Cold Email Generator for Technik Nest

Submitted By:

- **Hassan Ali**
- **Ahsan Naseer**

Course:

Software Design and Architecture

Instructor:

Dr. Muhammad Zeeshan & Mam Zainab Tahir

Abstract

This project presents the **Cold Email Generator**, designed to automate the creation of personalized cold emails for job opportunities. By scraping job descriptions from online job listings, refining the extracted data using an advanced Large Language Model (LLM), and storing this data in a vector database (ChromaDB), the system generates emails tailored to specific job postings. The project leverages modern technologies such as **Selenium** for web scraping, **Langchain** for LLM integration, and **Docker** for efficient containerization and deployment.

Introduction

In the fast-paced digital job market, the process of manually drafting personalized cold emails for job applications can be time-consuming and inefficient. This project addresses this problem by automating the generation of professional cold emails. By scraping job descriptions and refining them using an LLM, the system creates tailored emails that streamline the job application process. This tool is developed for **Technik Nest**, a software house, to improve job application efficiency.

Problem Statement

Creating personalized cold emails for job opportunities is a repetitive task that consumes significant time and effort. Job seekers need to manually tailor each email to reflect the specifics of the job listing, a process that is prone to errors and inefficiencies. Automating this process would enhance productivity and improve the overall experience for job seekers and recruiters.

Objectives

- Automate the generation of cold emails based on online job descriptions.
- Scrape job descriptions from job listings using **Selenium**.
- Refine the job descriptions using an advanced **Large Language Model (LLM)**.
- Generate personalized cold emails that include tailored job details and relevant project links.
- Containerize the entire system using **Docker** for easy deployment and portability.

User Requirements

Functional Requirements:

1. **Job Scraping:**
    - The system should scrape job descriptions from online job listings using **Selenium**.
    - The system must allow users to input URLs of job listings, from which descriptions will be extracted.
2. **Data Refining:**
    - The system must refine the scraped job descriptions using the **Llama3.1 70-b-versatile LLM** via **Langchain** to ensure professional, concise, and personalized content.
3. **Email Generation:**
    - The system must automatically generate cold emails based on the refined job descriptions.
    - Emails must include specific job details, such as role, responsibilities, qualifications, and project links tailored to the job description.
4. **Frontend Interface:**
    - The system should provide an easy-to-use interface via **Streamlit** where users can:
        - Input job URLs.
        - View generated cold emails.
        - Download or copy emails for sending.
5. **Data Storage:**
    - The system must store the refined job data, along with relevant project links, in a **ChromaDB** vector database for future use.
6. **Deployment:**

- The system should be packaged in a **Docker** container for portability and ease of deployment across different environments.

Non-Functional Requirements:

1. **Performance:**
    - The system should perform scraping, refining, and email generation tasks within an acceptable time frame, ideally under 10 seconds per job listing.
    - The containerized application should run efficiently even with a moderate load of job listings (up to 50 listings per session).
2. **Scalability:**
    - The system should be scalable, allowing for the integration of additional job scraping sources in the future without major system overhaul.
    - The email generation system should be able to handle multiple simultaneous users with minimal degradation in performance.
3. **Security:**
    - The system must ensure secure handling of job URLs and data. Sensitive data, such as personal email addresses or company information, should not be stored or exposed inappropriately.
4. **Reliability:**
    - The system must function reliably, with minimal downtime. The web scraping and email generation processes should be robust to variations in job listing formats and ensure that emails are always generated correctly.
    - Backup and recovery mechanisms should be in place for the ChromaDB storage.
5. **Usability:**
    - The **Streamlit** frontend should be intuitive and user-friendly, allowing job seekers to input job URLs, view generated emails, and make necessary edits with ease.
    - The system should be accessible to users without any prior technical knowledge.
6. **Compatibility:**
    - The system should be compatible with major operating systems (Linux, Windows, macOS) for both the development and deployment environments.
    - Docker containers should run seamlessly across these platforms.
7. **Maintainability:**
    - The system should be easy to maintain and update, with clear documentation on how to add new features, scrape additional job sites, and fine-tune the LLM for better email personalization.
8. **Availability:**

- The system should be accessible at any time, with a high uptime (above 99%) to ensure job seekers can generate cold emails whenever needed.

Technologies Used

- **Python**: The primary language used for backend development.
- **Selenium**: For web scraping job descriptions from online job listings.
- **Langchain**: Framework for LLM integration, enabling job description refinement.
- **ChromaDB**: Vector database used to store job data and project links.
- **Llama3.1 70-b-versatile**: Open-source LLM model from Meta used for refining job descriptions.
- **Streamlit**: Python library used to build the user interface of the email generator.
- **Docker**: Containerization platform used for deploying and scaling the project.

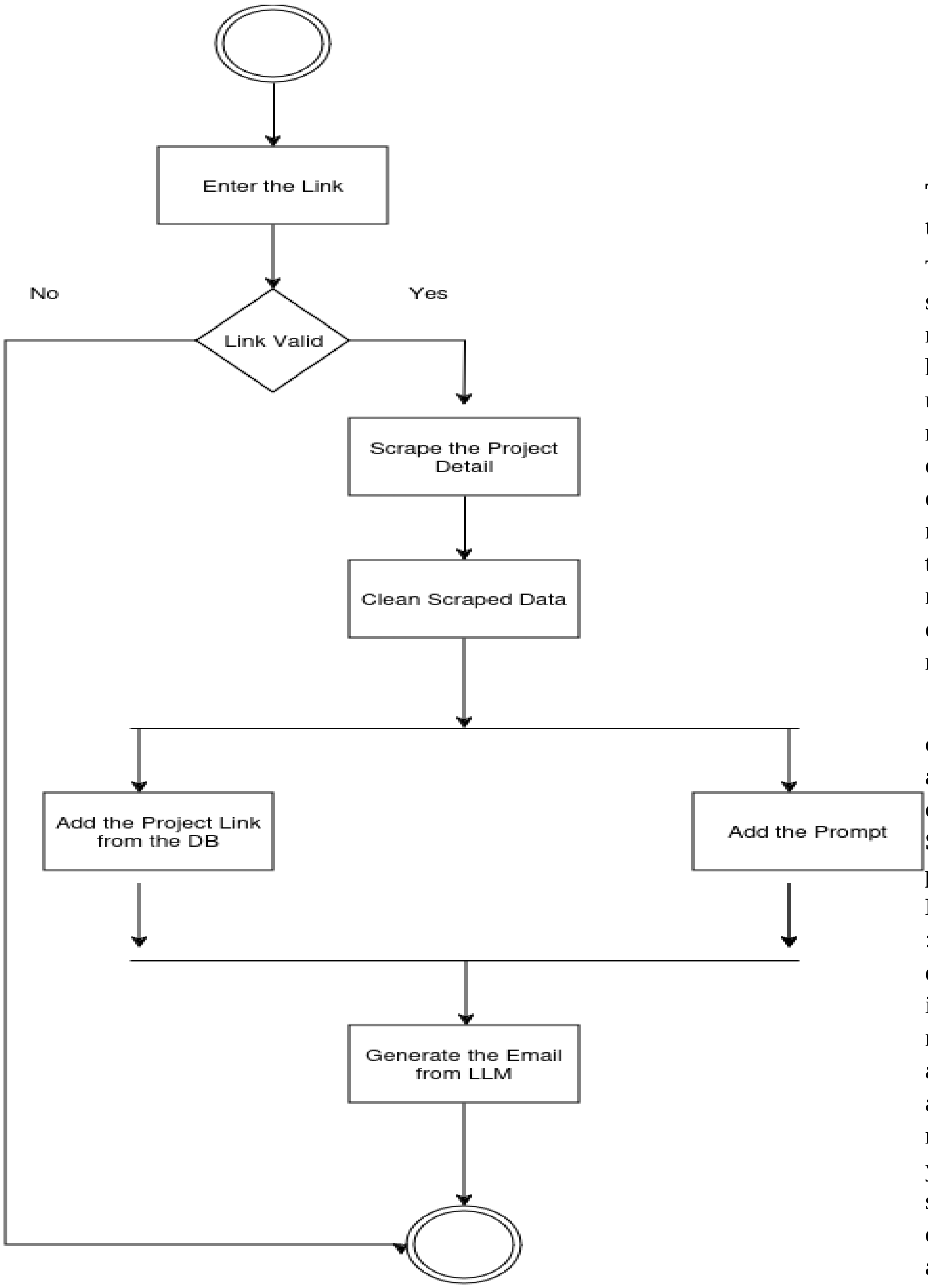
System Design and Architecture

The project follows a modular architecture, consisting of several key components that interact seamlessly:

1. **Web Scraping:** Selenium scrapes job descriptions from URLs provided by the user.
2. **Data Refining:** Langchain connects to the Llama3.1 LLM to process and refine the raw text.
3. **Data Storage:** Refined job data and project links are stored in a ChromaDB vector database.
4. **Email Generation:** The refined job data is used to automatically generate personalized cold emails.
5. **Frontend Interface:** Streamlit provides an intuitive interface for users to interact with the system.
6. **Containerization:** Docker ensures the project is portable and easy to deploy.

- **ChromaDB**: Vector database used to store job data and project links.
- **Llama3.1 70-b-versatile**: Open-source LLM model from Meta used for refining job descriptions.
- **Streamlit**: Python library used to build the user interface of the email generator.
- **Docker**: Containerization platform used for deploying and scaling the project.

Activity Diagram


Enter the Link
No
Yes
Link Valid
Scrape the Project Detail
Clean Scraped Data
Add the Project Link from the DB
Add the Prompt
Generate the Email from LLM

Testing

The system has undergone extensive testing to ensure:

- **Accuracy of Scraped Data**: Job descriptions are accurately scraped and processed.

- **Correct Email Generation**: Emails are correctly generated with job details and project links.
- **User Interface**: The Streamlit interface is tested for responsiveness and user-friendliness.
- **System Performance**: Docker container performance is validated across different environments and load conditions.

Challenges and Solutions

Challenges:

1. **Anti-Scraping Measures**: Some job listing websites have anti-scraping mechanisms in place, which required bypassing.
2. **LLM Performance**: Fine-tuning the LLM to produce concise, professional email content was an iterative process.

Solutions:

1. **Web Scraping**: Rotating user agents and using headless browsing in Selenium helped bypass anti-scraping measures.
2. **LLM Refining**: Regular prompt engineering and fine-tuning helped improve the quality of generated emails.

Results

The **Cold Email Generator** automates the process of generating tailored cold emails based on job descriptions, significantly improving efficiency for job seekers. The system successfully scrapes, refines, and stores job data, and generates personalized emails with the relevant job details and project links.

Conclusion

The **Cold Email Generator** project addresses the issue of manually crafting personalized cold emails for job applications. By automating this process through web scraping, LLM integration, and email generation, the system not only saves time but also enhances productivity. Future improvements could include expanding the scraping capabilities to support more job platforms and enhancing the LLM's ability to further tailor email content.

References

- Langchain Documentation
- Selenium Documentation

- ChromaDB Documentation
- Llama3.1 70-b-versatile Documentation
