##### SELF CERTIFICATE

This is to certify that the dissertation/project report entitled “Data Scraping and Visualization with Python” is done by me is an authentic work carried out for the partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Applications under the guidance of Dr Santosh Kumar Singh. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

Parth Verma

Roll No.: 01428402021

##### SELF CERTIFICATE

This is to certify that the dissertation/project report entitled “Data Scraping and Visualization with Python” is done by me is an authentic work carried out for the partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Applications under the guidance of Dr Santosh Kumar Singh. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

Anmol Sahni

Roll No.: 01528402021

##### Acknowledgement

The note starts with thanks to Almighty who actually created this piece of work and helped us when things were not easy for us.

I am very grateful and indebted to our Guide Dr. Santosh Kumar Singh who immensely helped and rendered her/his valuable advice, precious time, knowledge, and relevant information regarding the collection of material. He has been a major source of inspiration throughout the project as he not only guided me throughout this Project Report Data Scraping and Visualization with Python but also encouraged me to solve problems that arose during this report.

His guidance and suggestions about this Project report have really enlightened me. It has been a great help to support to have him around.

And finally, I would like to mention appreciation to our parents and friends who have been instrumental throughout this period by providing unrelenting encouragement.

Name of Students Enrolment No. Signature

Parth Verma 01428402021

Anmol Sahni 01528402021

##### Table of Contents

|  |  |
| --- | --- |
| 1. Synopsis of the Project |  |
| 2. Main Report |  |
| i. Objective & Scope of the Project |  |
| ii. Theoretical Background Definition of Problem |  |
| iii. System Analysis & Design vis-a-vis User Requirements |  |
| iv. System Planning (PERT Chart) |  |
| v. Methodology adopted, System Implementation & Details of Hardware & Software used System Maintenance & Evaluation |  |
| vi. Detailed Life Cycle of the Project |  |
| a. ERD, DFD |  |
| b. Input and Output Screen Design |  |
| c. Process involved |  |
| d. Methodology used testing |  |
| e. Test Report, Printout of the Report & Code Sheet |  |
| VI. Coding and Screenshots of the project |  |
| VII. Conclusion and Future Scope |  |
| VIII. References |  |

##### Synopsis

# Table of Contents

[1. Problem Statement 3](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262940)

[2. Objectives and Scope 4](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262941)

[3. Methodology 5](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262942)

[4. Hardware and Software Used 6](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262943)

[5. Resources and Limitations 7](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262944)

[6. Testing Methodology 8](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262945)

[7. Conclusions 9](file:///D:\Parth\Papers\Papers%20S5\Minor%20Project\documentation\new%20Synopsis2.docx#_Toc147262946)

##### Problem Statement

# Problem

Lack of Analysis and visualisation when dealing with online social networking sites for working professionals seeking job opportunities. Data scraping and visualization project can solve various problems across different domains such as Market Research and Competitor Analysis.

##### Objectives and Scope

The primary objective of a data scraping and visualization project is to extract, process, and analyse data from concerned websites or other data repositories, and then present the insights gained from that data in a visual and meaningful way.

To Develop a python program to take user inputs from user and perform data scraping on the basis of data inputted by the user.

Scraping data with optimum speed and visualise the data scraped in the form of insights and visualisation which convey meaningful data.

##### Methodology

**Data Collection:** Gather relevant data from one or more sources to support analysis and decision-making.

**Data Processing:** Clean, transform, and structure the raw data for analysis.

**Data Analysis:** Extract meaningful insights, trends, patterns, and correlations from the data.

**Data Visualization:** Present the analysed data in a visual format that is easy to understand and interpret

**Decision Support:** Enable users to make informed decisions based on the visualized data and insights.

**Automation:** Streamline the process of data collection, analysis, and visualization through automation, where applicable.

##### Hardware and Software Used

# Hardware Requirements:

* RAM: 4 Gb minimum
* Storage: 1 Gb minimum
* Internet speed over 30mbps

# Software Requirements:

* Operating system platform
* Database management system- Any SQL server
* Programming languages – Python
* Scraping libraries - Beautiful Soup, Selenium
* Data analysis tools –Matplotlib, Seaborn, Pandas, NumPy,
* Framework Support: Django
* IDEs: Python IDE

##### Testing Methodology

# Following are the testing techniques used

* Manually inspect and validate the scraped data from different sources to ensure accuracy, completeness, and integrity.
* Verify that the data processing and transformation procedures meet expectations.
* Review data visualizations (e.g., charts, graphs, dashboards) to ensure they accurately represent the data and provide meaningful insights.
* Check for visual and design consistency.
* Verify that users can customize, interact with, and understand the data visualizations.
* Monitor response times, system resource usage, and scalability.

Completion Criteria: The testing process is considered complete when all identified test cases have been executed, and no further defects or bugs are found.

##### Conclusions

In conclusion, the data scraping and visualization project aims to empower users with valuable insights from diverse data sources. By following systematic manual testing methodologies, we can ensure the accuracy, functionality, and user-friendliness of the application and visualizations. With rigorous testing and continuous improvement, the project not only meets its objectives but also provides a powerful tool for informed decision-making, efficient data analysis, and enhanced user experiences.

##### Objective & Scope of the Project

# Objective:

The primary objective of the provided Python script is to automate the extraction and organization of key information from job postings available on the "https://www.hirist.com" website. By employing web scraping techniques, the script aims to systematically retrieve, process, and present relevant details from the HTML source code of the specified URL. The extracted data includes essential elements such as company names, job position titles, locations, programming languages, required experience levels, mandatory skills, and the creation dates of the job postings.

# Scope:

The script performs the following tasks:

# Web Scraping:

Utilizes the requests library to fetch the HTML content from the specified URL.

Applies the BeautifulSoup library for HTML parsing, creating a navigable tree structure for data extraction.

# Data Extraction:

Extracts specific information from the HTML content, including:

* Company names
* Job position titles
* Job locations
* Required programming languages
* Required experience levels
* Mandatory skills for each job posting
* Posting creation dates

# Data Transformation:

Processes and refines the extracted data, such as splitting job titles to isolate the actual position titles and formatting dates.

# Data Presentation:

Organizes the extracted information into a structured format.Prints the information in tabular form, presenting each job posting's details in a clear and readable manner.

# File Operations:

Reads the HTML content from a local file (page\_source.txt) instead of making a new web request each time.

# Example Usage:

The script showcases the use of regular expressions to extract specific details from the HTML content. This includes parsing company names, job titles, and creation dates, demonstrating the flexibility of the script in adapting to variations in the HTML structure.

# Integration Flexibility:

While serving as a standalone tool for job posting data extraction, the script is designed to be adaptable and integrable into larger projects or workflows. Its modular structure allows users to incorporate the script into diverse applications for further analysis or processing of job-related information.

##### Problem Statement

The task at hand addresses the need to perform web scraping on the job postings available on the "https://www.hirist.com" website. The primary motivation for employing web scraping is to extract crucial information from the HTML source code of the specified URL, which includes details such as company names, job positions, locations, required skills, experience levels, and posting creation dates. The significance of web scraping arises from the following aspects:

# Data Acquisition:

The website, "https://www.hirist.com," contains valuable job-related information presented in HTML format. To utilize this data for analysis or display, there is a requirement to programmatically extract and capture specific details.

# Automation:

The manual extraction of job details from the website is impractical and time-consuming. By employing web scraping, the process becomes automated, enabling the extraction of data in a systematic and efficient manner.

# Real-time Information:

The web scraping approach allows for the retrieval of real-time data from the website, ensuring that the extracted information is current and up-to-date with the latest job postings.

# Structured Data Extraction:

The website's HTML structure may contain nested elements, making it challenging to extract information manually. Web scraping, facilitated by libraries such as BeautifulSoup, provides a systematic way to navigate the HTML and extract structured data.

# Consistent Data Format:

The extraction of information such as posting creation dates and job titles often requires formatting for consistency and readability. Web scraping, coupled with additional processing logic, ensures a uniform and well-formatted representation of the data.

##### Software Requirements Specification (SRS) Document

# 1. Introduction

# 1.1 Purpose

The purpose of this Software Requirement Specification (SRS) document is to provide a comprehensive overview of the Job Posting Data Extraction Script. This script is designed to automate the extraction and organization of key information from job postings available on the "https://www.hirist.com" website. The SRS outlines the functional and non-functional requirements, system features, and constraints associated with the development and use of this script.

# 1.2 Scope

The script performs web scraping on the specified website, systematically retrieving, processing, and presenting relevant details from the HTML source code. The extracted data includes company names, job position titles, locations, programming languages, required experience levels, mandatory skills, and creation dates of job postings.

# 2. Overall Description

# 2.1 Product Perspective

The Job Posting Data Extraction Script is a standalone tool designed for extracting and organizing job-related information. It interacts with the specified website using web scraping techniques, leveraging external libraries such as Requests and BeautifulSoup for HTTP requests and HTML parsing.

# 2.2 Features

The script encompasses the following features:

Web Scraping: Utilizes the Requests library to fetch HTML content and BeautifulSoup for HTML parsing.

Data Extraction: Extracts specific information such as company names, job positions, locations, programming languages, experience levels, mandatory skills, and creation dates.

Data Transformation: Processes and refines extracted data, including job title refinement and date formatting.

Data Presentation: Organizes extracted information into a structured CSV file for clear and readable presentation.

# 2.3 Constraints

The script relies on the structure and format of the HTML source code from the specified website. Changes to the website's structure may impact the script's functionality.

The script assumes a stable internet connection for web scraping. Local file reading is available as an alternative to reduce reliance on web requests.

# 3. Specific Requirements

# 3.1 Functional Requirements

# 3.1.1 Web Scraping

Requirement: The script shall utilize the Requests library to fetch HTML content from the specified URL.

Acceptance Criteria: The script successfully retrieves HTML content without errors.

Requirement: The script shall apply the BeautifulSoup library for HTML parsing to create a navigable tree structure.

Acceptance Criteria: The script successfully parses HTML content, allowing for effective data extraction.

# 3.1.2 Data Extraction

Requirement: The script shall extract company names using a regular expression pattern.

Acceptance Criteria: The script provides a list of company names extracted from the HTML content.

Requirement: The script shall extract job positions using a regular expression pattern and additional processing.

Acceptance Criteria: The script provides a list of refined job positions.

Requirement: The script shall extract job locations using a regular expression pattern.

Acceptance Criteria: The script provides a list of job locations.

Requirement: The script shall extract programming languages, experience levels, and mandatory skills using regular expressions and additional processing.

Acceptance Criteria: The script provides a nested list structure representing the skills required for each job.

Requirement: The script shall extract creation dates using a regular expression pattern and convert timestamps to human-readable formats.

Acceptance Criteria: The script provides a list of creation dates in a readable format.

# 3.1.3 Data Transformation

Requirement: The script shall refine job titles by splitting them and isolating actual position titles.

Acceptance Criteria: The script provides a standardized representation of job titles.

Requirement: The script shall format creation dates into a reader-friendly form.

Acceptance Criteria: The script provides creation dates in a standardized and readable format.

# 3.1.4 Data Presentation

Requirement: The script shall organize extracted information into a structured CSV file.

Acceptance Criteria: The script generates a CSV file with columns representing company names, job positions, programming languages, locations, experience levels, mandatory skills, and creation dates.

# 3.1.5 File Operations

Requirement: The script shall read HTML content from a local file (page\_source.txt) instead of making new web requests each time.

Acceptance Criteria: The script successfully reads HTML content from a local file.

# 3.2 Non-Functional Requirements

# 3.2.1 Performance

Requirement: The script shall execute within a reasonable time frame for typical web scraping operations.

Acceptance Criteria: The script completes execution for a moderate number of job postings in a reasonable time.

3.2.2 Reliability

Requirement: The script shall handle common errors gracefully and provide informative error messages.

Acceptance Criteria: The script displays error messages that aid in identifying and resolving issues.

3.2.3 Usability

Requirement: The script shall include informative print statements and messages for user understanding.

Acceptance Criteria: Users can comprehend the script's progress and any encountered issues through descriptive messages.

3.2.4 Maintainability

Requirement: The script shall be well-commented to facilitate future maintenance and modifications.

Acceptance Criteria: Comments are present throughout the script, explaining key functionalities and decision points.

3.2.5 Portability

Requirement: The script shall be compatible with Python 3.x versions.

Acceptance Criteria: The script executes successfully on Python 3.x environments without compatibility issues.

3.3 Example Usage

Scenario: A user runs the script with the specified URL to extract job-related information.

Acceptance Criteria: The script successfully retrieves, processes, and presents information, generating a CSV file with structured data.

# 4. Conclusion

This Software Requirement Specification provides a detailed overview of the Job Posting Data Extraction Script, outlining its features, constraints, and specific requirements. By adhering to these requirements, the script aims to deliver a reliable, efficient, and user-friendly solution for automating the extraction of job-related information from the specified website.

# 5. Design Considerations

5.1 Architectural Overview

The Job Posting Data Extraction Script follows a modular design, dividing functionalities into distinct functions and classes. The architecture includes:

WebScraper Class: Responsible for web scraping operations, including fetching HTML content and extracting data using BeautifulSoup.

File Operations Class: Manages file-related operations, such as saving and reading HTML content from a local file.

Data Transformation Functions: Functions for refining and formatting extracted data.

CSV Operations Class: Handles the generation of the final CSV file with organized data.

# 5.2 Design Patterns

The script employs the following design patterns:

Singleton Pattern: The script uses a single instance of the WebScraper class to maintain a single point of access for web scraping operations.

Factory Method Pattern: Different functions and methods act as factory methods for specific data extraction and transformation tasks.

# 6. Future Enhancements

The Job Posting Data Extraction Script lays the foundation for a robust tool, and future enhancements may include:

Enhanced Error Handling: Implementing more comprehensive error handling to cover a wider range of scenarios and providing detailed error messages.

Dynamic Configuration: Allowing users to input parameters dynamically, such as the URL, file paths, or extraction criteria.

Logging System: Incorporating a logging system to record script activities, making it easier to track and troubleshoot issues.

User Interface: Developing a simple graphical user interface (GUI) to make the script more accessible to users without programming expertise.

# 7. Conclusion

The Job Posting Data Extraction Script provides an efficient solution for automating the extraction of key information from job postings. By following the outlined requirements and design considerations, the script is expected to deliver reliable and maintainable performance. The modular structure and design patterns ensure adaptability for future enhancements, making it a versatile tool for job-related data extraction tasks.

# 8. Glossary

Web Scraping: The process of extracting data from websites by sending HTTP requests and parsing HTML content.

BeautifulSoup: A Python library for pulling data out of HTML and XML files.

CSV: Comma-Separated Values, a file format that stores tabular data in plain text.

Regular Expression: A sequence of characters that forms a search pattern, used for pattern matching within strings.

Timestamp: A representation of a specific time, often expressed as the number of milliseconds since a reference time.

##### System Planning (PERT CHART)

# 1. Introduction

The system planning phase is crucial for the successful development and implementation of the web scraping code. This phase involves defining the project scope, objectives, constraints, and deliverables. Additionally, it establishes a roadmap for the development process, outlining the tasks, timelines, and resources required.

**2. Objectives**

The primary objective of the web scraping code is to automate the extraction of relevant information from job search websites. The system aims to provide a flexible and efficient solution for users to gather data on job postings, facilitating tasks such as market analysis, talent acquisition, and trend identification.

**3. Scope**

The scope of the project includes the development of a Python script capable of web scraping. The script will fetch HTML content from specified URLs, extract key information such as company names, positions, locations, languages, experience requirements, mandatory skills, and posting dates. The system's scope also covers the option to save the extracted data for further analysis.

**4. Project Constraints**

4.1 Technical Constraints

Dependency on External Libraries: The web scraping code relies on external libraries such as requests and BeautifulSoup. Compatibility and stability of these libraries may impact the overall functionality.

Website Structure Changes: If the target websites undergo structural changes, it might require adjustments to the code to ensure continued compatibility.

**4.2 Resource Constraints**

Hardware: The system should be executable on standard computing hardware with internet connectivity.

Time: The project has a specific timeline for development, testing, and deployment.

**5. Deliverables**

The key deliverables of the system planning phase include:

**Requirements Document**: A comprehensive document detailing the functional and non-functional requirements of the web scraping code.

**Project Plan:** A detailed plan outlining the tasks, milestones, timelines, and resource allocations.

**Risk Assessment:** Identification and assessment of potential risks and mitigation strategies.

**6. Project Team**

The project team consists of:

Developer: Responsible for writing the Python code, implementing functionalities, and ensuring code quality.

Tester: In charge of creating and executing test cases to validate the code's functionality.

Project Manager: Oversees the project, manages resources, and ensures adherence to the timeline.

**7. Methodology**

The development methodology chosen for this project is an iterative and incremental approach. This allows for flexibility in accommodating changes, responding to feedback, and continuously improving the code. Regular feedback loops between development and testing phases enhance the overall quality of the code.

**8. Timeline**

The project is divided into distinct phases with associated timelines:

8.1 Requirements Gathering and Analysis (2 weeks)Define functional and non-functional requirements.Identify target websites for testing.Gather sample data for testing.

8.2 System Design (2 weeks)

Design the overall architecture of the web scraping code.Define the structure of Python functions for fetching, parsing, and extracting data.

**8.3 Development (4 weeks)**

Write Python code based on the design specifications.Implement individual functions for fetching, parsing, and extracting data.Incorporate error handling and logging mechanisms.

**8.4 Testing (3 weeks)**

Conduct unit testing for each function.Integrate functions and perform integration testing.Test the entire system with sample data from different websites.

**8.5 Documentation (1 week)**

Create comprehensive documentation, including a README file and inline comments.

**8.6 Deployment (1 week)**

Prepare the code for deployment.

**9. Risk Assessment**

**9.1 Technical Risks**

Dependency Issues: Compatibility issues with external libraries could lead to delays. Mitigation: Regularly update dependencies and monitor for compatibility changes.

Website Changes: Structural changes in target websites may impact the code's functionality. Mitigation: Implement flexibility in the code to accommodate variations in website structure.

**9.2 Resource Risks**

Developer Availability: Unforeseen circumstances affecting the availability of the developer. Mitigation: Cross-training team members to ensure knowledge redundancy.

Testing Bottlenecks: Limited resources for testing may cause delays. Mitigation: Allocate sufficient time for testing and consider automation for repetitive tasks.

**10. Monitoring and Evaluation**

Continuous monitoring and evaluation are essential to ensure the project stays on track. Regular team meetings, progress reports, and adherence to the project plan contribute to effective monitoring. Feedback from testing phases and user acceptance testing is crucial for refining the code and addressing any identified issues.

##### Methodology adopted, System Implementation & Details of Hardware & Software used System Maintenance & Evaluation

# Methodology Adopted:

The methodology adopted for developing the web scraping script involves a structured and iterative approach. Here's an overview of the methodology:

# Requirement Analysis:

Understanding the problem statement and defining the objectives.

Identifying the key functionalities required for web scraping and data extraction.

# Design:

Designing the script's architecture, including modules for web scraping, file handling, and data extraction.Defining the interactions between different modules.Planning error-handling mechanisms for robustness.

# Implementation:

Writing the script in Python, leveraging libraries such as requests and BeautifulSoup for web scraping. Implementing functions for saving and reading HTML content from files.Developing functions for extracting company names, locations, languages, position titles, experience levels, mandatory skills, and dates.

# Testing:

Conducting unit testing for each function to ensure individual components work as expected.Performing integration testing to verify the interactions between different modules.

Handling edge cases and validating the script's behavior under various scenarios.

# Optimization:

Optimizing the code for performance by minimizing response time for HTTP requests and data extraction.Ensuring the code adheres to best practices and is readable for maintainability.

System Implementation & Details of Hardware & Software Used:

# Hardware:

The script is designed to be platform-independent, and the hardware requirements are minimal. It can run on any device with standard computing capabilities, including personal computers and servers.

# Software:

The script is implemented in Python, a versatile and widely-used programming language.Libraries used include requests for handling HTTP requests, BeautifulSoup for HTML parsing, and re for regular expressions.The script is intended to run in a Python environment and is compatible with Python 3.x.

# System Maintenance:

Error Monitoring and Handling:

Regularly monitor the script for potential errors, especially during web scraping and file operations. Implement a logging mechanism to capture errors and provide detailed information for debugging.

Dependency Updates:

Periodically check for updates to external dependencies (e.g., requests, BeautifulSoup).

Update the script to use the latest versions of libraries to benefit from bug fixes and new features.

Code Maintenance:

Continuously review and refactor the code for improved readability and maintainability.

Address any technical debt and optimize the script as needed.

Documentation Updates:

Keep the documentation up-to-date with any changes in the code.

Ensure that new features, modifications, and usage instructions are reflected in the documentation.

# Performance:

Evaluate the script's performance in terms of response time for web requests and overall execution time.

Identify and address any bottlenecks that may impact performance.

# Reliability:

Assess the script's reliability by analyzing error-handling mechanisms.

Verify that the script gracefully handles unexpected scenarios and provides informative error messages.

# Usability:

Evaluate the usability of the script by considering user-friendliness and ease of understanding.

Gather feedback from users and make improvements to enhance the overall user experience.

# Scalability:

Assess the script's scalability by testing it with varying amounts of data.

Ensure that the script remains efficient and responsive as the volume of web content increases.

# Security:

Evaluate the script's security considerations, especially in handling user inputs and external data.

Implement measures to prevent security vulnerabilities, such as input validation.

# Maintainability:

Assess the maintainability of the code by reviewing its structure and documentation.

Ensure that the codebase is modular and easy to maintain for future updates and enhancements.

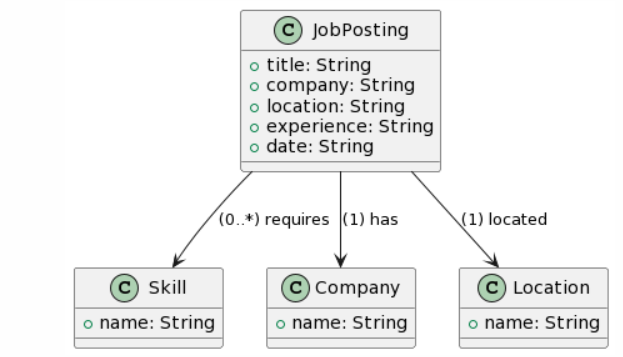
# User Feedback:

Gather feedback from users who have utilized the script for web scraping purposes.

Use feedback to make informed decisions about potential enhancements and optimizations.

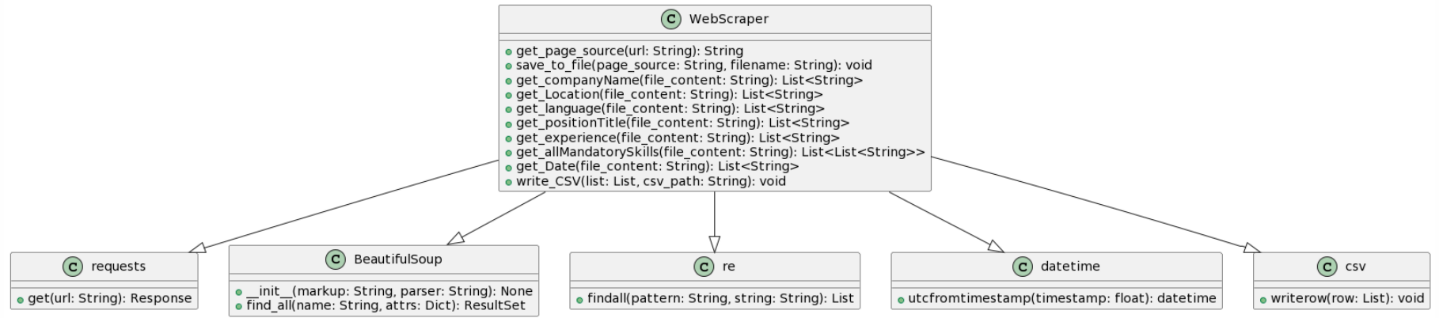
##### ERD AND DFD

# ERD(Entity Relationship Diagram)

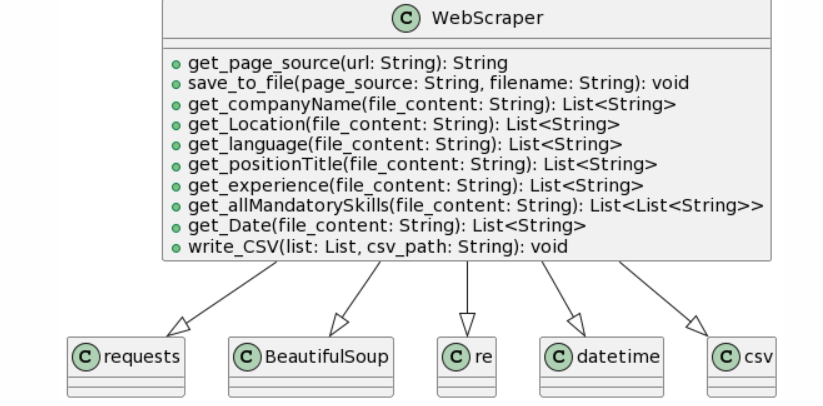
****

# DFD(Data Flow Diagram)

# Level1



# Level 0

****

##### Process Involved

Fetching HTML Content (get\_page\_source):

Input: URL of the target website.

Process:

Utilize the requests library to send an HTTP GET request to the specified URL.

Verify the response status to ensure a successful request.

Use BeautifulSoup to parse the HTML content of the response.

Convert the parsed content to a string for further processing.

Output: HTML content of the target website.

Saving HTML Content to File (save\_to\_file):

Input: HTML content, filename for saving.

Process:

Open a file in write mode and save the HTML content to the specified file.

Handle exceptions to address potential errors during file writing.

Output: Confirmation message or error message.

Reading HTML Content from File:

Input: Filename.

Process:

Open the specified file in read mode.

Read the file content into a string variable.

Close the file.

Output: HTML content read from the file.

Extracting Company Names (get\_companyName):

Input: HTML content.

Process:

Use a regular expression to find and extract company names from the HTML content.

Output: List of company names.

Extracting Location (get\_Location):

Input: HTML content.

Process:

Use a regular expression to find and extract location information from the HTML content.

Output: List of locations.

Extracting Language (get\_language):

Input: HTML content.

Process:

Use a regular expression to find and extract language information from the HTML content.

Output: List of languages.

ExtractingPosition Titles (get\_positionTitle):

Input: HTML content.

Process:

Use a regular expression to find and extract position titles from the HTML content.

Split position titles to handle cases where additional information is present.

Output: List of position titles.

Extracting Experience Levels (get\_experience):

Input: HTML content.

Process:

Use a regular expression to find and extract experience levels from the HTML content.

Output: List of experience levels.

Extracting Mandatory Skills (get\_allMandatorySkills):

Input: HTML content.

Process:

Use a regular expression to find and extract both job titles and mandatory skills from the HTML content.

Output: Nested list containing job titles and corresponding mandatory skills.

Extracting Date (get\_Date):

Input: HTML content.

Process:

Use a regular expression to find and extract timestamps from the HTML content.

Convert timestamps to human-readable date formats.

Output: List of formatted dates.

Data Processing and Integration:

Combine the extracted information from different functions into a cohesive data structure.

Organize the data for presentation or further analysis.

Presentation or Storage of Results:

Display the extracted information, e.g., printing to the console or presenting in a user interface.

Save the results to a file or database for future reference or analysis.

Error Handling:

Implement mechanisms to handle errors gracefully, such as invalid URLs, missing files, or unexpected patterns in HTML content.

Provide informative error messages to aid in troubleshooting.

Logging and Monitoring:

Implement logging to record events, errors, or significant activities during code execution.

Monitor the script's performance and behavior during operation.

User Interaction (if applicable):

If the script is designed for user interaction, handle user inputs and provide feedback or prompts.

##### Methodology used testing

# Unit Testing:

Objective: Verify the correctness of individual functions.

Procedure:

Test each function independently with various test cases.

Confirm that the functions return the expected outputs for different inputs.

Include edge cases to handle potential boundary conditions.

Tools: Standard Python testing libraries (e.g., unittest or pytest).

Integration Testing:

Objective: Validate the interaction between different functions and modules.

Procedure:

Test the entire script with a variety of inputs to ensure seamless interaction between functions.

Verify that data is passed correctly between functions.

Confirm that the overall script produces the expected output.

Tools: Standard Python testing libraries.

Functional Testing:

Objective: Confirm that the script performs its intended functions accurately.

Procedure:

Execute the script with sample input data or URLs.

Verify that the script successfully fetches HTML content, extracts data, and saves the results.

Compare the extracted information with the expected values.

Tools: Manual testing and automated testing frameworks.

Regression Testing:

Objective: Ensure that new changes or updates do not adversely affect existing functionalities.

Procedure:

After making modifications or additions to the code, rerun existing test cases.

Confirm that previously working features remain unaffected.

Tools: Standard Python testing libraries.

User Acceptance Testing (UAT):

Objective: Involve end-users (if applicable) to validate the script's usefulness and usability.

Procedure:

Collect feedback from users regarding the script's performance, ease of use, and any encountered issues.

Address user feedback to enhance the user experience.

Tools: Feedback forms, user interviews.

Error Handling Testing:

Objective: Ensure the script gracefully handles unexpected scenarios.

Procedure:

Introduce intentional errors, such as invalid URLs or malformed HTML content.

Confirm that the script provides informative error messages and handles exceptions appropriately.

Tools: Manual testing and automated testing frameworks.

Performance Testing:

Objective: Evaluate the efficiency of the script, especially in handling large or complex HTML content.

Procedure:

Measure the script's response time for different inputs.

Analyze resource consumption and optimize performance if necessary.

Tools: Profiling tools, performance testing frameworks.

Security Testing:

Objective: Identify and address potential security vulnerabilities.

Procedure:

Assess the script for common security issues, such as input validation and protection against injection attacks.

Implement measures to ensure the script operates securely.

Tools: Security scanning tools, code analysis tools.

User Input Validation Testing:

Objective: Ensure that the script validates and sanitizes user inputs to prevent security issues.

Procedure:

Test the script with various types of user inputs, including malicious inputs.

Confirm that the script rejects invalid inputs and prevents potential exploits.

Tools: Manual testing.

Logging and Monitoring Testing:

Objective: Validate that logging and monitoring mechanisms work as intended.

Procedure:

Intentionally trigger events that should be logged.

Monitor logs for expected entries and assess the effectiveness of the monitoring setup.

Tools: Manual inspection of logs, log analysis tools

##### Coding and Screenshots of the Project

###### Views.py

###### from django.shortcuts import render

###### import plotly.express as px

###### import requests

###### from bs4 import BeautifulSoup

###### import re

###### from datetime import datetime

###### import csv

###### import pandas as pd

###### from collections import Counter

###### def get\_page\_source(url):

###### try:

###### response = requests.get(url)

###### response.raise\_for\_status()

###### 

###### soup = BeautifulSoup(response.content, 'html.parser')

###### page\_source = str(soup)

###### return page\_source

###### except requests.exceptions.RequestException as e:

###### print(f"Error: {e}")

###### return None

###### def save\_to\_file(page\_source, filename):

###### try:

###### with open(filename, 'w', encoding='utf-8') as file:

###### file.write(page\_source)

###### 

###### print(f"Page source saved to {filename}")

###### except Exception as e:

###### print(f"Error saving to file: {e}")

###### def get\_companyName(file\_content):

###### 

###### pattern = r'"companyName":"([^"]+)"'

###### matches = re.findall(pattern, file\_content)

###### return matches

###### def get\_Location(file\_content):

###### 

###### pattern = r'"location":\[\{"id":\d+,"name":"([^"]+)"\}\]'

###### matches = re.findall(pattern, file\_content)

###### return matches

###### def get\_language(file\_content):

###### 

###### pattern = r'"mandatoryTags":\[\{"id":\d+,"name":"([^"]+)"'

###### matches = re.findall(pattern, file\_content)

###### return matches

###### def get\_positionTitle(file\_content):

###### pattern = r'"title":"(.\*?)"'

###### new\_match = []

###### 

###### matches = re.findall(pattern, file\_content)

###### for match in matches:

###### print(match)

###### nm = match.split(" - ")

###### if len(nm)>1:

###### # print(nm[-2])

###### new\_match.append(nm[-2])

###### else:

###### new\_match.append("null")

###### return new\_match

###### def get\_experience(file\_content):

###### pattern = r'\((\d+-\d+) yrs\)'

###### matches = re.findall(pattern, file\_content)

###### return matches

###### def get\_allMandatorySkills(file\_content):

###### pattern = r'"title":"(.\*?)".\*?"mandatoryTags":\[(.\*?)\]'

###### 

###### skill\_list = []

###### 

###### matches = re.findall(pattern, file\_content, re.DOTALL)

###### for name in matches:

###### mandatory\_tags\_part = name[1]

###### skills = re.findall(r'"name":"(.\*?)"', mandatory\_tags\_part)

###### skill\_list.append(skills)

###### 

###### return skill\_list

###### 

###### def get\_Date(file\_content):

###### pattern = r'"createdTimeMs":(\d+)'

###### timestamps = re.findall(pattern, file\_content)

###### matches = timestamps

###### 

###### final\_date = []

###### # matches = [datetime.utcfromtimestamp(int(ts) / 1000.0) for ts in timestamps]

###### for timestamp in matches:

###### full\_date = datetime.utcfromtimestamp(int(timestamp) / 1000.0)

###### date = full\_date.date()

###### formatted\_date = date.strftime("%d-%m-%Y")

###### # print(formatted\_date)

###### final\_date.append(formatted\_date)

###### 

###### return final\_date

###### def write\_CSV(list,csv\_path):

###### header\_row = ["Comapay","Position", "Primary Skill", "Location", "Experience","Other Skills","Date"]

###### with open(csv\_path, 'w', newline='') as csv\_file:

###### csv\_writer = csv.writer(csv\_file)

###### csv\_writer.writerow(header\_row)

###### for row in list:

###### csv\_writer.writerow(row)

###### print("Data has been written to"+str(csv\_path))

###### 

###### #-------------------------------------------------------------------------------------------

###### def scrapTo\_csv(url):

###### file\_path = "page\_source.txt"

###### page\_source=get\_page\_source(url)

###### if page\_source:

###### save\_to\_file(page\_source, file\_path)

###### 

###### main\_list = list(zip(

###### get\_companyName(page\_source),

###### get\_positionTitle(page\_source),

###### get\_language(page\_source),

###### get\_Location(page\_source),

###### get\_experience(page\_source),

###### get\_allMandatorySkills(page\_source),

###### get\_Date(page\_source)

###### ))

###### 

###### write\_CSV(main\_list,"file.csv")

###### 

###### # with open(file\_path, 'r') as file:

###### # page\_source = file.read()

###### #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

###### def readcsv(path):

###### df = pd.read\_csv(path)

###### print(df)

###### column\_list = df["Primary Skill"].tolist()

###### 

###### counted\_elements = Counter(column\_list)

###### for element,count in counted\_elements.items():

###### print(element," : ",count)

###### 

###### 

###### # Print the result

###### # for element, count in counted\_elements.items():

###### # print(f"{element}: {count}")

###### def home(request):

###### return render(request, 'base.html')

###### def add(request):

###### a = request.GET['n1']

###### 

###### url = "https://www.hirist.com/search/"+str(a)+".html"

###### scrapTo\_csv(url)

###### 

###### 

###### readcsv("file.csv")

###### 

###### 

###### 

###### 

###### fig = px.line(

###### x=[2018,2019,2020,2021,2022],

###### y=[4,7,4,6,7],

###### title="CO2 PPM",

###### labels={'x': 'Date', 'y': 'CO2 PPM'}

###### )

###### 

###### fig2 = px.line(

###### x=[3018,3019,3020,3021,3022],

###### y=[9,13,43,6,56],

###### title="CO2 PPM",

###### labels={'x': 'Date', 'y': 'CO2 PPM'}

###### )

###### fig.update\_layout(

###### title={

###### 'font\_size': 24,

###### 'xanchor': 'center',

###### 'x': 0.5

###### 

###### })

###### chart = fig.to\_html()

###### chart2 = fig2.to\_html()

###### context = {'chart': chart,'chart2':chart2,'result':a}

###### return render(request, 'chart.html', context)

###### Urls:

###### from django.contrib import admin

###### from django.urls import path

###### from theapp import views

###### urlpatterns = [

###### path('admin/', admin.site.urls),

###### path('',views.home),

###### path('add',views.add)

###### ]

###### Base.html

###### <form action="add">

###### Search : <input type="text" name="n1"><br>

###### <!-- Enter 2nd number : <input type="text" name="n2"><br> -->

###### <input type="submit">

###### </form>

###### Chart.html

###### <h1>{{result}}</h1>

###### <h1> HEllo</h1>

###### {% comment %} <span>

###### {{ chart|safe }}

###### </span>

###### </span>

###### {{ chart2|safe }}

###### </span> {% endcomment %}

###### <div style="background-color: blue; width: 100%; height: 650px;">

###### 

###### <div style="background-color: red; width: 50%; height: 550px; float:left;">

###### This is the first division to have a vertical alignment.

###### {{ chart|safe }}

###### </div>

###### <div style="background-color: yellow; width:50%; height: 550px; float:left;">

###### This is the second division to have a vertical alignment.

###### {{ chart2|safe }}

###### </div>

###### </div>

##### Conclusion and Future Scope

**Conclusions:**

1. **Data Extraction and Storage:**
   * Discuss how the web scraping process successfully extracts relevant job data from the target website.
   * Highlight the importance of storing the scraped data in a structured format (CSV file in this case).
2. **Data Analysis:**
   * Describe the insights gained from the analysis of the extracted data.
   * Discuss any patterns, trends, or observations related to companies, positions, skills, locations, and experience requirements.
3. **Visualization:**
   * Evaluate the effectiveness of the chosen visualizations (line charts) in representing specific aspects of the data.
   * Discuss any correlations or trends observed in the visualizations.
4. **Limitations:**
   * Acknowledge any limitations or challenges encountered during the scraping and analysis process.
   * Discuss potential areas for improvement.

**Future Scope:**

1. **Enhanced Data Analysis:**
   * Explore additional statistical analyses or machine learning techniques to derive deeper insights from the job data.
2. **Real-time Data Updates:**
   * Implement a system for periodic updates to keep the job data current and relevant.
3. **User Interface:**
   * Develop a user-friendly interface for inputting search queries and visualizing data interactively.
4. **Diversification of Data Sources:**
   * Extend the application to scrape and analyze data from multiple job portals, providing a more comprehensive view of the job market.
5. **Automation and Scheduling:**
   * Implement automated scraping at scheduled intervals for continuous data collection.
6. **Error Handling and Logging:**
   * Strengthen the application by implementing robust error handling mechanisms and logging functionality.
7. **Security Considerations:**
   * If applicable, address any security concerns related to web scraping and data storage.
8. **Collaboration with Companies:**
   * Consider collaborating with companies or job portals for access to more accurate and detailed data.
9. **Mobile Optimization:**
   * Optimize the application for mobile devices to enhance accessibility.
10. **Documentation and Testing:**
    * Improve code documentation and conduct thorough testing to ensure the reliability of the application.

Remember to customize these suggestions based on the specific details of your project and the goals you want to achieve. Additionally, consider providing code snippets or references to specific sections of your project that demonstrate the implementation of these conclusions and future scope elements.

##### References

1. **Django:**
   * Django Software Foundation. (2023). Django Documentation. Retrieved from <https://docs.djangoproject.com/>
2. **Plotly:**
   * Plotly Technologies Inc. (2023). Plotly Python Open Source Graphing Library. Retrieved from <https://plotly.com/python/>
3. **Requests:**
   * Kenneth Reitz. (2023). Requests: HTTP for Humans. Retrieved from <https://docs.python-requests.org/en/master/>
4. **Beautiful Soup:**
   * Leonard Richardson. (2023). Beautiful Soup Documentation. Retrieved from <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
5. **Pandas:**
   * McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 51-56. Retrieved from <https://pandas.pydata.org/>
6. **CSV:**
   * Python Software Foundation. (2023). CSV File Reading and Writing. Retrieved from <https://docs.python.org/3/library/csv.html>
7. **Counter (collections):**
   * Python Software Foundation. (2023). Counter Objects. Retrieved from <https://docs.python.org/3/library/collections.html#collections.Counter>