Royal University of Phnom Penh
Faculty of Engineering
Data Science and Engineering Program

SmartCareerBot

AI-Powered Job Landscape Chatbot for Cambodia

Subject: Web Mining
Lecturer: Chim Bunchun

Group 01
Group Members: Vay Mithona, Phon Lihour,
Rop Borom, Hak Vitou, Cheklim Sunimithm

Table of Content

Problem Statement

- Cambodian job seekers often check multiple sites manually
- There’s no unified platform to explore and compare job descriptions
- Skill trends, demand, ands salary comparisons are heard to access.

Project Goal and Key Objectives

Goal: Build a chatbot that provides real-time, structured job information for job seekers in Cambodia.

Objectives:

- Scrape jobs from local sites (Jobify, CamHR, Workingna).
- Clean and structure the data (JSON/Markdown).
- Allow AI chatbot to answer natural language job queries.

Target Job Websites

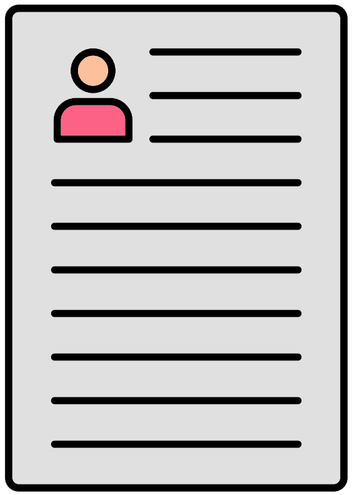
Jobify.works

CamHR.com

Workingna.com

Extracted Fields

Job title	Company name	Location
Salary	Experience	Qualification
Responsibilities	Rquirements	Education
Post	Application deadline	

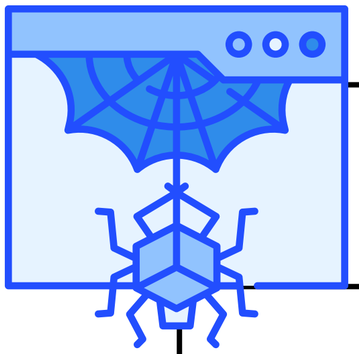


AI Chatbot Integration and Use Cases

Chatbot Workflow

- User asks: “What are the top-paying IT jobs in Phnom Penh?
- Bot queries sturctured data, filters by city + salary
- Returns Markdown-styled job list with links

Web Scraping Pipeline Overview



Define job ID or job page URL

Load content using Selenium

Parse HTML using BeautifulSoup

Extract key fields using regex/XPath

Save output as CSV or JSON

Tools Used

JSON

.CSV

BeautifulSoup

WebDriverWait

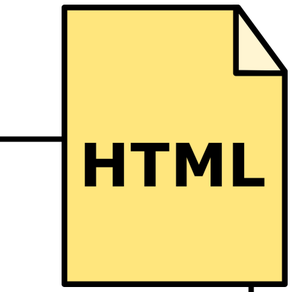
Selenium Python

#1 Jobify Scraper

Libraries Used

- Selenium: for dynamic content loading
- csv: for saving extracted rows
- time, os, WebDriverWait: for delays, retires, and stability

Key HTML Elements



- **Job Title:** `<h2 class="job-title">`
- **Info Fields: Detected via XPath:**
  - `//strong[text()='Salary:']/following-sibling::text()`
- **Requirements:** `<h5>Job Requirement</h5>` → `<ul><li>`

Scraping Workflow

Loop through job IDs (1086 to 500)
Use **Selenium** (headless Chrome) to load page
Wait for elements with **WebDriverWait**
Extract fields via **XPath**
Store in **job4.csv**

#1 Jobify Scraper: Code Snippet

Initialize Browser

```
from selenium import webdriver
from selenium.webdriver.chrome.service import
Service

service = Service("path/to/chromedriver.exe")
options = webdriver.ChromeOptions()
options.add_argument("--headless")  # Run
without opening a browser window

driver = webdriver.Chrome(service=service,
options=options)
```

Label-Based Extraction

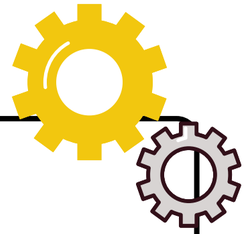
```
def get_job_detail(label):
    try:
        element = driver.find_element(By.XPATH,
f"//strong[text()='{label}']")
        return element.find_element(By.XPATH,
"./following-sibling::text()").strip()
    except:
        return "N/A"
```

Extract Requirements

```
job_req_section = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.XPATH, "//h5[text()='Job Requirement']/following-
sibling::div"))
)
ul_elements = job_req_section.find_elements(By.TAG_NAME, "ul")
li_elements = [li.text for ul in ul_elements for li in ul.find_elements(By.TAG_NAME, "li")]
```

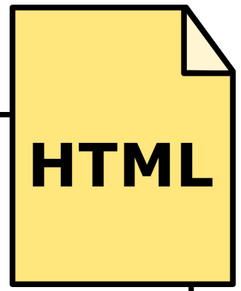
#2 CamHR Scraper

Libraries Used



- **Selenium**: browser automation
- **BeatifulSoup**: HTML parsing
- **csv, os, time:** save, manage, and retry scraping

Key HTML Elements



- **Job Title:** `<h5 class="css-97a38i">`
- **Company:** `<div class="css-aabkpg">`
- **Salary:** `<span class="css-10bh2m3">`
- **Job Requirements:** `<div class="ql-editor">` and/or `<ul><li>`

Scraping Workflow

Loop through job IDs

Wait for *.job-header-content* to load using **Selenium**

Parse HTML with **BeautifulSoup**

Extract table + description fields

Output to sturctured CSV (**camhr_data.csv**)

#2 CamHR Scraper: Code Snippet

Extract Metadata

```
<span class="job-name-span">Job Title</span>
<p class="mb-1 company-headbox"><a>Company
Name</a></p>
<table class="mailTable">...Job Details...
</table>
<div class="job-descript">...<span>Job
Requirements</span>...</div>
<div class="send-date"><span>Publish Date</span>
<span>Closing Date</span></div>
```

Header Info Extractor

```
job_title = soup.find("span", class_="job-name-
span").text.strip()
company = soup.find("p", class_="company-
headbox").find("a").text.strip()
table = soup.find("table", class_="mailTable")
job_req = soup.find("div", class_="job-
descript").get_text()
```

Load Job Page

```
for job_id in range(start_id, end_id + 1):
    url = base_url.format(job_id)
    driver.get(url)
    WebDriverWait(driver, 5).until(EC.presence_of_element_located((By.CLASS_NAME, "job-header-
content")))
```

#3 WorkingnaScraper

Libraries Used

- **Selenium**: control browser behavior
- **BeatifulSoup**: navigate and parse HTML
- **csv, urlib, os, time**: save, manage, and retry scraping

Key HTML Elements

- **Job Title:** `<span class="job-name-span">`
- **Company:** `<p class="company-headbox">`
- **Job Info Table:** `<table class="mailTable">`
- **Requirements:** `<div class="job-descript">`
- **Dates:** `<div class="send-date">`

HTML

Scraping Workflow

Loop through jobs IDS from 10755 to 11683
Use try logic (max 2 attempts)
Detect and skip invalid or placeholder pages
Clean and format fields with bullet points
Save results to **New_Data_workingna.csv**

#3 Workingna Scraper: Code Snippet

Content Structure

```
<h5 class="css-97a38i">Job Title</h5>
<div class="css-aabkpg"><h6>Company Name</h6>
</div>
<span class="css-10bh2m3">Salary</span>
<div class="ql-editor"><p>Job
Responsibilities</p></div>
```

Text Extractor

```
def extract_ql_editor_content(soup,
heading_text):
    heading = soup.find(lambda tag:
heading_text.lower() in tag.get_text().lower())
    ql_editor = heading.find_next("div",
class_="ql-editor")
    paragraphs = ql_editor.find_all("p")
    return " • ".join(p.get_text().strip() for p
in paragraphs)
```

Page Iterator

```
for job_id in range(10755, 11683 + 1):
    url = f"https://workingna.com/job/{job_id}"
    driver.get(url)
    soup = BeautifulSoup(driver.page_source, "html.parser")
    job_title = soup.find("h5", class_="css-97a38i").text.strip()
```

Converting Scraped Job Data to Markdown Format

Why Markdown?

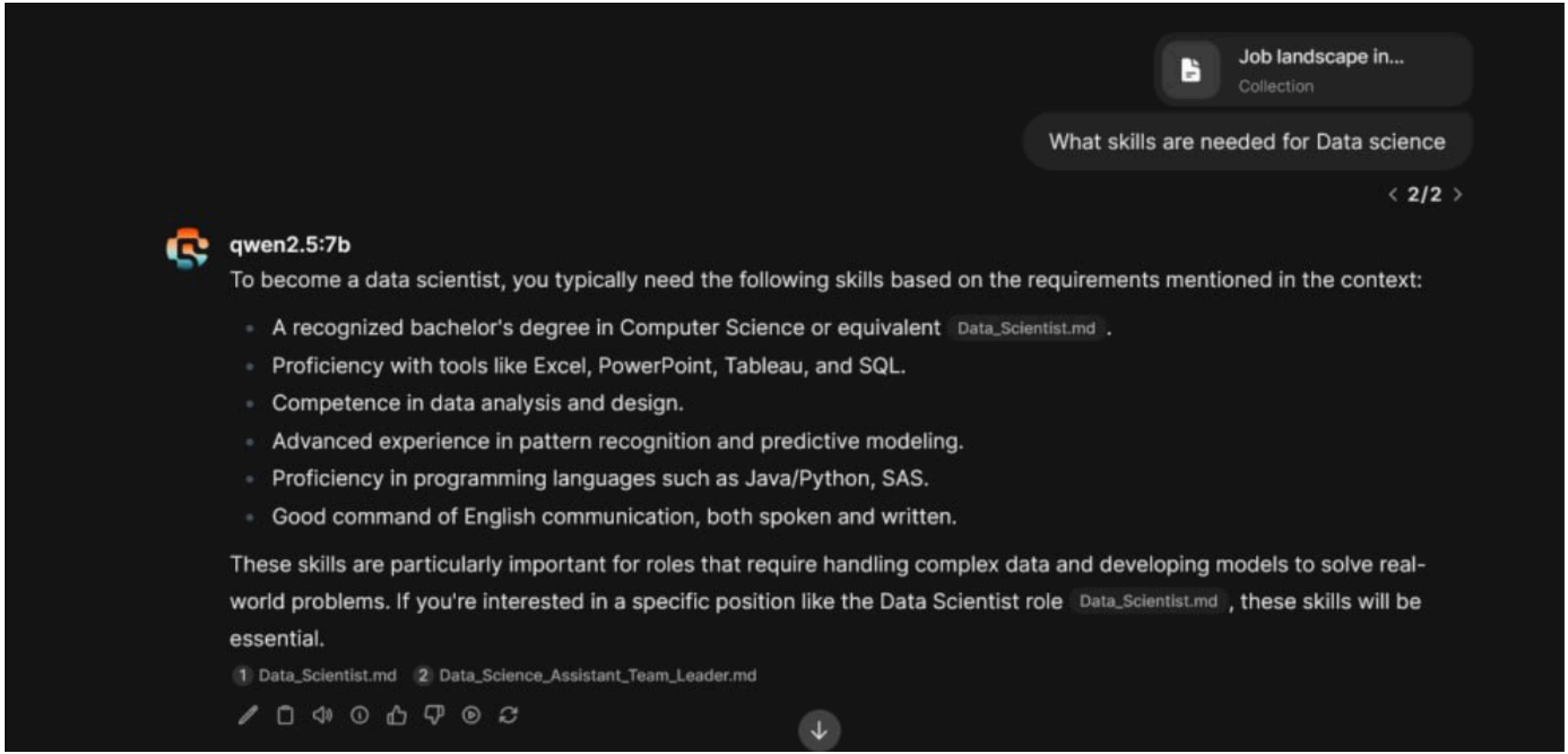
- Clean, readable formatting for chatbots
- Compatible with Open WebUI, Telegram, and web apps
- Supports basic rich text (bold, bullets, line breaks)
- Lightweight and fast to render

Integration with Open WebUI (RAG-Ready)

What is Open WebUI?

- A local web-based chatbot interface for models like Ollama
- Supports natural language queries against preloaded Markdown or CSV
- Uses RAG (Retrieval-Augmented Generation) to find relevant job info
- Privacy-focused and runs offline

Integration with Open WebUI (RAG-Ready)



Integration with Open WebUI (RAG-Ready)

How Integration Works:

- Scraped jobs are converted into Markdown files
- Files are embedded into a vector store (e.g., FAISS or ChromaDB)
- User asks: “What IT jobs in Phnom Penh pay over $700?”
- Open WebUI retrieves the matching job entries and answers naturally

Thank you!