSV/JSON/Excel)
- Click Scrape → Wait → Click Download File

5. Output Data

- Amazon → Title, Price, Rating, Reviews, Delivery, URL
- YouTube → Title, Views, Upload Date

6. Troubleshooting

- Wrong login? → Check credentials
- URL error? → Enter valid Amazon/YouTube URL
- Scraping hangs? → Check internet, restart app
- No file download? → Ensure output format selected before scraping

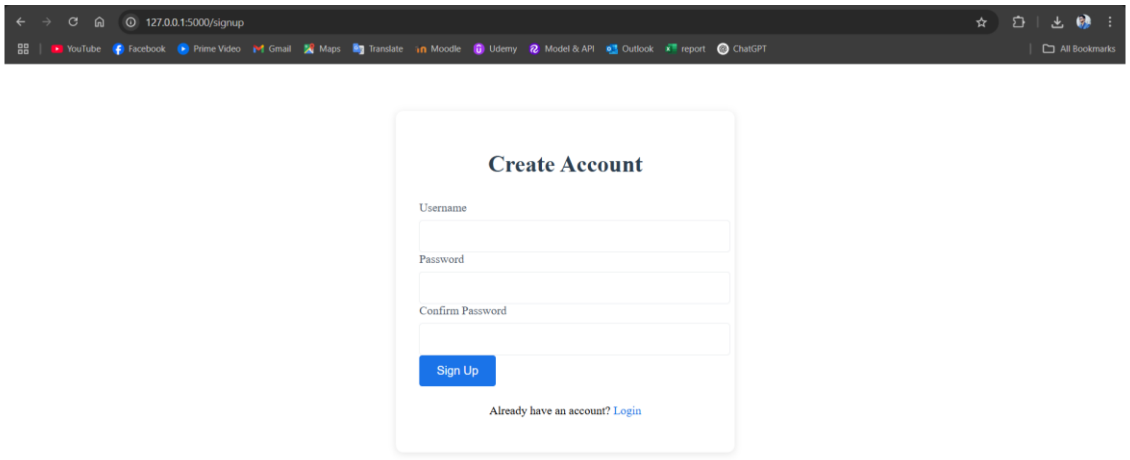
7. Logout & Exit

- Click Logout to sign out
- Close browser & stop Python app to exit

10.2 Screenshots with Explanation

10.2.1 Sign Up Page

This is the page where new users can register and create their new account to login to the web app



Create Account

Username

Password

Confirm Password

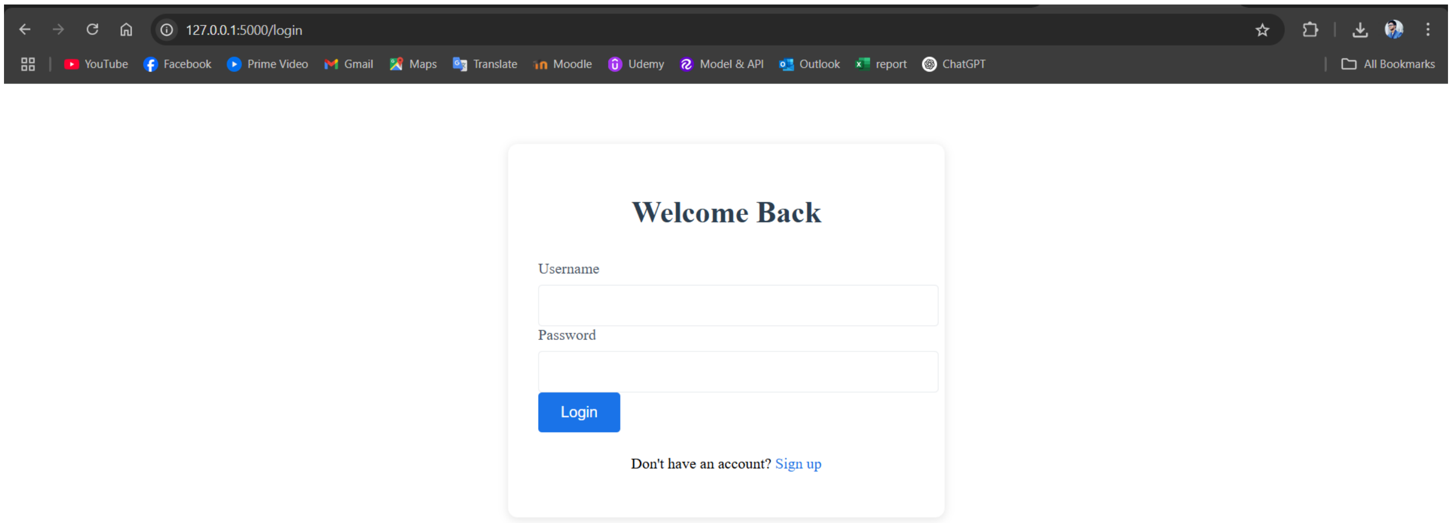
Sign Up

Already have an account? Login

Figure 14: Sign Up page

10.2.2 Login Page

This is the page where already registered users can login into their accounts without any hassle and access the web app features



Welcome Back

Username

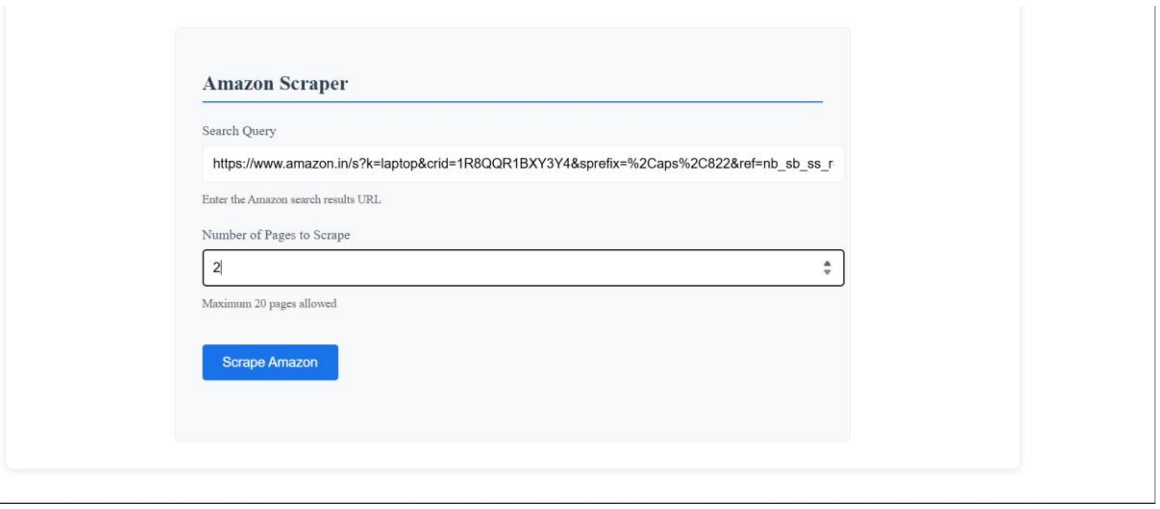
Password

Don't have an account? Sign up

Figure 15: Login Page

10.2.3 Scraper Page

This is the page where the scrapers are available for the already logged in user to use for the scraping needs


Amazon Scraper

Search Query

https://www.amazon.in/s?k=laptop&crid=1R8QQR1BXY3Y4&sprefix=%2Caps%2C822&ref=nb_sb_ss_r

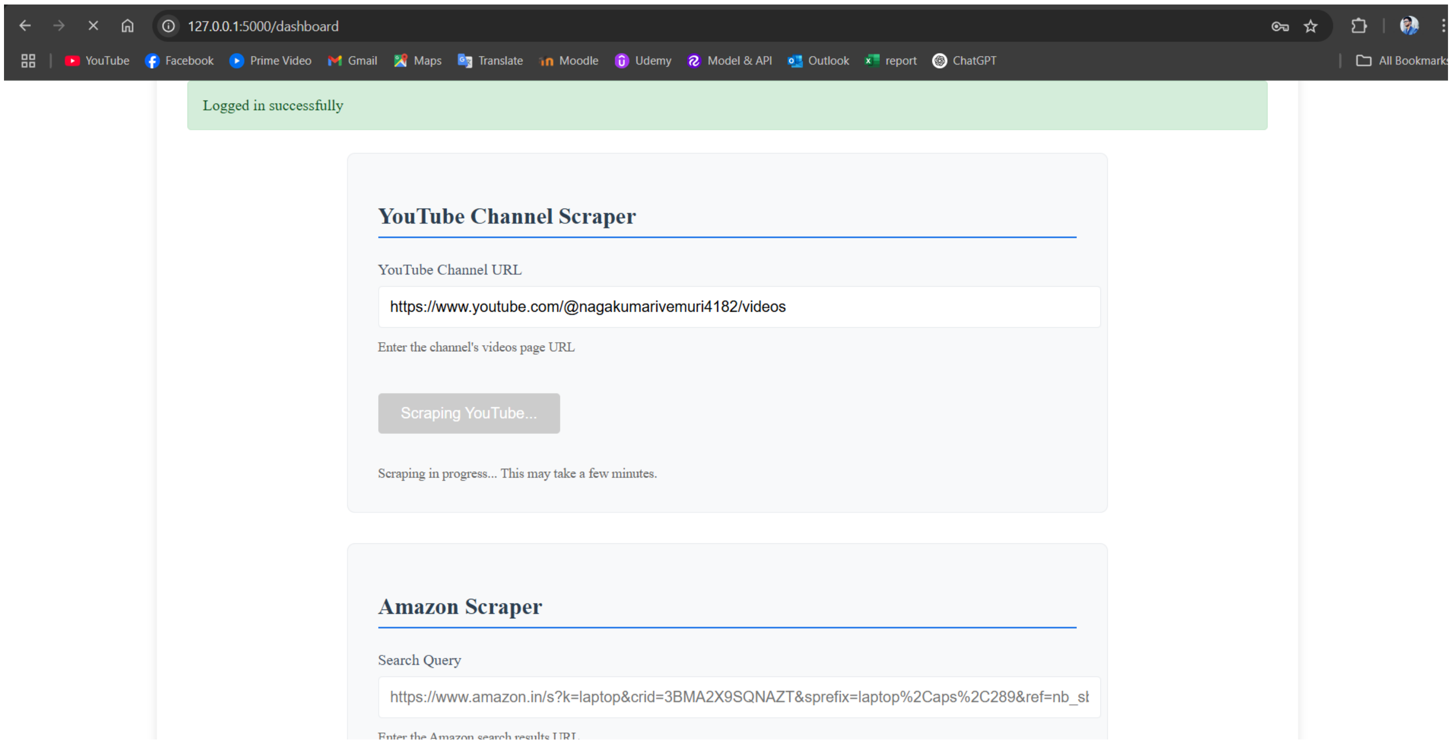
Enter the Amazon search results URL

Number of Pages to Scrape

2

Maximum 20 pages allowed

Figure 16: Amazon Scraper



127.0.0.1:5000/dashboard

Logged in successfully

YouTube Channel Scraper

YouTube Channel URL

https://www.youtube.com/@nagakumarivemuri4182/videos

Enter the channel's videos page URL

Scraping in progress... This may take a few minutes.

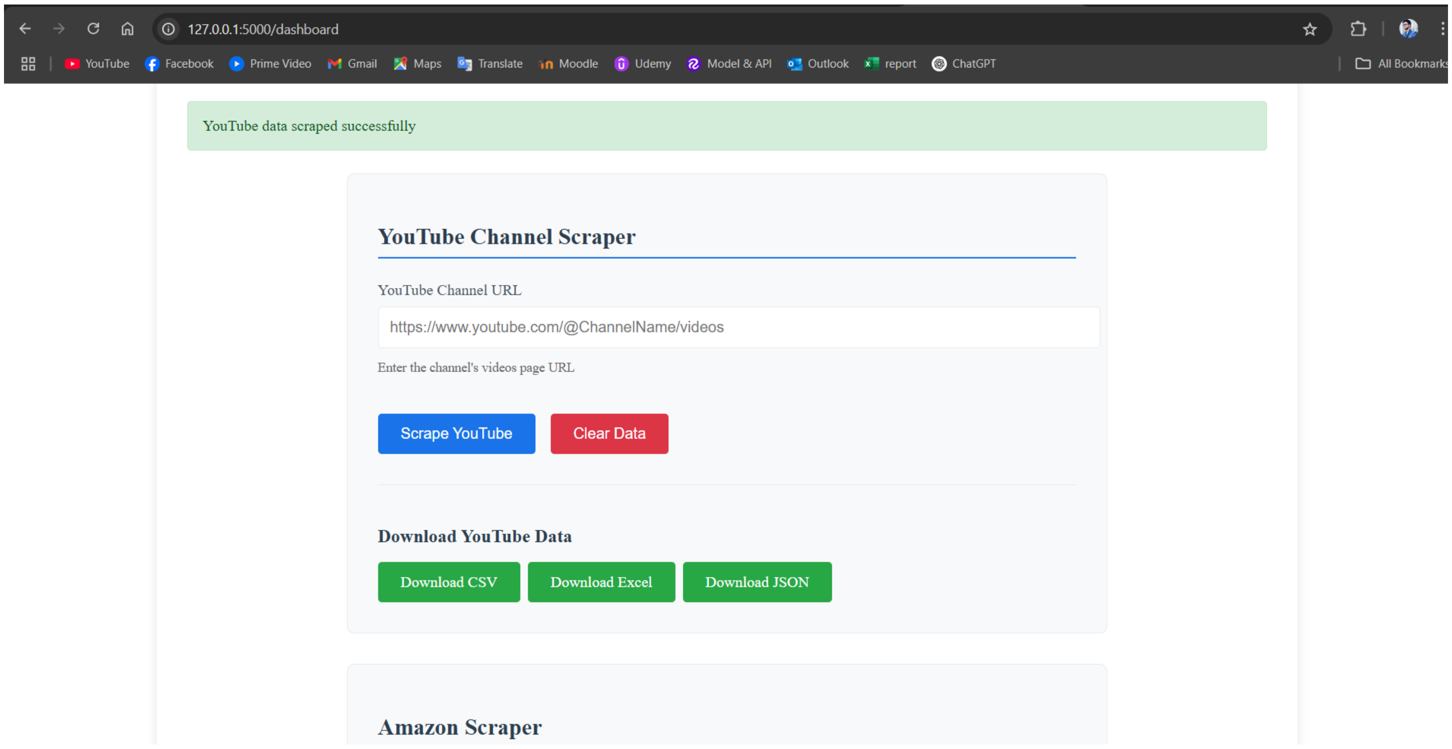
Amazon Scraper

Search Query

https://www.amazon.in/s?k=laptop&crid=3BMA2X9SQNAZT&sprefix=laptop%2Caps%2C289&ref=nb_sb

Enter the Amazon search results URL

Figure 17: YouTube Scraper



127.0.0.1:5000/dashboard

YouTube data scraped successfully

YouTube Channel Scraper

YouTube Channel URL

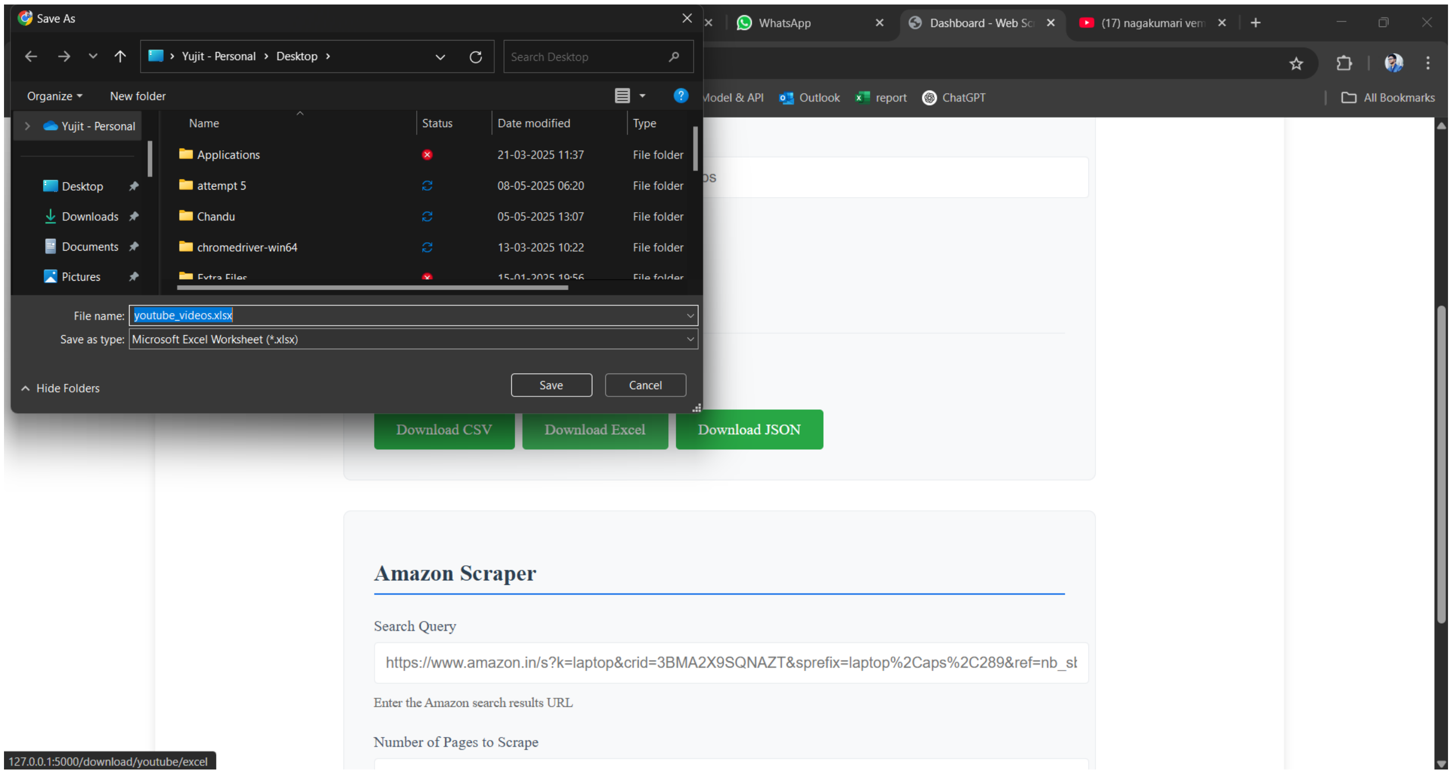
https://www.youtube.com/@ChannelName/videos

Enter the channel's videos page URL

Download YouTube Data

Amazon Scraper

Figure 18: After scraping download options



Name	Status	Date modified	Type
Applications		21-03-2025 11:37	File folder
attempt 5		08-05-2025 06:20	File folder
Chandu		05-05-2025 13:07	File folder
chromedriver-win64		13-03-2025 10:22	File folder

File name: youtube_videos.xlsx
Save as type: Microsoft Excel Worksheet (*.xlsx)

Amazon Scraper

Search Query

https://www.amazon.in/s?k=laptop&crid=3BMA2X9SQNAZT&sprefix=laptop%2Caps%2C289&ref=nb_sb

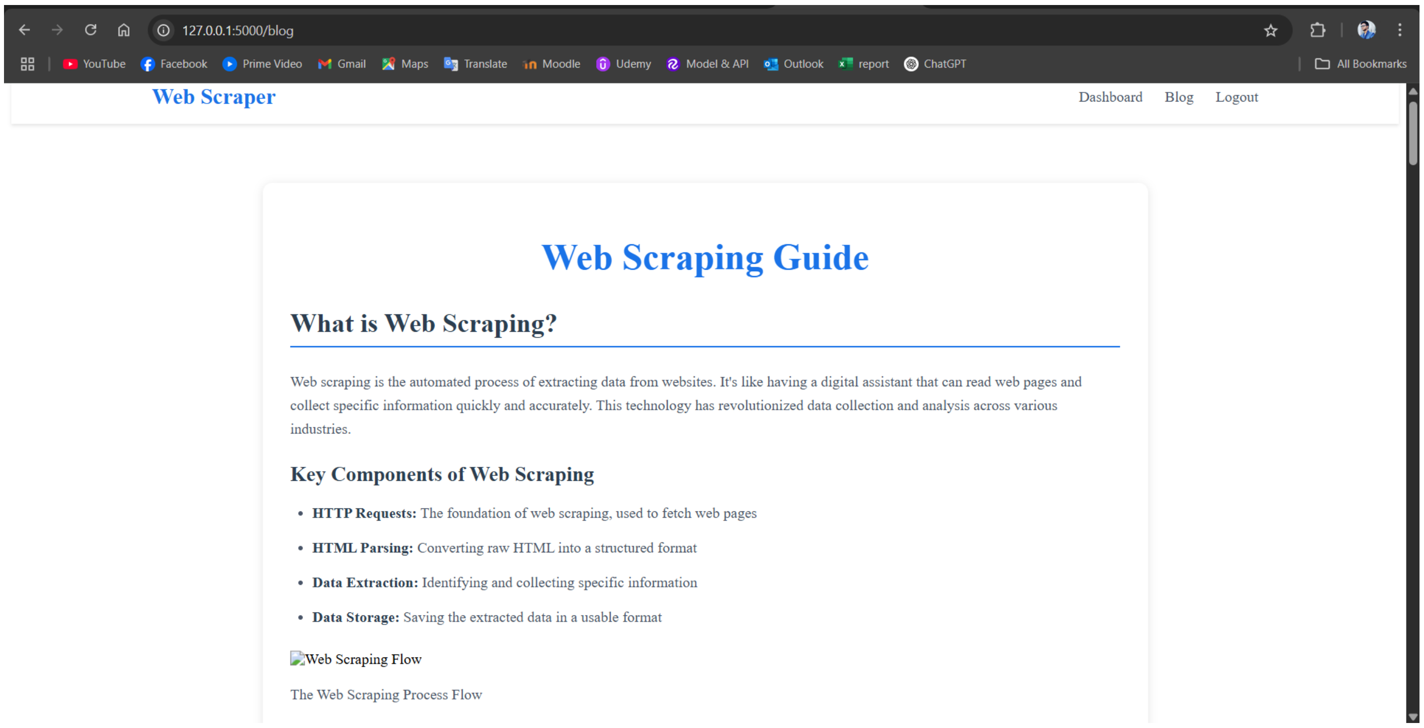
Enter the Amazon search results URL

Number of Pages to Scrape

Figure 19: Downloading of the file

10.2.4 Blog Page

In this page the user can view the information about web scraping and gain knowledge and learn how to do web scraping



Web Scraping Guide

What is Web Scraping?

Web scraping is the automated process of extracting data from websites. It's like having a digital assistant that can read web pages and collect specific information quickly and accurately. This technology has revolutionized data collection and analysis across various industries.

Key Components of Web Scraping

- **HTTP Requests:** The foundation of web scraping, used to fetch web pages
- **HTML Parsing:** Converting raw HTML into a structured format
- **Data Extraction:** Identifying and collecting specific information
- **Data Storage:** Saving the extracted data in a usable format

Web Scraping Flow

The Web Scraping Process Flow

Figure 20: Blog Page

11.CONCLUSION AND FUTURE WORKS

11.1 Conclusion

The development of the Amazon and YouTube Web Scraper Application has successfully demonstrated the design and implementation of a robust, user-friendly, and efficient data scraping system. This project integrates several modern technologies including Python, Flask, Selenium, BeautifulSoup, and Pandas into a unified web platform that enables users to easily extract valuable information from two of the most widely used platforms: Amazon and YouTube.

Through a clean and responsive web interface, authenticated users can effortlessly scrape multi-page product details from Amazon as well as comprehensive video information from YouTube channels. The system also provides multiple output formats (CSV, Excel, JSON), ensuring flexibility and ease of use for diverse data processing requirements. Furthermore, by incorporating essential security mechanisms such as password hashing, CSRF protection, and secure session management, the system ensures safe and reliable user interaction.

The inclusion of anti-detection features such as rate limiting, random delays, and user-agent rotation within the scrapers enhances the sustainability and ethical compliance of the scraping operations. Comprehensive error handling and logging mechanisms improve the robustness of the system, ensuring smooth performance even under unexpected conditions.

Overall, this project stands as a complete, production-ready solution that automates the time-consuming task of data collection from e-commerce and video-sharing platforms, while upholding standards of user experience, security, and data accuracy.

11.2 Future works

While the current system offers a wide range of functionalities, there is ample scope for future enhancements to increase its utility, scalability, and performance. The following points outline potential directions for future work:

1. Expand to Additional Platforms

- Integrate scrapers for other e-commerce platforms (e.g., Flipkart, eBay)
- Add support for additional video platforms (e.g., Vimeo, Dailymotion)

2. API Integration

- Incorporate Amazon Product Advertising API and YouTube Data API for more structured and reliable data access
- Reduce dependency on Selenium-based scraping

3. Enhanced Anti-Bot Measures

- Implement advanced anti-bot strategies such as Captcha solving integration, Proxy rotation, and Distributed scraping
- Improve scalability to minimize chances of IP blocking

4. Data Visualization

- Integrate basic data visualization tools (charts, graphs) within the dashboard to give users immediate insights from the scraped data

5. Advanced User Management

- Role-based access control (Admin/User roles)
- Two-factor authentication (2FA) for enhanced security

6. Asynchronous Scraping & Task Queues

- Integrate Celery or RQ (Redis Queue) with Flask to handle scraping tasks asynchronously
- Allow users to queue multiple scraping jobs and monitor progress in real time

7. Cloud Deployment & Scalability

- Deploy the application on cloud platforms (AWS, Azure, GCP) for broader accessibility and higher availability
- Enable distributed scraping across multiple instances

8. Improved File Management

- Enable automatic archival of scraped files
- Implement file version control and user-specific storage limits

9. Mobile Responsiveness

- Further optimize the UI for mobile and tablet devices to enable scraping from any device

10. Comprehensive Reporting System

- Automatic generation of summary reports post-scraping
- Provide users with actionable insights and analytics dashboards

11. Automated Updates & Maintenance

- Auto-update scraping modules to handle changes in Amazon/YouTube website structures
- Integrate monitoring tools to detect and repair scraper breakages automatically

12.ANNEXURE

12.1 Glossary of Terms and Abbreviations

Term / Abbreviation	**Description**
API	Application Programming Interface. A set of rules that allows software programs to communicate with each other.
CSV	Comma Separated Values. A file format used to store tabular data in plain text.
JSON	JavaScript Object Notation. A lightweight data-interchange format that is easy to read and write for humans and machines.
UI	User Interface. The visual part of a software application that users interact with.
UX	User Experience. The overall experience of a person using a product, especially in terms of ease of use.
HTML	HyperText Markup Language. The standard language for creating web pages.
CSS	Cascading Style Sheets. A language used to describe the style of an HTML document.
Flask	A micro web framework written in Python for developing web applications.
Selenium	An open-source tool used for automating web browser interactions.
Pandas	A Python library providing data structures and data analysis tools.
Data Scraping	The process of extracting data from websites.
Proxy Rotation	A technique used to avoid detection while scraping by changing the IP address periodically.
Captcha	A challenge–response test used to determine whether a user is human.
XPath	XML Path Language. A query language for selecting nodes from an XML document (used in Selenium).
SQL	Structured Query Language. A language used to communicate with databases.
CRUD	Create, Read, Update, Delete. The four basic functions of persistent storage.
DFD	Data Flow Diagram. A graphical representation of the flow of data in a system.
2-Tier Architecture	A client-server architecture where the client communicates directly with the database server.

3-Tier Architecture	A software architecture with three layers: presentation, application (logic), and database.
CSV, Excel, JSON Export	Functionality that allows data to be saved in different file formats.

Table 12: Glossary

12.2 References

- Flask Documentation: https://flask.palletsprojects.com
- Selenium Official Documentation: https://www.selenium.dev/documentation
- Pandas Documentation: https://pandas.pydata.org/docs
- Amazon Website: https://www.amazon.in
- YouTube Official Website: https://www.youtube.com
- W3Schools HTML Reference: https://www.w3schools.com/html
- Python Official Documentation: https://docs.python.org/3/
- PlantUML Documentation: https://plantuml.com/

About College (UVPCE)

U.V. Patel College of Engineering

Ganpat University

Ganpat University-U. V. Patel College of Engineering (GUNI-UVPCE) is situated in Ganpat Vidyanagar campus. It was established in September 1997 with the aim of providing educational opportunities to students from It is one of the constituent colleges of Ganpat University various strata of society. It was armed with the vision of educating and training young talented students of Gujarat in the field of Engineering and Technology so that they could meet the demands of Industries in Gujarat and across the globe. The College is named after Shri Ugarchandbhai Varanasi Bhai Patel, a leading industrialist of Gujarat, for his generous support. It is a self-financed institute approved by All India Council for Technical Education (AICTE), New Delhi and the Commissionerate of Technical Education, Government of Gujarat. The College is spread over 25 acres of land and is a part of Ganpat Vidyanagar Campus. It has six ultra-modern buildings of architectural splendour, class rooms, tutorial rooms, seminar halls, offices, drawing hall, workshop, library, well equipped departmental laboratories, and several computer laboratories with internet connectivity through 1 Gbps Fiber link, satellite link education center with two-way audio and one-way video link. The superior infrastructure of the Institute is conducive for learning, research, and training. The Institute offers various undergraduate programs, postgraduate programs, and Ph.D. programs. Our dedicated efforts are directed towards leading our student community to the acme of technical excellence so that they can meet the requirements of the industry, the nation and the world at large. We aim to create a generation of students that possess technical expertise and are adept at utilizing the technical 'know-hows' in the service of mankind. We strive towards these Aims and Objectives:

-To offer guidance, motivation, and inspiration to the students for well-rounded development of their personality.

- To impart technical and need-based education by conducting elaborated training programs.

-To shape and Mold the personality of the future generation.

-To construct fertile ground for adapting to dire challenges.