DOCUMENTATION FOR WEBSITE INFORMATION EXTRACTION SOLUTION

BURRA AJAY KUMAR

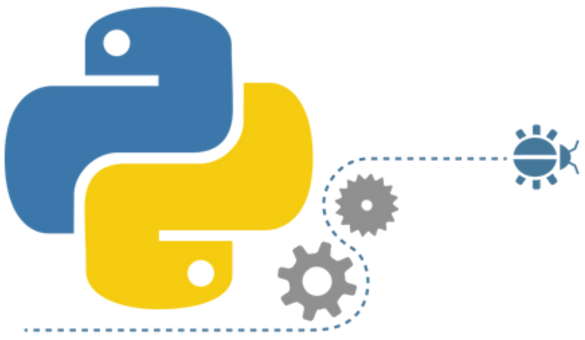
Table of CONTENTS

Introduction

This documentation provides a comprehensive guide to understanding and setting up the solution for extracting essential information from websites using Python. The solution uses Selenium, BeautifulSoup, and MySQL to extract and store data including meta information, social media links, technology stacks, payment gateways, website language, and category.

SETUP INSTRUCTIONS

Python Environment:



- Ensure Python 3.x is installed on your system.

Dependencies:

```
pip install requests beautifulsoup4 mysql-connector-python selenium
```

```
!pip install webdriver_manager
```

```
pip install langcodes
```

- Install required Python packages using ‘pip’.

Import Statements:

```python
import requests
from bs4 import BeautifulSoup
from selenium import webdriver
import mysql.connector
import re
import langcodes
```

- The import statements are used to include libraries for HTTP requests (requests), HTML parsing (BeautifulSoup), browser automation (Selenium), database connectivity (mysql.connector), regular expressions (re), and language code handling (langcodes).

Initialize WebDriver:

```python
options = webdriver.ChromeOptions()
options.add_argument('--headless')
options.add_argument('--no-sandbox')
options.add_argument('--disable-dev-shm-usage')
options.add_argument('user-agent=Mozilla/5.0 \
                     (Windows NT 10.0; Win64...))

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

- `--headless`: Runs Chrome in headless mode, without a GUI.
- `--no-sandbox`: Disables the sandbox security feature.
- `--disable-dev-shm-usage`: Prevents issues related to limited `/dev/shm` space.
- `user-agent`: Sets a user-agent string to mimic a regular browser.

Usage: Initializes the Chrome WebDriver with specified options for automated web page interaction.

Load URL:

```
if not url.startswith('http://') and not url.startswith('https://'):
    url = f'https://{url}'

try:
    driver.get(url)
except Exception as e:
    print(f"Error loading URL {url}: {e}")
    driver.quit()
    return None
```

- Prepend `https:// ` if the URL doesn't start with `http://` or `https://`.
- Load the URL using the WebDriver and handle any exceptions.

Parse HTML with BeautifulSoup:

```
soup = BeautifulSoup(driver.page_source, 'html.parser')
driver.quit()
```

- Parse the page source using BeautifulSoup.

Extract Meta Title and Description:

```
meta_title = soup.find('title').text if soup.find('title') else None
meta_description = soup.find('meta', attrs={'name': 'description'})['content']
                   if soup.find('meta', attrs={'name': 'description'}) else None
```

- Extract the meta title and description from the HTML.

Extract Social Media Links:

```
social_media_links = []
index = 1
added_platforms = set()
platforms = ['facebook', 'twitter',...]

for link in soup.find_all('a', href=True):
    link_url = link['href'].lower()
    for platform in platforms:
        if platform in link_url and platform not in added_platforms:
            social_media_links.append(f"{index}. {link['href']}")
            index += 1
            added_platforms.add(platform)

if not social_media_links:
    social_media_links.append("Not Specified")
```

- Iterates through all anchor tags with `href` attributes in the HTML.
- Checks if the `href` contains any of the social media platform keywords and ensure no duplicates.
- Adds found social media links to the list, otherwise notes "Not Specified" if none are found.

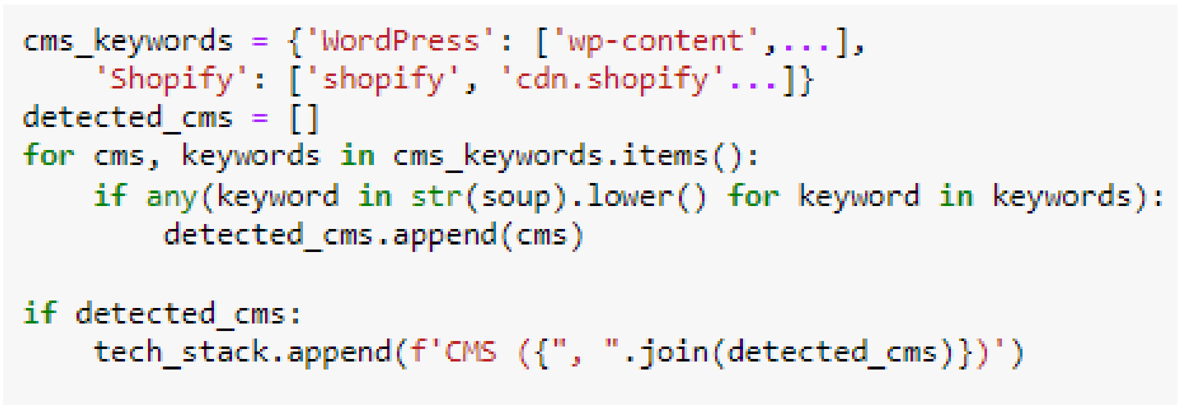
Extract Technology Stack:

1. MVC (Model-View-Controller) Frameworks:

```
tech_stack = []
mvc_frameworks = ['angular', 'react',...]
detected_mvc_frameworks = []
for framework in mvc_frameworks:
    if soup.find_all('script', src=lambda src: src and framework in src.lower()):
        detected_mvc_frameworks.append(framework.capitalize())
if detected_mvc_frameworks:
    tech_stack.append(f"MVC Framework ({', '.join(detected_mvc_frameworks)})")
```

- Defines a list of common MVC frameworks.
- Searches for scripts in the HTML that reference these frameworks.
- Adds detected frameworks to the technology stack.

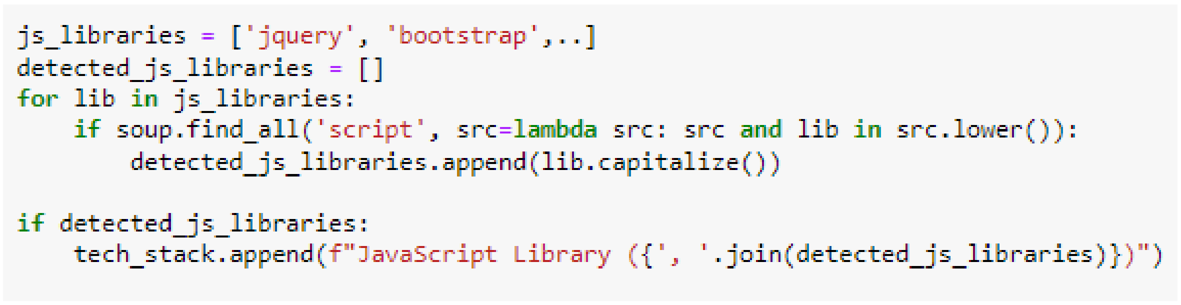
2. CMS (Content Management System) Platforms:



```python
cms_keywords = {'WordPress': ['wp-content',...],
    'Shopify': ['shopify', 'cdn.shopify'...]}
detected_cms = []
for cms, keywords in cms_keywords.items():
    if any(keyword in str(soup).lower() for keyword in keywords):
        detected_cms.append(cms)

if detected_cms:
    tech_stack.append(f'CMS ({", ".join(detected_cms)})')
```

- Defines a dictionary of CMS platforms and their identifying keywords.
- Checks the HTML content for these keywords.
- Adds detected CMS platforms to the technology stack.

3. JavaScript Libraries:



```python
js_libraries = ['jquery', 'bootstrap',..]
detected_js_libraries = []
for lib in js_libraries:
    if soup.find_all('script', src=lambda src: src and lib in src.lower()):
        detected_js_libraries.append(lib.capitalize())

if detected_js_libraries:
    tech_stack.append(f"JavaScript Library ({', '.join(detected_js_libraries)})")
```

- Defines a list of common JavaScript libraries.
- Searches for scripts in the HTML that reference these libraries.
- Adds detected libraries to the technology stack.

4. CSS (Cascading Style Sheets) Frameworks:

```
css_frameworks = ['bootstrap', 'foundation',...]
detected_css_frameworks = []
for framework in css_frameworks:
    if soup.find_all('link', rel='stylesheet',
                     href=lambda href: href and framework in href.lower()):
        detected_css_frameworks.append(framework.capitalize())

if detected_css_frameworks:
    tech_stack.append(f"CSS Framework ({', '.join(detected_css_frameworks)})")
```

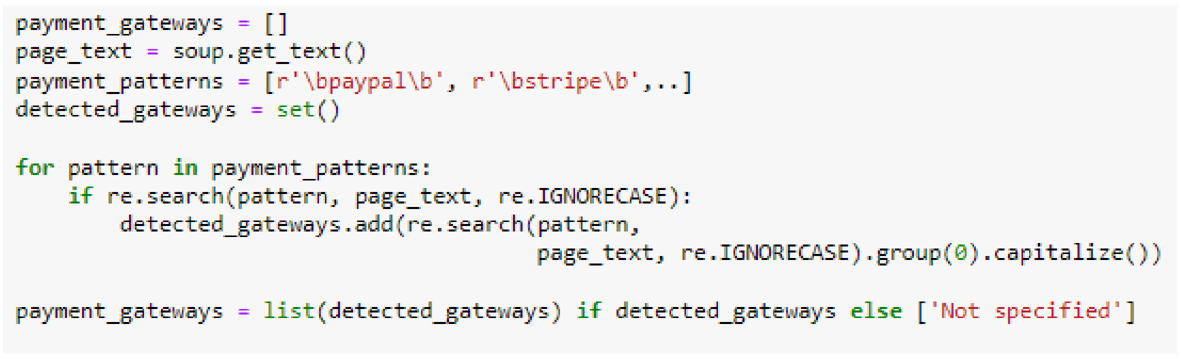
- Defines a list of common CSS frameworks.
- Searches for stylesheet links in the HTML that reference these frameworks.
- Adds detected frameworks to the technology stack.

5. Backend Technologies:

```
backend_tech = ['django', 'flask',...]
detected_backend_tech = []
for tech in backend_tech:
    if soup.find_all('script', src=lambda src: src and tech in src.lower()):
        detected_backend_tech.append(tech.capitalize())

if detected_backend_tech:
    tech_stack.append(f"Backend Technology ({', '.join(detected_backend_tech)})")
```

- Defines a list of backend frameworks and languages.
- Searches for scripts in the HTML that reference these technologies.
- Adds detected backend technologies to the technology stack.

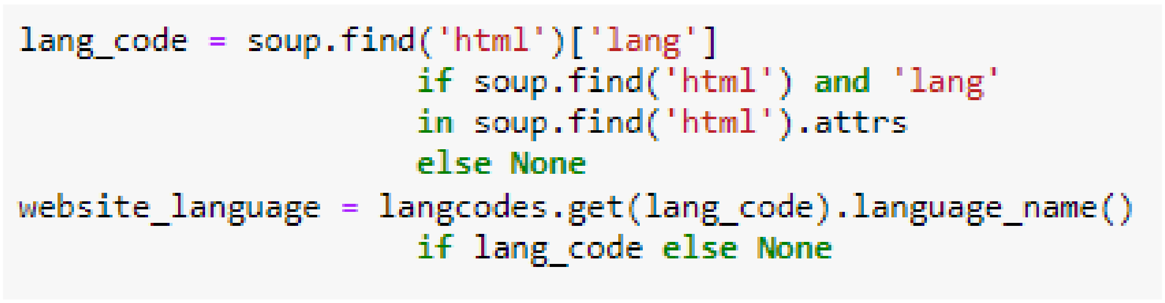
Extract Payment Gateways:



```python
payment_gateways = []
page_text = soup.get_text()
payment_patterns = [r'\bpaypal\b', r'\bstripe\b',..]
detected_gateways = set()

for pattern in payment_patterns:
    if re.search(pattern, page_text, re.IGNORECASE):
        detected_gateways.add(re.search(pattern,
                                        page_text, re.IGNORECASE).group(0).capitalize())

payment_gateways = list(detected_gateways) if detected_gateways else ['Not specified']
```

- Searches the text content using the defined payment patterns.
- Adds detected payment gateways to a list if matches are found.
- Defaults to "Not specified" if no payment gateways are identified based on the patterns searched.

Extract Website Language:



```python
lang_code = soup.find('html')['lang']
                if soup.find('html') and 'lang'
                in soup.find('html').attrs
                else None
website_language = langcodes.get(lang_code).language_name()
                if lang_code else None
```

- Checks for the presence of a `lang` attribute within the `<html>` tag.
- Gets the language code (lang_code) from the attribute if present.
- Uses the `langcodes` library to convert the language code (`lang_code`) into the corresponding language name (`website_language`).
- Returns the identified website language. If no language code is found, it returns `None`.

Determine Website Category:

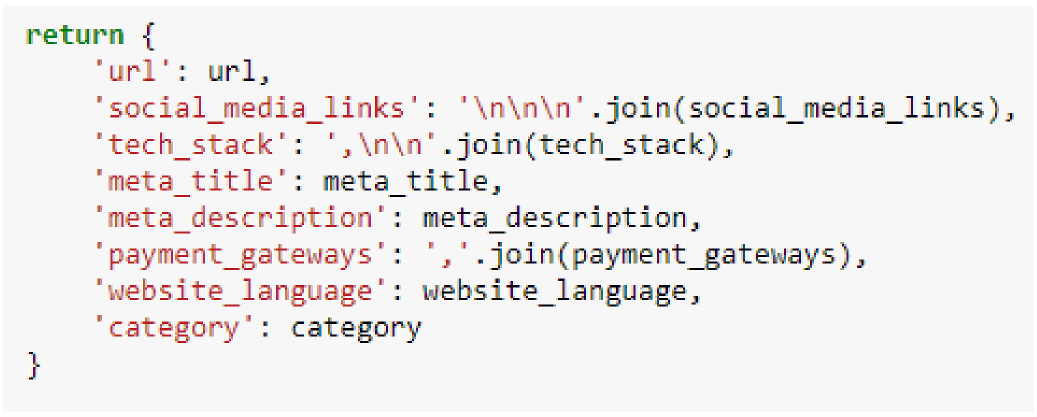
```
categories = {'E-commerce': ['shop', 'cart',...],
    'Blog': ['blog', 'post',..']}

text_content = ' '.join([tag.get_text().lower()
                        for tag in soup.find_all(
                            ['title', 'meta', 'h1', 'h2', 'h3', 'p'])])

category_scores = {category: 0 for category in categories}

for category, keywords in categories.items():
    for keyword in keywords:
        category_scores[category] += text_content.count(keyword)

primary_category = max(category_scores, key=category_scores.get)

if category_scores[primary_category] == 0:
    category = 'General'
else:
    category = primary_category
```

Let's Play How Scores categories Work in the gathered text content

EDUCATION	course	~~lesson~~	~~school~~	~~college~~	lecture	**2**
TECHNOLOGY	software	programming	~~developer~~	IT	tech	**4** WON
Finance	~~bank~~	~~loan~~	money	~~stock~~	~~insurance~~	**1**

- Scores categories: It iterates through each category and its associated keywords, counting how many times each keyword appears in the gathered text content. This count is added to the corresponding category's score in `category_scores`.
- Determines primary category: It identifies the category with the highest score (`primary_category`). If no category scores above 0, it defaults to a "General" category.
- Returns category: Finally, it returns the primary category determined based on the highest score.

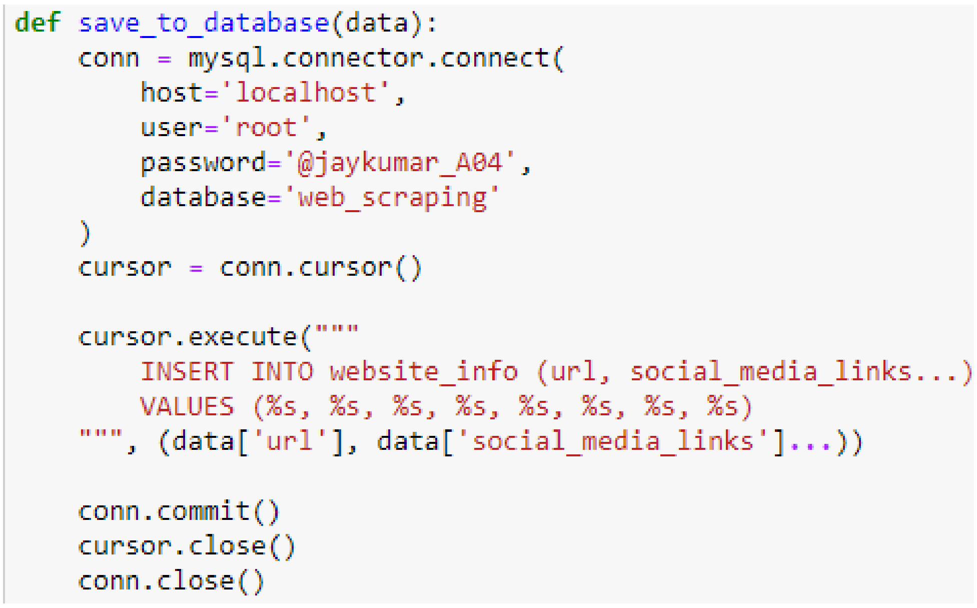
Return Extracted Data:



```
return {
    'url': url,
    'social_media_links': '\n\n\n'.join(social_media_links),
    'tech_stack': ',\n\n'.join(tech_stack),
    'meta_title': meta_title,
    'meta_description': meta_description,
    'payment_gateways': ','.join(payment_gateways),
    'website_language': website_language,
    'category': category
}
```

- The script extracts various data from a website, including social media links, technology stack, meta title, meta description, payment gateways, website language, and website category, and returns them as a structured data object.

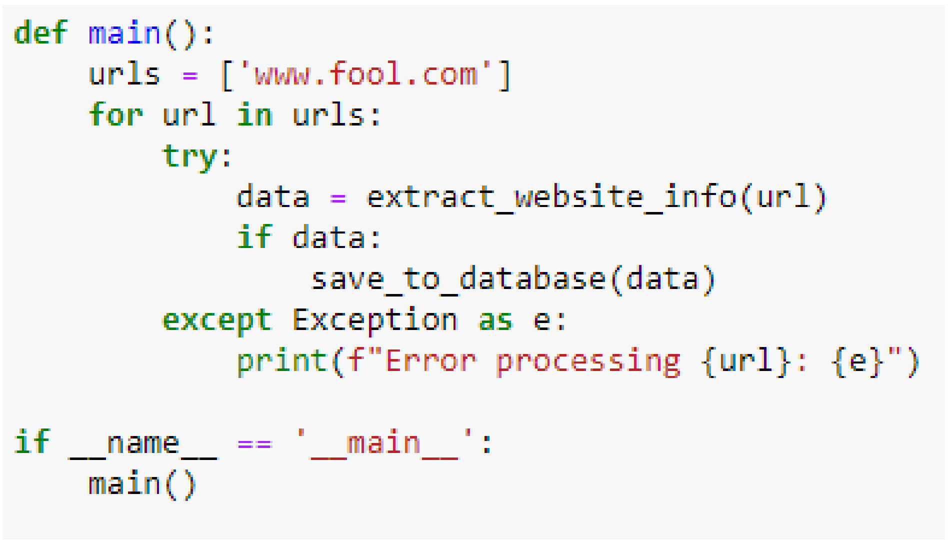
Save to MySQL database Function:



```
def save_to_database(data):
    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='@jaykumar_A04',
        database='web_scraping'
    )
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO website_info (url, social_media_links...)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """, (data['url'], data['social_media_links']...))

    conn.commit()
    cursor.close()
    conn.close()
```

- Connect to the MySQL database.
- Insert the extracted data into the website_info table.
- Close the database connection.

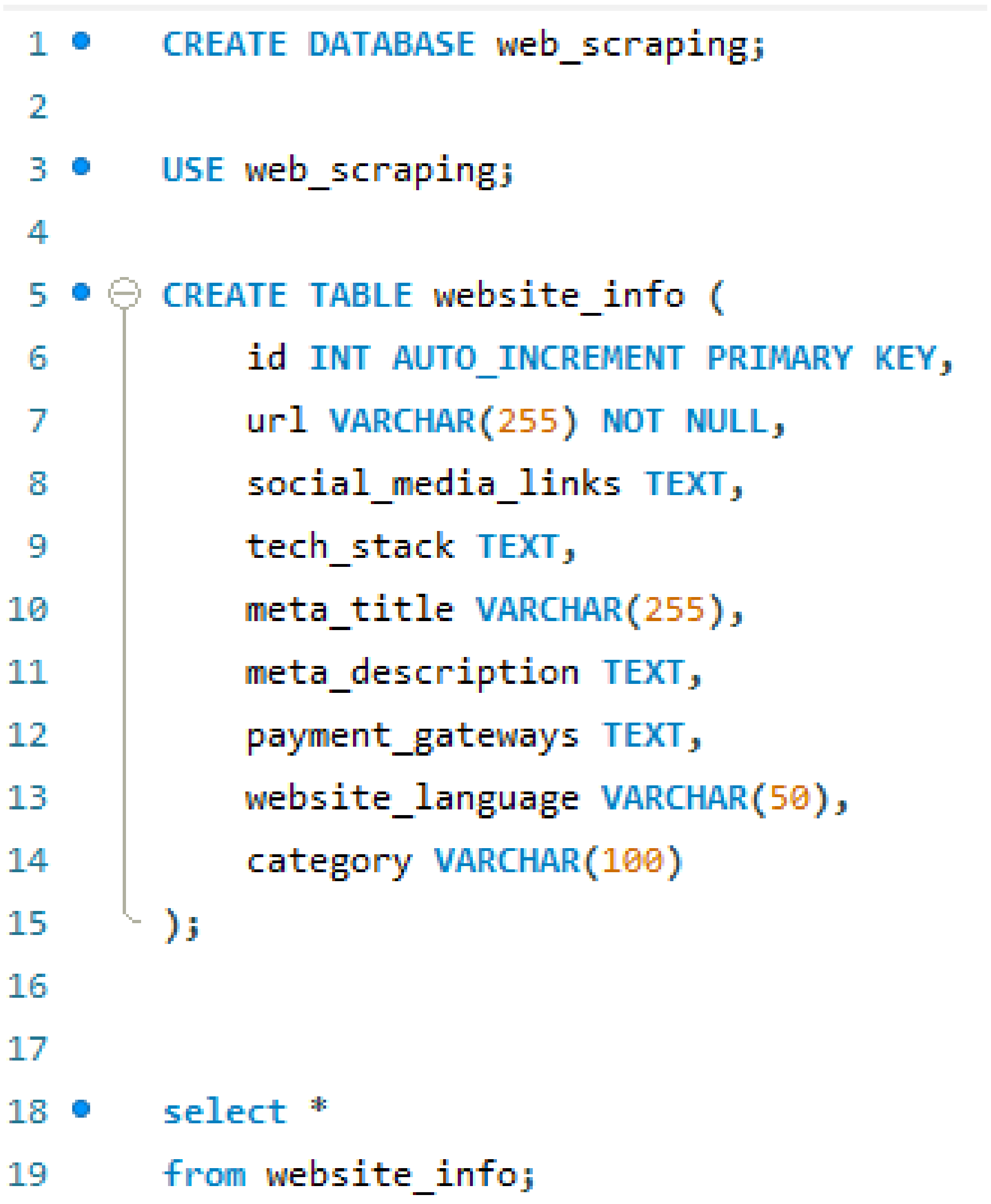
Main Function:



```python
def main():
    urls = ['www.fool.com']
    for url in urls:
        try:
            data = extract_website_info(url)
            if data:
                save_to_database(data)
        except Exception as e:
            print(f"Error processing {url}: {e}")

if __name__ == '__main__':
    main()
```

- Define a list of URLs to process.
- Extract information for each URL and save it to the database.
- Handle any exceptions during processing.

CREATE A MySQL DATABASE



```sql
CREATE DATABASE web_scraping;

USE web_scraping;

CREATE TABLE website_info (
    id INT AUTO_INCREMENT PRIMARY KEY,
    url VARCHAR(255) NOT NULL,
    social_media_links TEXT,
    tech_stack TEXT,
    meta_title VARCHAR(255),
    meta_description TEXT,
    payment_gateways TEXT,
    website_language VARCHAR(50),
    category VARCHAR(100)
);


select *
from website_info;
```

- The script creates a MySQL database named web_scraping, selects it, and creates a table website_info to store website details including URL, social media links, technology stack, meta title, meta description, payment gateways, website language, and category, using an auto-incrementing id as the primary key.