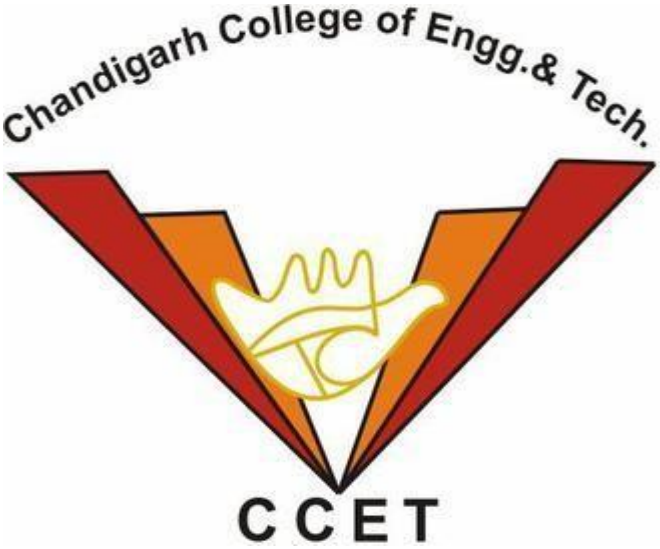

(Diploma Wing)

Minor Project Report

On

'Web Scrappy'

Submitted To:

Dr. Santosh Kumar Yadav
Lecturer CSE Department

Submitted By:

Amey	Rajat Garg
9503/22	9527/22
Comp Sci Engg	Comp Sci Engg

WEB SCRAPPY: A WEB SCRAPING PROJECT

1. Introduction

In today's data-driven world, collecting and analysing web data is essential for market research, competitive analysis, and academic research. However, manual data collection is time-consuming and inefficient. This project provides an automated solution for extracting structured data from web pages, ensuring efficiency, accuracy, and scalability.

The Web Scrappy our web scraping project will provide users with a streamlined method to collect and process data from multiple sources efficiently. By leveraging technologies like Scrapy and Selenium, the system will be capable of handling dynamic and static web pages while maintaining robust filtering mechanisms to ensure data relevance.

2. Objectives

- **Automated Data Extraction**: Saves time and effort in collecting website data.
- **Filtering Mechanism**: Ensures only relevant links are processed.
- **Scalability**: Works with multiple websites and large datasets.
- **Customizable & Extensible**: Easily adaptable to different scraping requirements.
- **Secure Data Handling:** Implement security measures to protect API keys and sensitive information.
- **Logging and Debugging:** Maintain structured logging for better error handling and monitoring.

3. Features and Functionalities

- **URL Extraction:**
  - Uses Selenium to extract hyperlinks from web pages.
  - Stores extracted links in JSON format for further processing.
- **Content Scraping:**
  - Utilizes Scrapy to crawl filtered web pages and extract structured content.
  - Saves scraped data in JSON and Excel formats.
- **Filtering System:**
  - Ensures only relevant links are processed based on predefined rules.
  - Reduces redundant or unwanted data collection.

- **API Integration:**
  - Enables API-based data retrieval for additional metadata.
  - Securely manages API keys and credentials.
- **Logging & Error Handling:**
  - Logs execution details for debugging.
  - Implements exception handling to prevent crashes.
- **Modular Structure:**
  - Well-structured project files for ease of maintenance and expansion.

4. System Architecture

The project follows a modular architecture for maintainability and efficiency. Key components include:

- **Frontend:**
  - CLI-based interface for executing scraping tasks.
- **Backend:**
  - Python scripts utilizing Scrapy and Selenium.
  - Data storage using JSON and Excel files.
- **Security & Authentication:**
  - API keys managed securely in environment variables.
- **Tools & Libraries:**

- **Scrapy** - For structured web scraping.
- **Selenium** - For dynamic website interaction.
- **pandas** - For handling and storing data in structured formats.
- **openpyxl** - For reading/writing Excel files.
- **logging** - For structured logging.
- **requests** - For API calls and fetching data.

5. Methodology

- **Requirement Analysis:** Understanding the data extraction needs.
- **Design:** Structuring modules, defining workflows, and selecting technologies.
- **Development:** Implementing scraping logic, filtering mechanisms, and storage formats.
- **Testing:** Conducting unit tests and integration tests.
- **Deployment:** Deploying the scraper for real-world data extraction.
- **Maintenance & Enhancements:** Continuous improvements based on user feedback.

6. Expected Outcomes

- **Efficient Data Collection:** Automates web scraping to save time and effort.
- **Data Accuracy & Relevance:** Filters out irrelevant data to improve quality.
- **Scalability:** Supports multiple websites and large datasets.
- **Enhanced Security:** Protects sensitive credentials and ensures reliable data handling.
- **Continuous Innovation:** Future improvements including multi-threading, database integration, and AI-based content filtering.

7. Error Handling & Logging

- Errors are logged in website_processing.log.
- The project uses **try-except blocks** to handle exceptions gracefully.
- Logs include timestamps, error messages, and debugging details.

8. Future Enhancements

- **Multi-threading for Faster Processing**: Implementing parallel requests.
- **Database Integration**: Storing scraped data in a database.
- **Advanced Filtering**: AI-based content filtering.
- **Headless Browsing**: Optimizing Selenium execution.

9. Conclusion

This web scraping project automates the process of extracting, filtering, and analysing website content. By leveraging Scrapy and Selenium, it efficiently handles large datasets, making it ideal for research, monitoring, and data collection tasks.