Project Report: Easy Hunt — LinkedIn Job Scraper Bot

Author: Rohit Bhavsar
Date: July 22, 2025

Abstract

This report details the design, development, and deployment of the "Easy Hunt - LinkedIn Job Scraper Bot," a web application created to automate and enhance the process of searching for job opportunities on LinkedIn. The project addresses the inefficiencies of manual job searching by providing a web-based interface where users can programmatically scrape, filter, and analyze job listings. The system is built using Python, with Streamlit for the user interface, Selenium for web automation, and Pandas for data handling. A key architectural feature is its containerization with Docker, which ensures reliable deployment on cloud platforms like Render, overcoming significant environment-specific challenges encountered during development.

---

1. Introduction

1.1. Problem Statement

In the current competitive job market, professionals often spend a significant amount of time manually searching through job portals like LinkedIn. This process is highly repetitive and lacks efficient methods for aggregating and analyzing data across multiple searches. Standard user interfaces do not allow for sophisticated filtering, historical tracking, or bulk data export, making it difficult for job seekers to gain strategic insights from job market trends.

1.2. Project Objectives

The primary objectives of this project were to:

1. Develop a tool to automate the scraping of job listings from LinkedIn based on user-defined criteria (role, location).
2. Create an intuitive and interactive web interface for users to control the scraper and view the results.
3. Implement features to clean, filter, and sort the scraped data to enhance its utility.

4. Provide a simple method for users to export the data for offline analysis.
5. Ensure the application is robust and can be reliably deployed to a public cloud platform.

1.3. Scope

The scope of the initial version (Tier 1) includes scraping summary data from LinkedIn's search results page, filtering by experience level, removing duplicates, sorting by post-date, and providing a CSV download. The project is designed to be extensible, with a clear roadmap for future enhancements such as database integration and automated scraping.

---

2. System Architecture and Design

2.1. Core Components

- **Streamlit (Frontend/UI):** Chosen for its ability to rapidly create interactive data-centric web applications directly from Python scripts, eliminating the need for separate frontend development.
- **Selenium (Web Scraper):** The core automation engine. It launches and controls a headless web browser (Chromium), simulating user actions to navigate LinkedIn and retrieve page content.
- **BeautifulSoup4 (HTML Parser):** Works in tandem with Selenium to parse the raw HTML content of the job listings page, making it easy to extract specific data points like job titles, companies, and links.
- **Pandas (Data Manipulation):** Used to structure the scraped data into a clean DataFrame, handle data cleaning operations (like dropping duplicates), and facilitate the CSV export.
- **Docker (Containerization):** The key to solving deployment issues. It encapsulates the application, the Python environment, and all system-level dependencies (including the Chromium browser) into a single, portable container image.
- **Render (Deployment Platform):** A cloud platform chosen for its excellent native support for Docker, providing a straightforward path to deploying the containerized application.

2.2. Data Flow

The application follows a simple, linear data flow:

1. **User Input:** The user enters a job role and locations into the Streamlit web interface and submits the form.
2. **Scraping Execution:** Streamlit triggers the main Python function, which initializes the Selenium WebDriver.
3. **Data Retrieval:** Selenium navigates to LinkedIn, performs the search, scrolls to load all results, and fetches the page's HTML source.
4. **Parsing & Cleaning:** BeautifulSoup parses the HTML, and the extracted data is loaded into a Pandas DataFrame for cleaning and sorting.
5. **Display & Export:** The final, clean DataFrame is sent back to the Streamlit UI for display in a table and visualization in a chart. The user is presented with a button to download the data as a CSV file.

---

3. Features Implemented (Tier 1)

- **Targeted Scraping:** Users can specify a job title and multiple comma-separated locations.
- **Advanced Filtering:** An experience-level filter allows users to pre-filter the jobs on LinkedIn before scraping begins.
- **Data Deduplication:** A crucial data cleaning step that removes redundant listings, providing a clean final dataset.
- **Temporal Sorting:** Results are programmatically sorted to show the most recently posted jobs first.
- **Data Visualization:** A bar chart displays the top 20 companies by the number of job postings, offering a quick market overview.
- **CSV Export:** A one-click download button for the final dataset.

---

4. Deployment

4.1. Initial Deployment Challenges

The initial attempts to deploy the application on Streamlit Community Cloud failed. The root cause was a series of environment-specific dependency issues:

- **System Package Unavailability:** The cloud environment's package manager (apt-get) could not locate the required browser packages (google-chrome-stable, chromium-browser).
- **Driver Mismatch:** When a browser package was successfully installed, its version was often incompatible with the version of ChromeDriver being installed by Python libraries, leading to SessionNotCreatedException errors.

4.2. Solution: Containerization with Docker

To overcome these platform limitations, the project was containerized using Docker. A Dockerfile was created to define a custom, reproducible environment. This approach provided complete control over the system, allowing for:

- The installation of a specific Linux distribution (python:3.9-slim).
- The explicit installation of the chromium system package.
- The use of Selenium's built-in manager to automatically fetch the correct, compatible driver for the installed browser version.

4.3. Deployment on Render

With the Dockerfile in place, the application was successfully deployed on Render. Render's native support for Docker made the process seamless. By pointing Render to the project's GitHub repository, it automatically detected the Dockerfile, built the container image, and deployed it as a web service without any of the previous dependency conflicts.

4.4 Live Demo

Live Demo: https://linkedin-job-scraper-u88y.onrender.com/

---

5. Future Roadmap

The project is designed for future expansion. The planned roadmap includes:

- **Tier 2:** Integrating a PostgreSQL database for persistent data storage, scraping full job descriptions for deeper analysis, and implementing logic to extract key skills from the text.

- **Tier 3:** Automating the entire workflow with scheduled cron jobs, adding email alerts for new job listings, and potentially leveraging AI/LLMs for advanced text summarization and resume matching.

---

6. Conclusion

The LinkedIn Job Scraper project successfully met all its initial objectives. It provides a functional and user-friendly tool for automating job searches and demonstrates a robust solution to a common real-world data acquisition problem. More importantly, the project serves as a practical case study in modern application deployment, highlighting the critical role of containerization with Docker in overcoming platform-specific dependency issues and ensuring reliable, portable deployment.