McJobs

A PROJECT REPORT

submitted by

JESWIN ELDHO SAJI (MUT20CS073)

to

the APJ Abdul Kalam Technological University in partial fullfilment of the requirements for the award of the Degree

of

Bachelor of Technology

In

Computer Science & Engineering

Department of Computer Science & Engineering

Muthoot Institute of Technology and Science

Varikoli PO, Puthencruz - 682308

JUNE 2023

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

JESWIN ELDHO SAJI

Place:

Date:

Muthoot
Institute of Technology & Science

CERTIFICATE

This is to certify that the report entitled **"McJobs"**, *submitted by* **JESWIN ELDHO SAJI (MUT20CS073)** *to Muthoot Institute of Technology and Science, Varikoli for the award of the degree of Bachelor of Technology in Computer Science & Engineering is a bonafide record of the project work carried out by her, under our supervision and guidance. The content of the report, in full or parts have not been submitted to any other Institute or University for the award of any other degree or diploma.*

Ms.Asha Raj	Ms.Sheena K V	Dr.Anand Hareendran S
Project Guide	Project Coordinator	Head of the Department

Place

Date

ACKNOWLEDGMENT

I am grateful to almighty who has blessed me with good health, committed and continuous interest throughout the project work.

I express my sincere thanks to my guide, **Ms. Asha Raj**, Assistant Professor, Department of Computer Science And Engineering, Muthoot Institute of Technology and Science and **Dr. Anand Hareendran S**, Professor, Head Of the Department, Muthoot Institute of Technology and Science for their guidance and support which were instrumental in all the stages of the project work and without whom the project could not have been accomplished.

In particular, I also wish to express my sincere appreciation to **Dr. Anand Hareendran S**, Head Of the Department, Muthoot Institute of Technology and Science, who was willing to spend his precious time to give some ideas and suggestion towards this project.

I am grateful to my project coordinator **Ms. Sheena K V** Assistant Professor, Department of Computer Science And Engineering, Muthoot Institute of Technology and Science, for her guidance and support.

I would like to thank **Dr. Neelakantan P.C.**, Principal, Muthoot Institute of Technology and Science, Varikoli for providing us all the necessary facilities.

The last but not the least, I extend my sincere thanks to the entire teaching and non-teaching staff of Computer Science And Engineering of Muthoot Institute of Technology and Science for their help and co-operation throughout our project work.

ABSTRACT

Our mini project focuses on the development of a web scraping application that utilizes Selenium, Beautiful Soup, and Flask to collect job links from various websites. Additionally, the project incorporates a resume builder feature to assist users in creating their resumes based on the job listings they discover.

The web scraping process combines the power of Selenium and Beautiful Soup. Selenium, a web automation tool, is employed to navigate and interact with dynamic web pages, allowing the application to access job portals that require user interactions. Beautiful Soup, a Python library for web scraping, is then utilized to extract relevant data from the targeted websites.

The application, built using Flask, provides a user-friendly interface for users to interact with the web scraping functionalities. Users can specify their search criteria, such as location, job type and the application will scrape job links from multiple websites accordingly. The collected data is presented in a structured format within the web app, enabling users to browse and explore the available job opportunities conveniently.

In addition to web scraping, the project incorporates a resume builder feature. Based on the job listings discovered, the application provides a resume template and allows users to input their relevant information. The resume builder streamlines the process of creating tailored resumes for specific job applications, enhancing users' chances of securing employment.

By combining Selenium, Beautiful Soup, and Flask, this mini project offers a comprehensive solution for job searching and resume building. It eliminates the need for manual navigation through multiple job portals and provides a centralized platform for collecting and exploring job links. The inclusion of the resume builder feature enhances the overall user experience by facilitating the creation of personalized resumes based on the scraped job listings.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Web scraping is the process of extracting data from websites, and it can be a valuable tool for gathering information from job portals. Job portals are online platforms where employers post job listings, and individuals search and apply for available positions. Web scraping allows you to automate the extraction of job-related data, such as job titles, company names, locations, job descriptions, and application deadlines, from these portals.

Our mini project focuses on the development of a web scraping application designed to revolutionize the job search process. With the ever-increasing reliance on online job portals, it has become essential to provide job seekers with a centralized platform that aggregates job listings from various websites. Leveraging the power of Flask, Beautiful Soup, and Selenium, our application collects job links and presents them in a user-friendly web app interface. This project aims to simplify and streamline the job search experience by automating data collection, offering advanced filtering options, and providing a consolidated platform for accessing job opportunities. Moreover, we have incorporated a resume builder feature to assist users in tailoring their resumes based on the job listings they discover, elevating their chances of success in securing their desired positions. By saving time, improving efficiency, and empowering job seekers, our mini project aims to make a significant impact in the realm of job search and application processes.

However, it's important to note that when web scraping job portals, you must respect the website's terms of service and adhere to legal and ethical guidelines. Some job portals may have specific restrictions on scraping their data, so it's crucial to review and comply with their policies.

1.2 SCOPE

Enhanced Job Filtering: Expand the filtering capabilities of the web scraping application to include more advanced criteria such as salary range, experience level, or specific skills required.

User Accounts and Saved Searches: Implement a user account system that allows users to create profiles, save their preferred job search criteria, and receive notifications when new job listings matching their criteria are added. This will provide a personalized and convenient experience for users.

Job Application Integration: Integrate with popular job application platforms, such as LinkedIn or Indeed, to enable users to apply for jobs directly from the web app. This will streamline the application process and eliminate the need for users to visit each job portal individually. Sentiment Analysis and Company Insights: Incorporate sentiment analysis to gather insights on companies and job listings. Analyze reviews and ratings to provide users with information about the work environment, employee satisfaction, and overall company reputation. This will help users make informed decisions before applying for jobs.

1.3 MOTIVATION

Efficiency: The project aims to improve the efficiency of the job search process by developing a web scraping application. Manual searching and gathering of job links from various websites can be time-consuming. By automating the process, users can access multiple job portals simultaneously, saving time and effort.

Centralization: The project provides a centralized platform for job seekers to access job listings from different websites. Instead of visiting each portal individually, users can rely on the web app built using Flask and Beautiful Soup. This streamlines the job search process and enhances the user experience.

Streamlined Applications: Integrating with popular job application platforms simplifies the application process for users. By applying directly from the web app, users save time and enjoy a seamless experience without navigating multiple portals

CHAPTER 2

PROPOSED WORK

Web scraping involves extracting data from websites by automatically fetching and parsing. Here are a few examples of proposed works that you can accomplish using web scraping: Data Collection: Retrieve information from websites, such as product details, prices, reviews, ratings, and specifications. This data can be used for market research, price comparison, or building a product catalog. Social Media Analysis: Scrape social media platforms to collect data on user profiles, posts, comments, and engagement metrics. This information can be used for sentiment analysis, trend identification, or targeted marketing. Job Market Monitoring: Extract job postings, companies, locations, and other relevant details from job portals to analyze trends in the job market, identify popular skills, or track hiring patterns. Competitive Intelligence: Scrape competitor websites to monitor their pricing strategies, product offerings, marketing campaigns, or customer reviews. Government Data Extraction: Collect public data from government websites, such as census data, public records, or legislative information, for analysis or research purposes.

2.1 OBJECTIVES

To create a web application prototype that can combine web scraping process, aggregation and dashboard. We aggregate job suggestions from multiple job portals on our scraping portal, providing a comprehensive selection of opportunities. To develop an innovative online job search platform that connects job seekers with multiple job opportunities sourced from various job portals.The primary objective of web scraping is to extract specific data from websites. This could include structured data such as product information, prices, ratings, reviews, or unstructured data like articles, news updates, or social media posts. Web scraping allows you to gather data from multiple sources and aggregate it into a single dataset or platform. This helps in consolidating information from different websites or platforms for analysis, comparison, or presen-

tation purposes.Web scraping can provide training data for machine learning models or natural language processing tasks. By scraping websites that contain labeled or annotated data, you can build datasets for training and evaluating ML models.

2.2 PROBLEM STATEMENT

The purpose of this research is to develop a web application that utilizes Web Scraping and Regular Expression methods to retrieve job vacancy information. The objective is to create a user-friendly web app that efficiently collects and displays the extracted data. By achieving this, we aim to enhance the job search process by providing users with a convenient platform to access and explore relevant job vacancies.

2.3 EXISTING SYSTEM AND PROPOSED SOLUTION

2.3.1 EXISTING SYSTEM

The existing system in web scraping for jobs involves the use of automated scripts or tools to extract job-related information from various job portals, company websites, and other online sources. These systems aim to gather job postings, company details, job descriptions, and other relevant data for analysis, aggregation, or display on job search platforms.

Here are some key components and characteristics of the existing system:

Data Sources: The existing system targets popular job portals such as Indeed, LinkedIn, Glassdoor, Monster, or specialized industry-specific job boards. It may also scrape career pages of individual companies to collect job listings directly from their websites.

Scraping Techniques: The system utilizes web scraping techniques to automatically fetch and parse HTML code from the targeted websites. It may employ libraries or frameworks like BeautifulSoup, Scrapy, or Selenium to handle the scraping process.

Job Posting Extraction: The system identifies and extracts job postings from the web pages using specific HTML tags, class names, or XPath queries. It retrieves details such as job title, company name, location, salary, required qualifications, and application deadlines.

Data Normalization: The extracted job data is normalized to ensure consistent for-

matting and structure across different sources. This involves cleaning and organizing the data, standardizing values (e.g., location names), and handling variations in data presentation.

Data Storage: The system typically stores the scraped job data in a structured format, such as a database or CSV file. This allows for efficient storage, retrieval, and further processing of the collected information.

Job Monitoring and Updates: Some systems incorporate job monitoring capabilities to track changes in job listings over time. This involves periodic re-scraping of the targeted websites to identify new job postings, removed listings, or updates to existing job information.

Data Presentation: The collected job data is often presented to users through user-friendly interfaces or integrated into job search platforms. It may include features like pagination, job details view, saved searches, and alerts for new job postings matching specified criteria.

2.3.2 PROPOSED SYSTEM

The proposed system aims to enhance the existing web scraping process for jobs by incorporating additional features and improvements. The system focuses on collecting comprehensive job data, improving data quality, enhancing user experience, and ensuring legal compliance. Here are the key components of the proposed system:

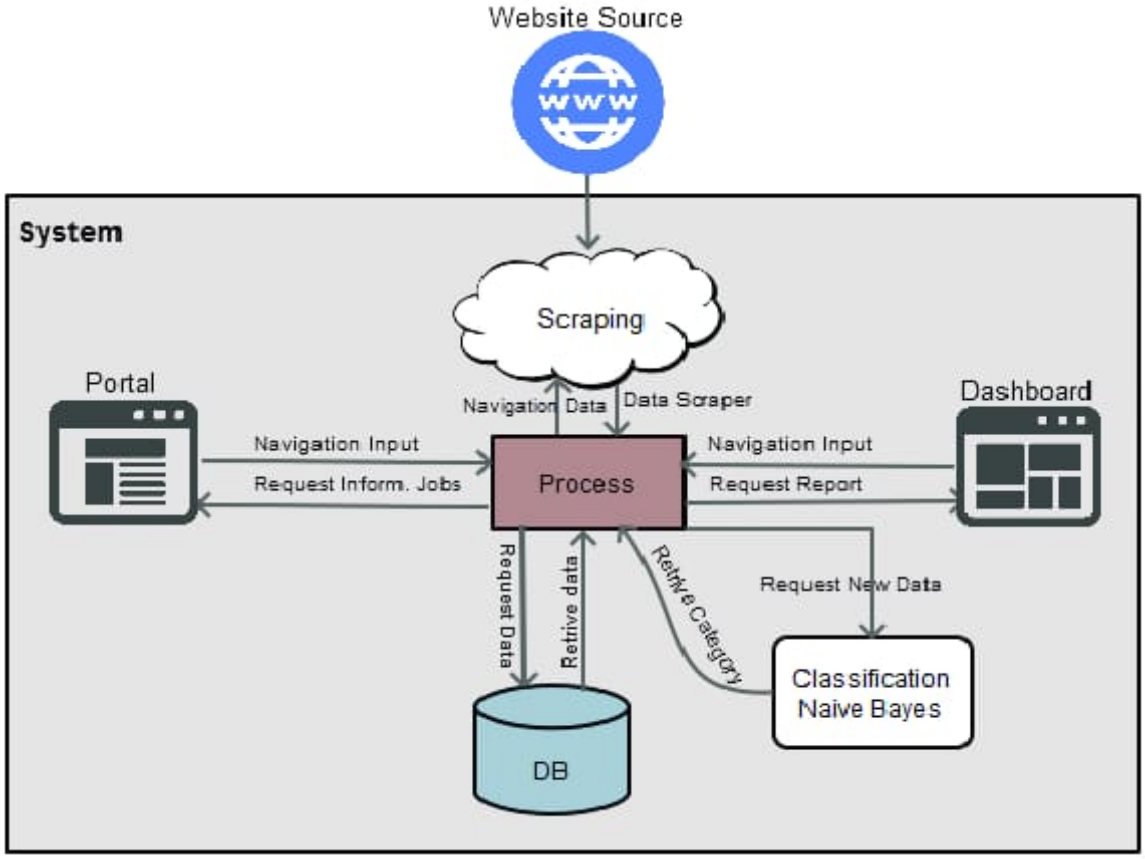
Expanded Data Collection: The proposed system targets a broader range of job portals and sources, including both general job boards and industry-specific platforms. It seeks to cover a wide variety of job postings to provide a comprehensive and diverse dataset for users.

Advanced Job Information Extraction: The system utilizes advanced scraping techniques, natural language processing, and machine learning algorithms to extract detailed job information. This includes extracting key skills, required qualifications, company profiles, benefits, and additional contextual information from job descriptions.

CHAPTER 3

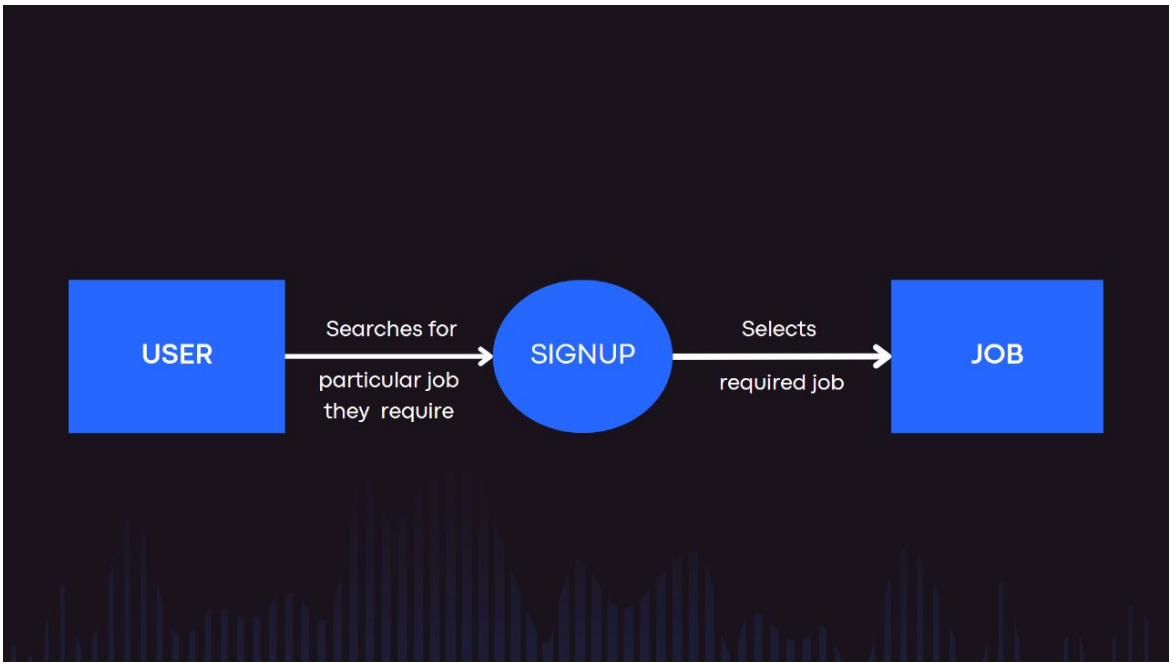
PROJECT DESIGN

3.1 SYSTEM ARCHITECTURE

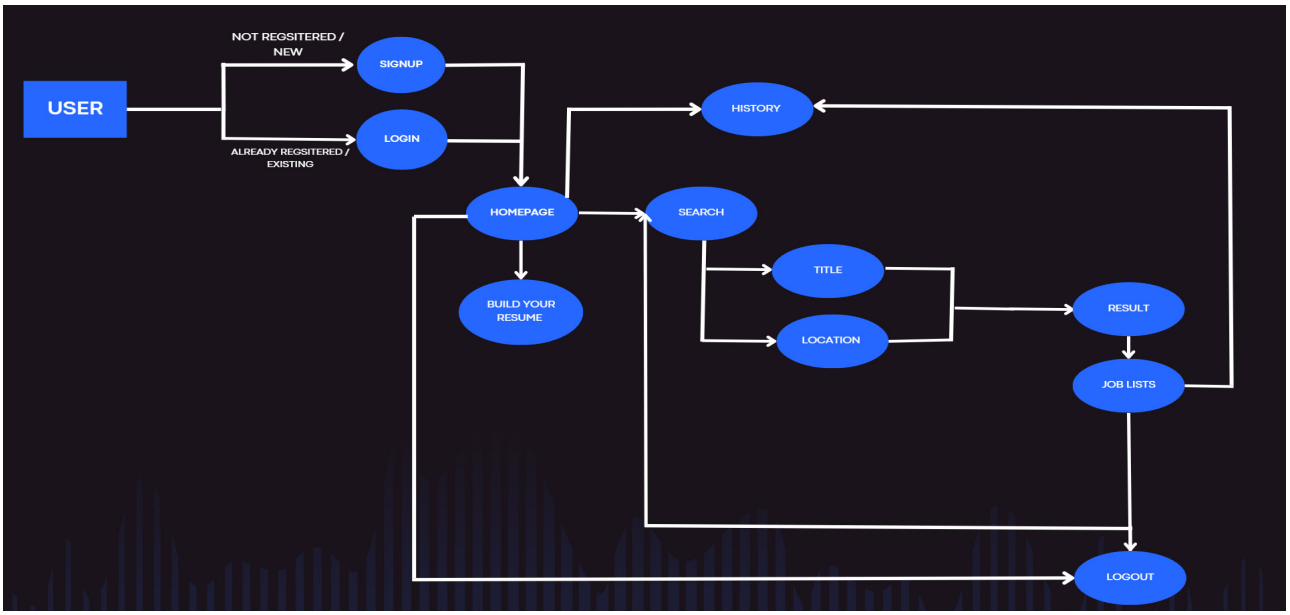


3.2 DATA FLOW DIAGRAM

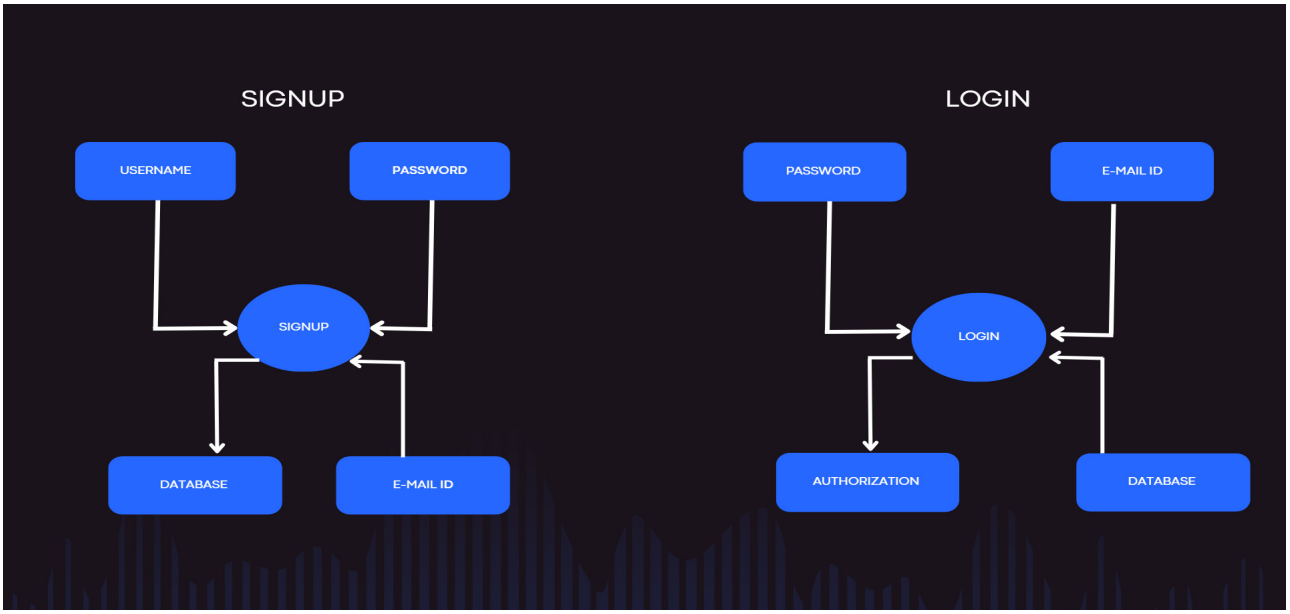
3.2.1 DFD LEVEL 0

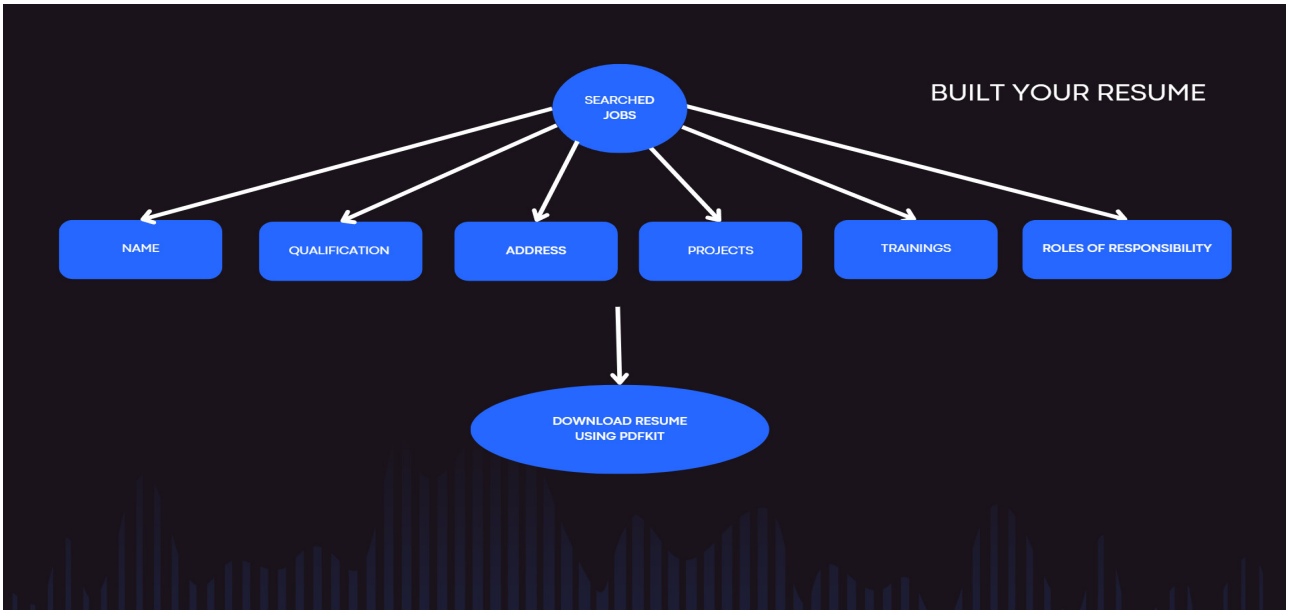


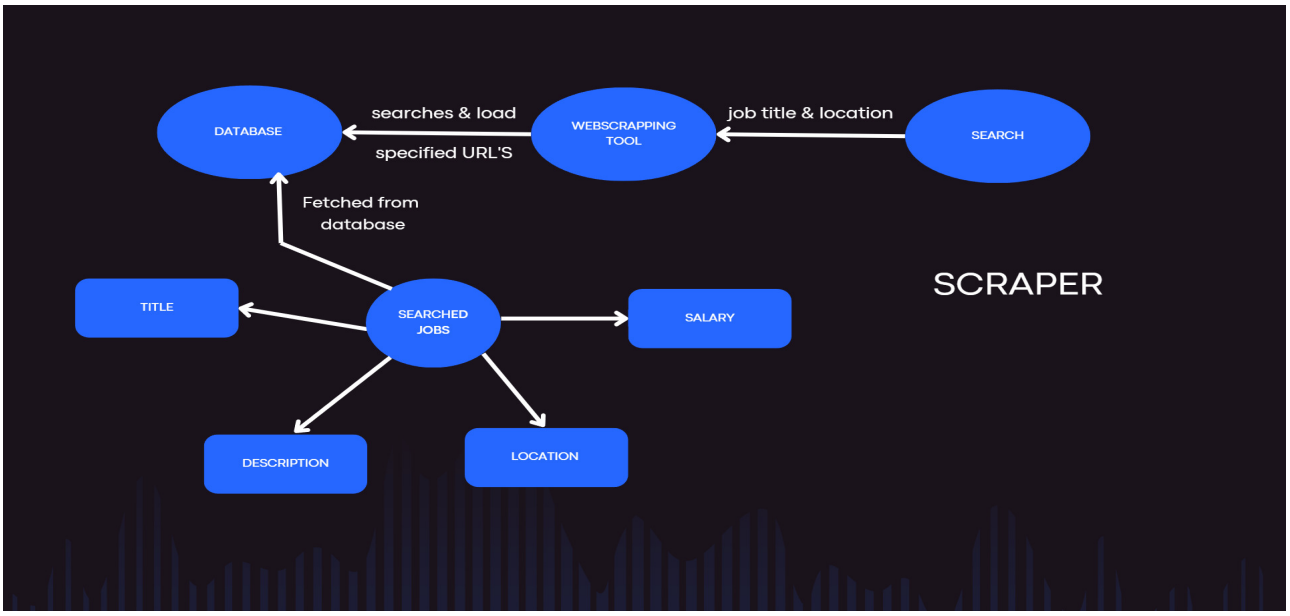
3.2.2 DFD LEVEL 1



3.2.3 DFD LEVEL 2







3.3 DATABASE TABLE DESIGN

DATABASE STRUCTURE

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	username	varchar(100)	utf8mb4_general_ci		Yes	*NULL*			Change Drop M
2	email	varchar(100)	utf8mb4_general_ci		No	*None*			Change Drop M
3	password	varchar(100)	utf8mb4_general_ci		No	*None*			Change Drop M
4	history	mediumtext	utf8mb4_general_ci		No	*None*			Change Drop M

3.4 GUI DESIGN

Graphical User Interface (GUI) Design focuses on the interaction between a user and a computer. This includes everything from starting or logging into the system to presenting the desired inputs and outputs. The overall flow of screens and messages that facilitate this interaction is known as a dialogue.

There are several guidelines for User Interface Design that can be satisfied by a web page. Some of these guidelines include:

1. Clarity: The interface should be easy to understand and use, with clear and concise language and intuitive navigation.
2. Consistency: The interface should be consistent in its design and behavior, with similar elements behaving in similar ways.
3. Efficiency: The interface should be designed to minimize the number of steps required to complete a task, making it easy for users to accomplish their goals.
4. Flexibility: The interface should be flexible and adaptable, allowing users to customize it to their needs and preferences.
5. Feedback: The interface should provide clear and immediate feedback to the user, letting them know the results of their actions.
6. Error prevention and recovery: The interface should be designed to prevent errors from occurring, and to help users recover from errors when they do occur.

3.4.1 COMPONENTS IN GUI

The following represents the components used in the development of the GUI

- Button: Used in the web app for registering, logging inside the app, placing orders etc.
- Date: It comes under one of the types of input in HTML used in obtaining dates of past orders made by the user
- Its of type input which is used in the app for the search of food items in the food list
- Used in several places in the app for inputting name, password, name of the food item, number of plates required etc.
- Images: Several images are used in the web app to enhance the user experience.

- Table: Several tables are shown in the web app on the user's side and on the admin's side to display statistics, costs, number of plates etc.
- Drop-down list: This component is used to select an item from the list of items stored inside a drop-down list in HTML.

3.5 API DESIGN

The Application Programming Interface (API) which is used in this system is Django REST framework whose details are given below:

3.5.1 INTRODUCTION:

REST APIs are an industry-standard way for web services to send and receive data. They use HTTP request methods to facilitate the request-response cycle and typically transfer data using JSON, and more rarely - HTML, XML and other formats.

3.5.2 REST API:

REST (Representational State Transfer) is a standard architecture for building and communicating with web services. It typically mandates resources on the web are represented in a text format (like JSON, HTML, or XML) and can be accessed or modified by a predetermined set of operations. An API (Application Programming Interface), as the name suggests, is an interface that defines the interaction between different software components. Web APIs define what requests can be made to a component, how to make them (for example, a GET request), and their expected responses.

3.6 SYSTEM REQUIREMENTS

The web application is designed to be compatible with a wide range of devices and browsers. It can be accessed and run on popular web browsers like Google Chrome, Mozilla Firefox or Microsoft Edge, ensuring compatibility across different platforms such as Android phones, desktops, laptops, and tablets. It is recommended to have

a system that meets certain hardware requirements. These requirements include a processor with a minimum specification of an i3 or higher, although an i5 processor is recommended for smoother performance. A minimum of 8GB of RAM is expected to handle the application's processes efficiently. In terms of storage, it is recommended to have a hard disk with a capacity of 500GB or above. This ensures sufficient space to store the web application's files, databases, and any additional data that may be generated or accessed during its usage. For user input and interaction, a standard USB optical mouse and keyboard are required to navigate and interact with the web application effectively. By meeting these system requirements, users can expect optimal performance, responsiveness, and usability while running the web application.

CHAPTER 4

IMPLEMENTATION

4.1 CODE SNIPPETS

```python
from flask import Flask, render_template, request, redirect, url_for, session, send_file
from indeed_jobs_webscrapping import scrape_indeed_jobs
import re
import MySQLdb.cursors
import pdfkit

app = Flask(__name__)

db = MySQLdb.connect(
    host='localhost',
    user='root',
    password='',
    database='users',
    cursorclass=MySQLdb.cursors.DictCursor
)

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

@app.route('/login.html', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        cursor = db.cursor()

        # Query the database for the user with the entered email
        cursor.execute('SELECT * FROM users WHERE email = %s', (email,))
```

code.app.py1

```python
indeed_jobs_webscrapping.py
import pandas as pd
from selenium import webdriver
from bs4 import BeautifulSoup


def scrape_indeed_jobs(job_title, location):
    # Create Chrome webdriver
    driver = webdriver.Chrome()
    url = "https://in.indeed.com/"
    driver.get(url)

    what_box = driver.find_element(by="xpath", value="//input[@id='text-input-what']")
    where_box = driver.find_element(by="xpath", value="//input[@id='text-input-where']")
    find_jobs_button = driver.find_element(by="xpath", value="//button[@class='yosegi-InlineWhatWhere-prim
    what_box.send_keys(job_title)
    where_box.send_keys(location)
    find_jobs_button.click()

    df = pd.DataFrame({'Company Name': [''], 'Location': [''], 'Expected Salary': [''], 'Job Title': [''],
                       'Description About the Job': [''], 'Link': [''], 'Date Posted/Active': ['']})

    while True:
        soup = BeautifulSoup(driver.page_source, 'lxml')
        start = soup.find('ul', class_='jobsearch-ResultsList css-0')

        for litag in start.find_all('li'):
            try:
                title = litag.find('h2', class_='jobTitle css-1h4a4n5 eu4oa1w0').text
                company = litag.find('span', class_='companyName').text
                link = litag.find('a', class_='jcs-JobTitle css-jspxzf eu4oa1w0').get('href')
```

Figure 4.1: code.app.py2

```python
            # Signup successful, redirect to login page
            return redirect(url_for('login'))
        else:
            return render_template('signup.html')

@app.route('/success.html', methods=['GET', 'POST'])
def success():
    if request.method == 'POST':
        username = request.form['username']
        return redirect(url_for('success', username=username))
    else:
        username = request.args.get('username')
        if username:
            return render_template('success.html', username=username)
        else:
            return redirect(url_for('login'))


@app.route('/about_us.html',methods=['GET', 'POST'])
def about_us():
    return render_template('about_us.html')

@app.route('/resume.html')
def resume_form():
    return render_template('resume.html')

@app.route('/resume_builder', methods=['POST'])
def resume_builder():
    # Retrieve form data
    name = request.form['name']
```

Figure 4.2: code.app.py3

```
92      email = request.form['email']
93      phone = request.form['phone']
94      address = request.form['address']
95      education = request.form['education']
96      trainings = request.form['trainings']
97      projects = request.form['projects']
98      skills = request.form['skills']
99      roles = request.form['roles']
100
101     # Generate resume PDF using a suitable library
102     # Example using pdfkit: Install the library using `pip install pdfkit` and configure the PDF rendering
103     pdfkit.from_string(
104         render_template('resume_template.html', name=name, email=email, phone=phone, address=address,
105                         education=education, trainings=trainings, projects=projects, skills=skills, roles=
106         'resume.pdf'
107     )
108
109     # Return the PDF as a downloadable response
110     return send_file('resume.pdf')
111
112 @app.route('/scrape.html', methods=['GET', 'POST'])
113 def scrape():
114     return render_template('scrape.html')
115
116 @app.route('/scrape', methods=['GET', 'POST'])
117 def scraperesults():
118     if request.method == 'POST':
119         job_title = request.form['job_title']
120         location = request.form['location']
121         html_table = scrape_indeed_jobs(job_title, location)
```

Figure 4.3: code.app.py4

```
122         return render_template('result.html', html_table=html_table)
123
124
125 if __name__ == '__main__':
126     app.run(debug=True)
127
```

Figure 4.4: code.app.py5

```python
indeed_jobs_webscrapping.py
import pandas as pd
from selenium import webdriver
from bs4 import BeautifulSoup


def scrape_indeed_jobs(job_title, location):
    # Create Chrome webdriver
    driver = webdriver.Chrome()
    url = "https://in.indeed.com/"
    driver.get(url)

    what_box = driver.find_element(by="xpath", value="//input[@id='text-input-what']")
    where_box = driver.find_element(by="xpath", value="//input[@id='text-input-where']")
    find_jobs_button = driver.find_element(by="xpath", value="//button[@class='yosegi-InlineWhatWhere-prim
    what_box.send_keys(job_title)
    where_box.send_keys(location)
    find_jobs_button.click()

    df = pd.DataFrame({'Company Name': [''], 'Location': [''], 'Expected Salary': [''], 'Job Title': [''],
                       'Description About the Job': [''], 'Link': [''], 'Date Posted/Active': ['']})

    while True:
        soup = BeautifulSoup(driver.page_source, 'lxml')
        start = soup.find('ul', class_='jobsearch-ResultsList css-0')

        for litag in start.find_all('li'):
            try:
                title = litag.find('h2', class_='jobTitle css-1h4a4n5 eu4oa1w0').text
                company = litag.find('span', class_='companyName').text
                link = litag.find('a', class_='jcs-JobTitle css-jspxzf eu4oa1w0').get('href')
```

Figure 4.5: code.indeed$_j obs_w ebscraping.py1$

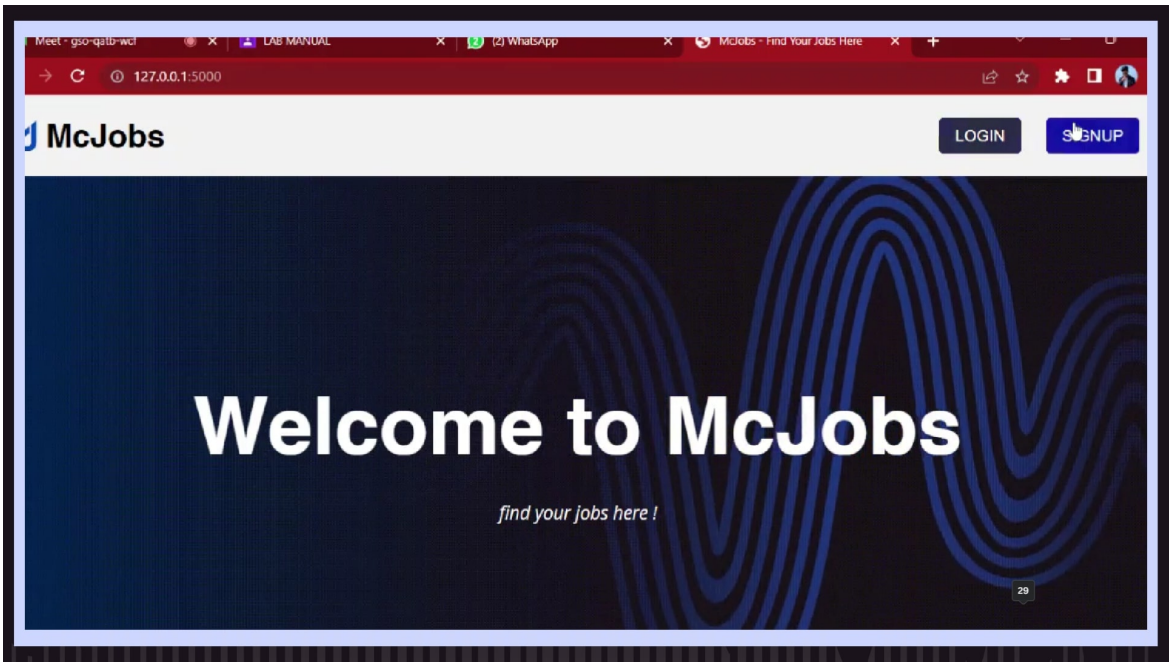
```python
                link_full = 'https://in.indeed.com' + link
                description = litag.find('ul', {
                    'style': 'list-style-type:circle;margin-top: 0px;margin-bottom: 0px;padding-left:20px;

                try:
                    location = litag.find('div', class_='companyLocation').text
                except:
                    location = 'N/A'

                try:
                    salary = litag.find('div', class_='metadata salary-snippet-container').text
                except:
                    salary = 'N/A'

                date = litag.find('span', class_='date').text

                length = len(df)
                df.loc[length] = {'Company Name': company, 'Location': location, 'Expected Salary': salary
                                  'Job Title': title, 'Description About the Job': description, 'Link': li
                                  'Date Posted/Active': date}
            except:
                pass

        try:
            button_to_next_page = soup.find('a', {'aria-label': 'Next Page'}).get('href')
            driver.get('https://in.indeed.com' + button_to_next_page)
        except:
            print("All Data Scraped Successfully.....")
            break
```

Figure 4.6: code.indeed$_j obs_w ebscraping.py2$

```
62      driver.quit()
63      html_table = df.to_html(index=False)
64      return html_table
65
```

Figure 4.7: code.indeed$_j$*obs*$_w$*ebscraping.py*3

4.2 SCREENSHOTS



Introduction Page

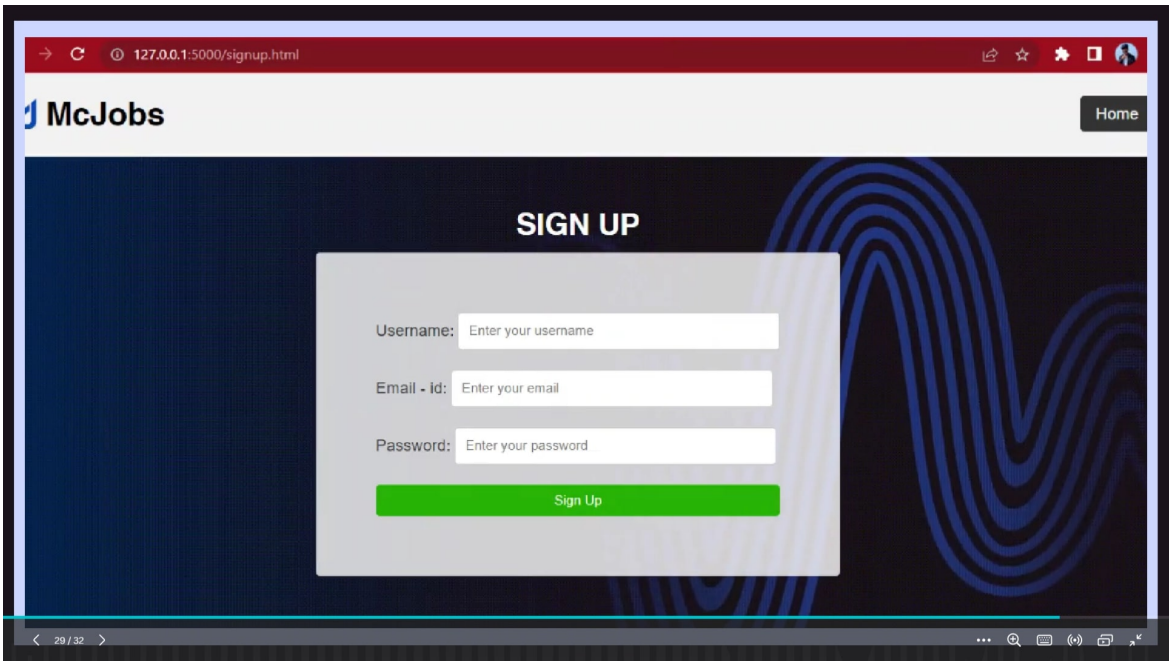


Figure 4.8: Sign Up Page

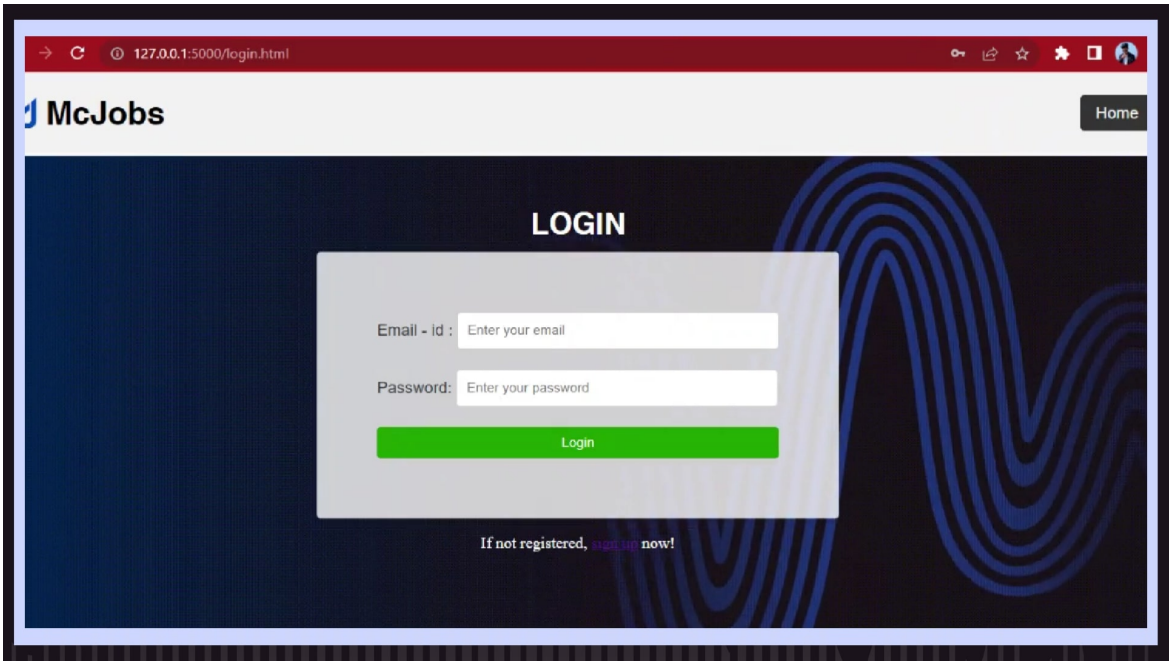


Figure 4.9: Login Page

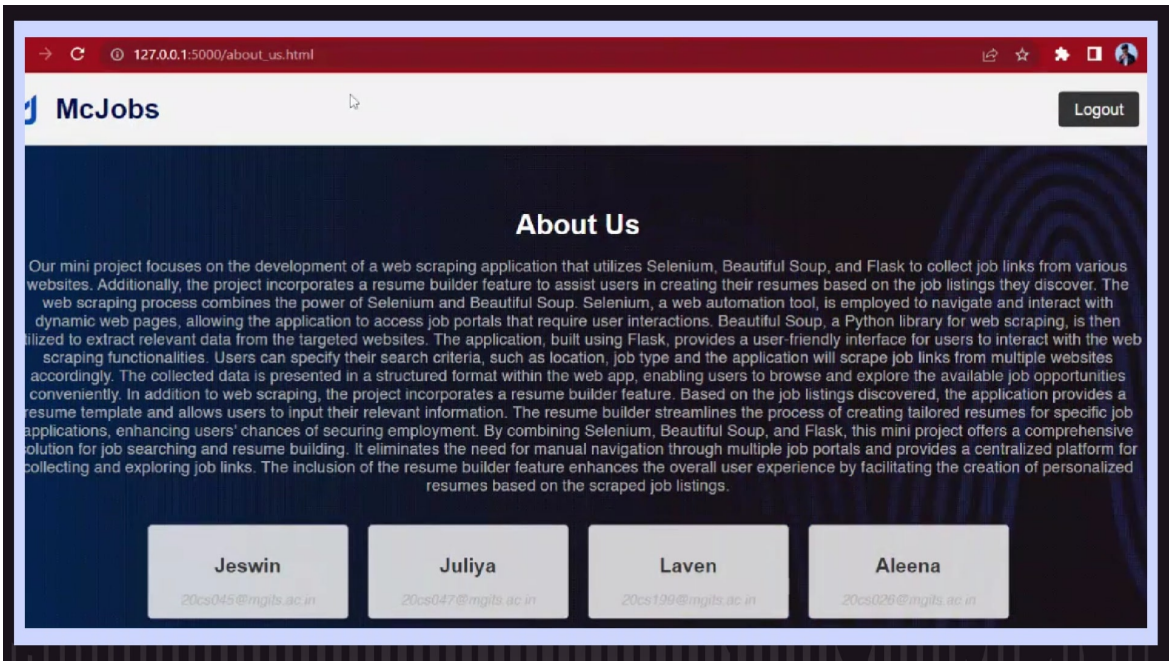


Figure 4.10: About Us

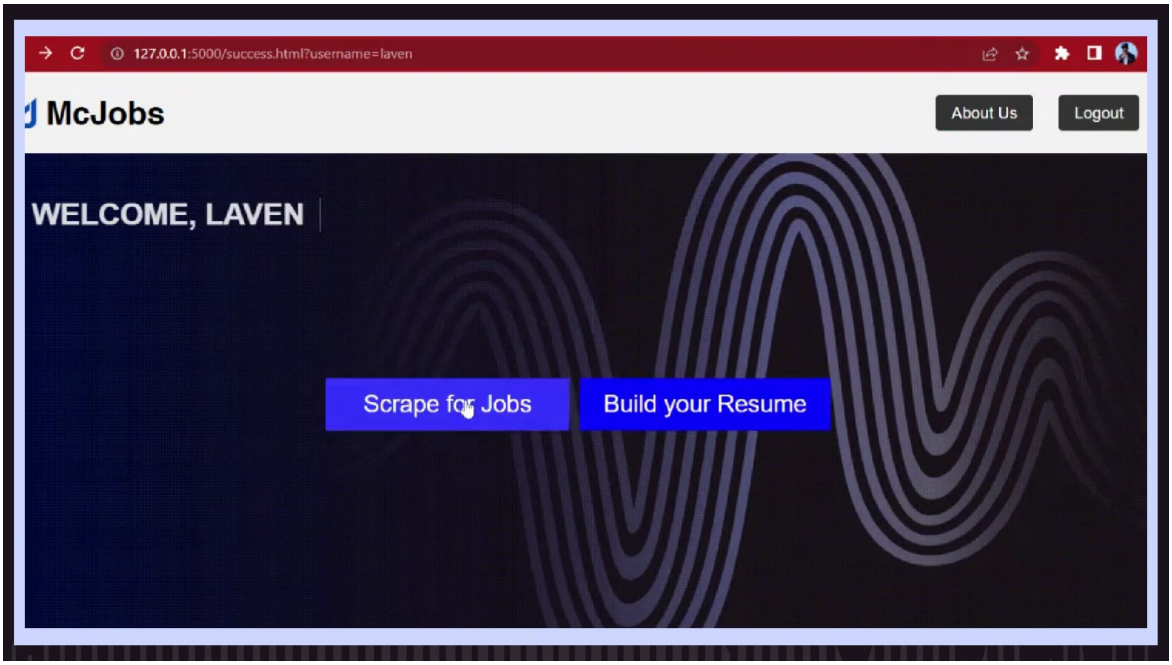


Figure 4.11: Home Page

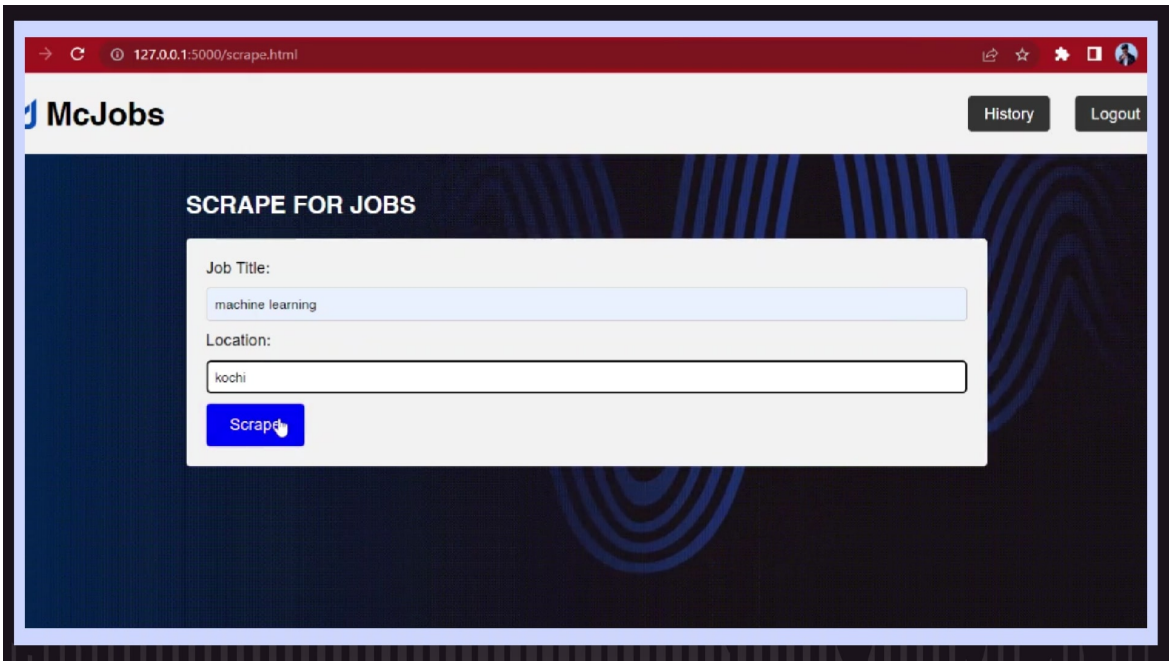


Figure 4.12: Landing Page

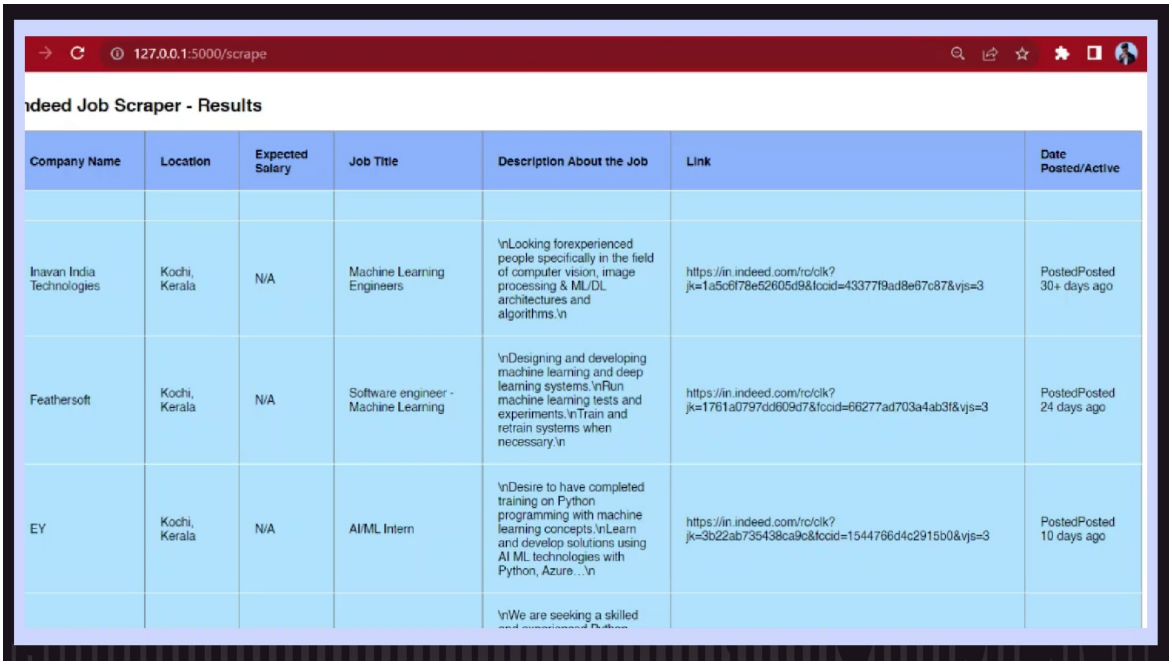


Figure 4.13: Scraped Results

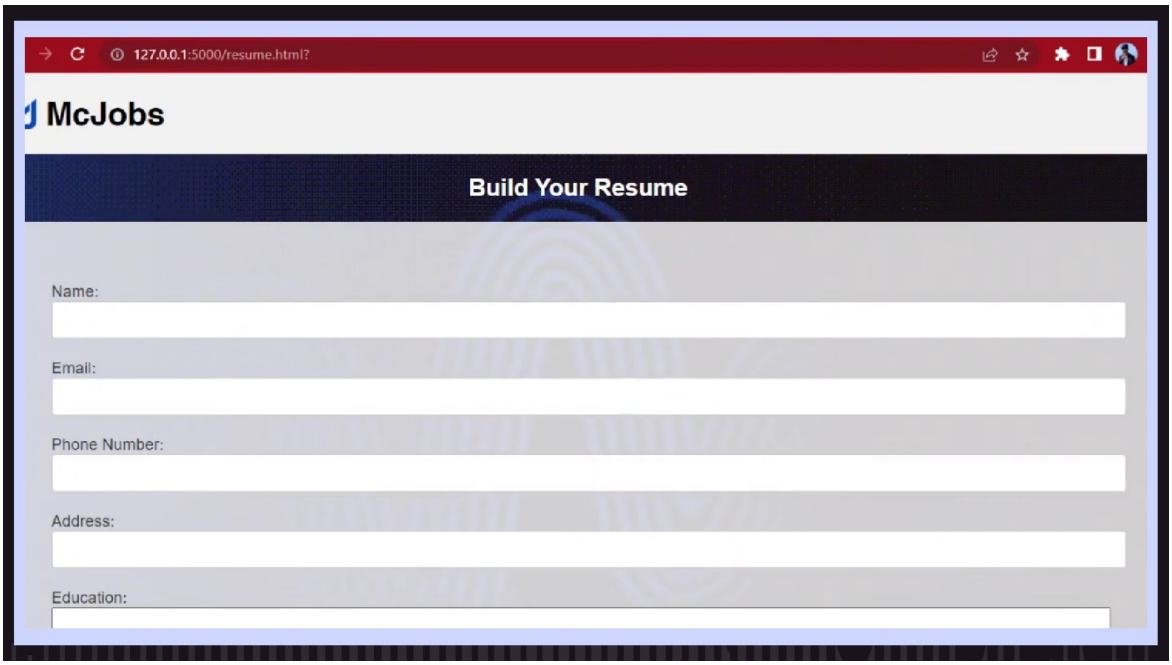


Figure 4.14: Resume Builder Form

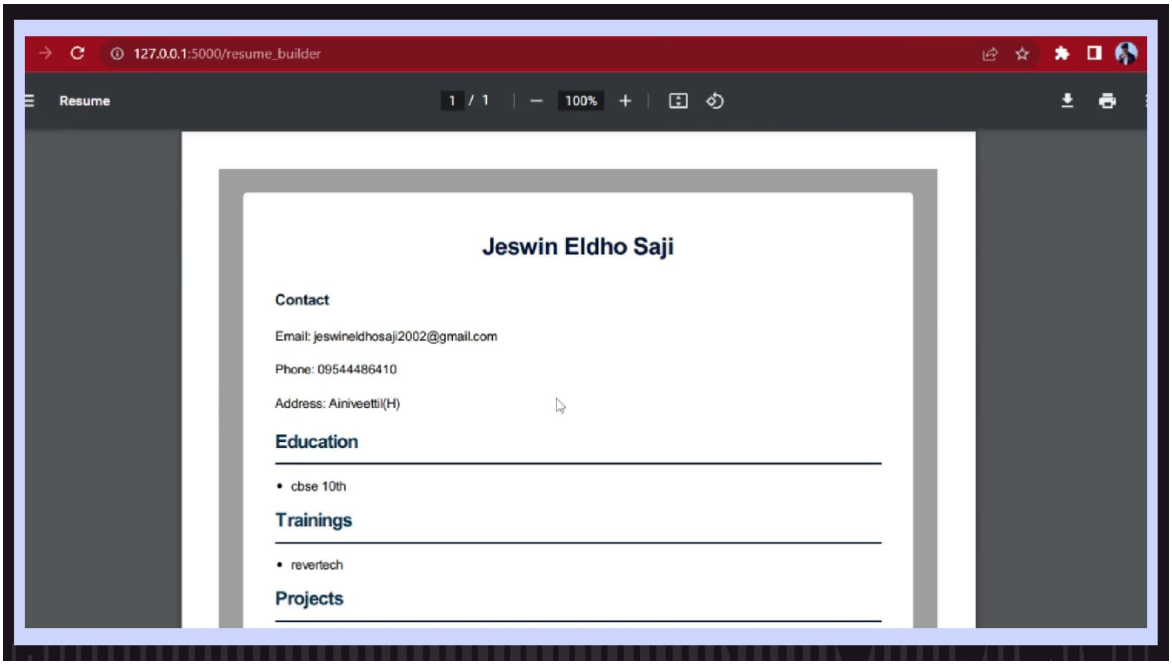


Figure 4.15: Resume

CHAPTER 5

CONCLUSION

In conclusion, web scraping offers several advantages when it comes to extracting data from job portals. By automating the process of gathering job-related information, web scraping allows for efficient analysis of job listings, trends, and market insights. It provides benefits for job seekers, recruiters, and researchers alike.

Web scraping of job portals can help job seekers streamline their job search by consolidating job listings from multiple sources into a single database or spreadsheet. This makes it easier to compare and evaluate opportunities based on various criteria such as job titles, locations, or application deadlines.

For recruiters, web scraping can assist in collecting and organizing candidate data from different job portals, saving time and effort in the hiring process. It enables recruiters to gain a broader view of available talent and make more informed decisions.

Researchers can leverage web scraping to analyze job market trends, identify patterns, and extract valuable data for academic studies or market research. This can lead to insights and recommendations that can contribute to improving recruitment strategies or understanding industry dynamics.

However, it's important to approach web scraping job portals responsibly and ethically. Adhering to the terms of service of the job portals and being mindful of any legal restrictions is crucial. Additionally, web scraping code or tools may require maintenance and updates to adapt to changes in website layouts or anti-scraping measures.

Overall, web scraping of job portals provides a powerful means to gather and analyze job-related data efficiently. It empowers users with valuable insights, enhances the job search process, and supports decision-making for both job seekers and recruiters. By understanding and respecting the ethical and legal considerations, web scraping can be a valuable tool in the realm of job portal data extraction.

5.1 REFERENCES

1. Alavi, M., Leider, D. (1999). Knowledge management systems: Emerging views and practices from the field. In System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on (pp. 8-pp). IEEE.

2. Yang, Zhilin, Shaohan Cai, Zheng Zhou, and Nan Zhou. "Development and validation of an instrument to measure user perceived service quality of information presenting web portals." Information Management 42.

3. Rafter, R., Bradley, K., Smyth, B. (2000). Personalized retrieval for online recruitment Services, In Proceedings of the 22nd Annual Colloquium on Information Retrieval. 4. Vivek Kumar Sehgal1, "Job Portal-A Web Application for Geographically Distributed Multiple Clients", 1 Department of CSE and ICT Jaypee University of Information Technology, Waknaghat, Solan, H.P (INDIA)M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

4. Bizer, C., Heese, R., Mochol, M., Oldakowski, R., Tolksdorf, R. Eckstein, R. (2005). The impact of semantic web technologies on job recruitment processes.

5. https://ieeexplore.ieee.org/document/6959916

6. https://www.ijcaonline.org/archives/volume180/number36/tiwari-2018-ijca-916895.pdf.

7. https://www.signitysolutions.com/blog/importance-creating-job-portals/

8. https://github.com/Ashishkapil/Web-scraping-job-portal-sites

9. https://research.aimultiple.com/web-scraping-recruitment/

10. https://www.signitysolutions.com/blog/importance-creating-job-portals/