AL MAAREF UNIVERSITY

Faculty of Sciences – Computer Sciences

CSC497 – Practical Training

Bootcamp Data Science (Python).

MOHAMAD SAYED ALI – ID: 10 11 92 95

October, 2024

Email 10119295@mu.edu.lb

sayedalimohamad@gmail.com

GitHub Account sayedalimohamad

Repositories

- https://github.com/sayedalimohamad/task01.git
- https://github.com/sayedalimohamad/task02.git
- https://github.com/sayedalimohamad/task03.git
- https://github.com/sayedalimohamad/task04.git

YouTube live preview

- https://www.youtube.com/watch?v=OcNPVfINXtU

Contents

Introduction

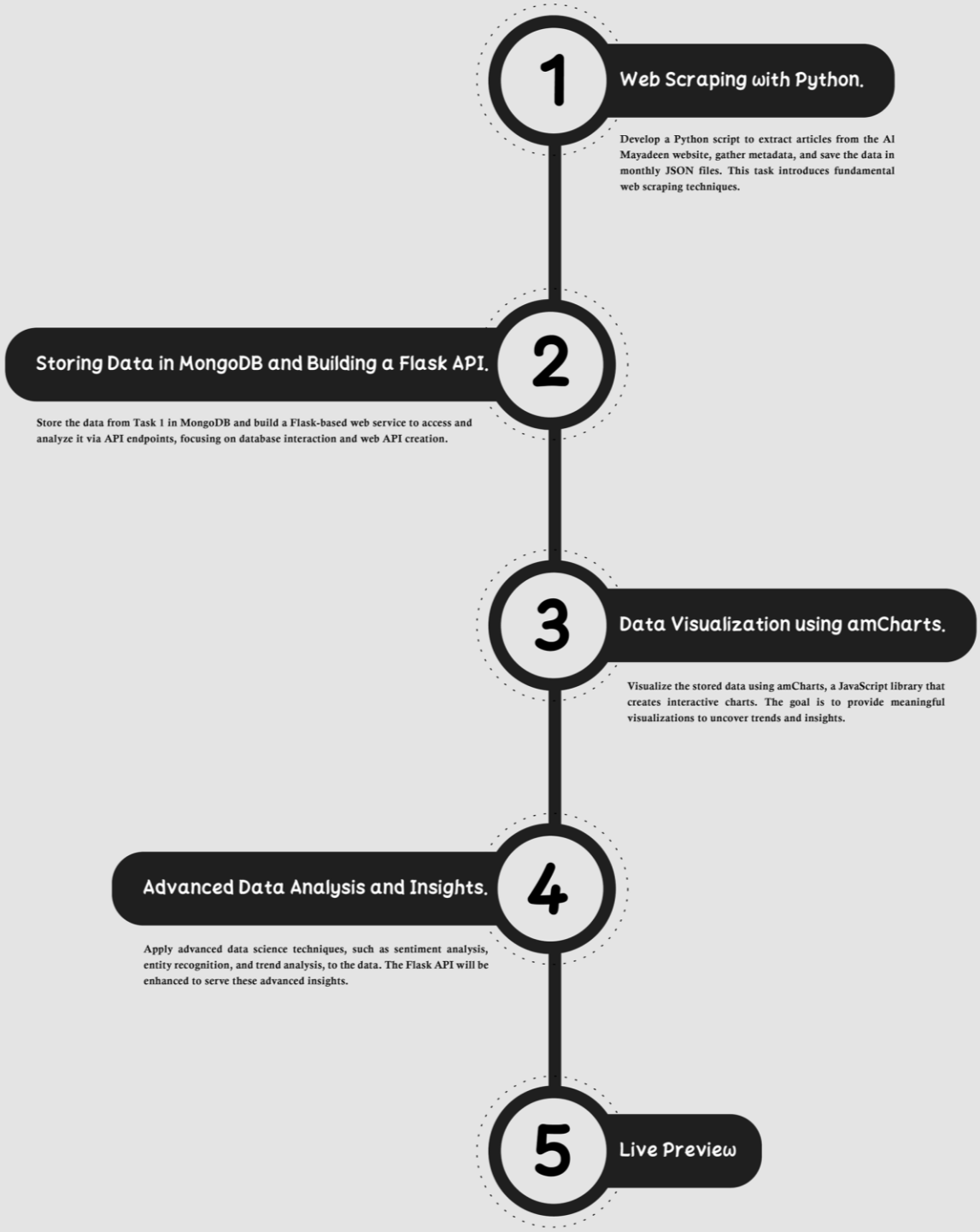
The summer training of 2024 was an enriching experience that allowed me to dive deep into the world of web development and data analysis. Over four tasks, I engaged in a structured curriculum that included web scraping, database management, data visualization, and advanced data analysis.

In Task 1, I learned to scrape articles from the Al Mayadeen website using Python, gaining hands-on experience with data extraction and organization. Task 2 focused on storing this data in MongoDB and developing a Flask API, which provided a foundational understanding of databases and web services. Task 3 introduced me to data visualization with amCharts, enabling me to create interactive charts that brought the data to life. Finally, Task 4 was dedicated to advanced data analysis techniques, including sentiment analysis and entity recognition, allowing me to extract meaningful insights from the collected data.

Additionally, I explored dashboard UI/UX design and implemented user authentication, further enhancing my understanding of web applications. Throughout this project, I utilized various programming languages and frameworks, including **Python, Flask, HTML, CSS, Bootstrap, JavaScript, and amCharts**, alongside libraries and tools such as *os, json, requests, logging, re, time, dataclasses,* and *BeautifulSoup* to support my work with Flask and MongoDB.

This report details my journey through each task, the skills I acquired, and the insights gained throughout this transformative summer training experience.

Steps Of Project – All Tasks



Check out this link for an exclusive live preview on YouTube!

https://www.youtube.com/watch?v=OcNPVfINXtU

TASK 01

Task 01 – Web Scraping with Python.

Note: Two versions have been implemented, with version 2 **being the final version used in all later tasks.**

https://github.com/sayedalimohamad/task01.git

Objective:

Develop a Python script to extract articles from the Al Mayadeen website, gather metadata, and save the data in monthly JSON files. This task introduces fundamental web scraping techniques.

Prerequisites:

- Access to the sitemap: Al Mayadeen sitemap
- A computer with internet access
- Basic understanding of Python programming
- Willingness to explore web scraping techniques

The Article Scraper is designed to scrape and store articles efficiently by processing a website's sitemap. It extracts key details such as title, author, content, and metadata, saving this information in JSON format. The scraper is modular, with components for sitemap parsing, URL extraction, and file handling. It also includes error handling and the ability to resume scraping after interruptions, ensuring smooth and continuous operation.

Version 2 (Merged with New Features & Updated Info):

The Al Mayadeen Article Scraper version 2 is a refined Python-based scraper capable of handling large-scale article scraping, now supporting up to 28,000 articles. This version adds several enhancements, such as the ability to extract video duration from embedded videos, while continuing to scrape vital article details like title, author, keywords, language, and word count, saving everything in JSON format.

A significant improvement in version 2 is the check-if-exists feature. Before processing new articles, the scraper verifies if an article has already been scraped and stored in the JSON file, ensuring no duplication occurs. This saves time and resources by skipping already processed articles.

The JSON file is updated in real-time, meaning that each article is immediately saved as soon as it is processed, without waiting for the entire scraping session to finish. This ensures that in case of interruptions, the data collected so far is preserved. The scraper also supports resumption of interrupted scraping sessions, processing only the remaining articles. The 90% threshold check further optimizes performance by skipping months where most articles are

already scraped. The default configuration allows scraping of up to 20,000 articles, which can be modified as needed.

KEY ENHANCEMENTS:

- **Check-if-exists feature:** The scraper checks whether an article has already been scraped by comparing URLs to avoid duplication, saving time and resources.
- **Real-time JSON updates:** Data is saved to JSON immediately after scraping each article, ensuring no loss of data even if the process is interrupted.
- **Resume functionality:** In case of an interruption, the scraper can pick up where it left off, processing only remaining articles.
- **90% threshold check:** If 90% of articles for a given month are already scraped, the scraper skips further processing to optimize efficiency.
- The default configuration allows scraping of up to 20,000 articles, adjustable as needed.

CHALLENGES IN THE CODE:

1. **Handling Intermittent Network Failures:** Ensuring reliable scraping over thousands of articles requires handling frequent network issues, with multiple retry attempts and delays for failed HTTP requests.
2. **Parsing Complex HTML Structures:** Extracting video durations and other metadata from embedded elements like media requires handling diverse and often irregular HTML structures, adding complexity to the scraping process.
3. **Managing Large Data Sets:** Processing and storing up to 28,000 articles across multiple monthly JSON files, while ensuring real-time saving without data loss or duplication, requires careful management of file I/O operations.
4. **Edge Cases and Missing Data:** Articles may have missing or incorrectly formatted metadata (e.g., no publication date or missing keywords), which makes handling edge cases difficult without causing errors in the scraping process.

TASK 02

Task 02 – Storing Data in MongoDB and Building a Flask API.

Note: This task builds on the previous one and is crucial for data management and retrieval.

https://github.com/sayedalimohamad/task02.git

Objective:

Store the data from Task 1 in MongoDB and build a Flask-based web service to access and analyze it via API endpoints, focusing on database interaction and web API creation.

Prerequisites:

- Completion of Task 1 (with data collected from the Al Mayadeen website)
- Basic understanding of databases and how to store/retrieve data
- Introduction to Flask, a Python framework for building web applications and APIs

The data_storage.py script processes the JSON data obtained from the web scraping task. It cleans the data by removing any entries with errors and saves the cleaned data into a new JSON file, ensuring that only valid articles are stored. This script also connects to the MongoDB database and inserts the cleaned data, allowing for efficient storage and retrieval.

KEY FEATURES:

- **Data Cleaning:** The script identifies and removes articles with invalid formats, ensuring data integrity before insertion into the database.
- **MongoDB Integration:** It connects to a local MongoDB instance to store the cleaned data, providing a robust solution for data management.
- **Error Handling:** Includes mechanisms to handle connection failures and file access issues, ensuring reliable operation.

app.py creates a Flask web service that serves multiple API endpoints for accessing and analyzing the stored data. This allows users to query the database for specific metrics, such as top authors, recent articles, and articles by various filters.

KEY API ENDPOINTS OVERVIEW:

- **/top_keywords:**
  - Returns the top 10 most frequent keywords in the articles.
  - Utilizes the $unwind and $group MongoDB operators.
- **/top_authors:**
  - Retrieves the top 10 authors based on the number of articles they've written.
- **/articles_by_date:**
  - Groups articles by their publication date and returns counts for each date.
- **/articles_by_word_count:**
  - Fetches articles grouped by their word count, sorting them by frequency.
- **/articles_by_language:**
  - Groups articles by language and returns the counts.
- **/recent_articles:**
  - Returns the 10 most recent articles based on the published_time field.
- **/articles_by_author/<author_name>:**
  - Filters articles by the specified author and returns their titles.
- **/articles_by_keyword/<keyword>:**
  - Retrieves articles containing a specific keyword.
- **/articles_by_year/<year>:**
  - Returns the count of articles published in the specified year.
- **/longest_articles:**
  - Fetches the longest articles by word count.
- **/shortest_articles:**
  - Fetches the shortest articles by word count.
- **/popular_keywords_last_int:x_days:**
  - Retrieves the most popular keywords used in articles published within the last x days.
- **/articles_with_more_than/int:word_count:**
  - Returns articles with a word count greater than the specified word count.
- **/most_updated_articles:**
  - Retrieves the articles that have been updated the most times after their initial publication.

CHALLENGES IN THE CODE:

1. **Database Connection Management:** Ensuring a stable connection to MongoDB while handling possible failures gracefully.
2. **Data Validation:** Implementing thorough checks to clean and validate data before database insertion to maintain data quality.
3. **API Performance:** Optimizing query performance for potentially large datasets to ensure quick responses from the API.
4. **Complex Aggregation Queries:** Constructing and managing intricate MongoDB aggregation pipelines for various data analysis requirements.

TASK

03 & 04

Note: Task 3 and Task 4 are closely related. The data visualizations created in Task 3 are essential for the advanced analysis and insights in Task 4. The tasks support each other, with the visual representations from Task 3 improving the clarity and effectiveness of the advanced data analysis in Task 4. Additionally, Task 4 includes a significant UI update.

Task 03 – Data Visualization using amCharts

https://github.com/sayedalimohamad/task03.git

Objective:

Task 3 focused on visualizing the data collected and processed in the previous tasks. During this stage, I learned to use amCharts, a powerful JavaScript library, to create interactive and visually appealing charts. These visualizations were essential in understanding patterns and insights from the data more effectively. The prerequisites for this task included the completion of Task 2, ensuring that the Flask API could successfully query data from MongoDB. Additionally, a basic understanding of HTML and JavaScript was required, including familiarity with how to include JavaScript libraries in an HTML page and basic DOM manipulation.

Prerequisites

- Completion of Task 2: Ensure that you have a working Flask API that can query data from MongoDB.
- Basic Understanding of HTML and JavaScript: Familiarity with how to include JavaScript libraries in an HTML page and basic DOM manipulation.

The project utilizes the following technologies and frameworks:

- **HTML** : Used for structuring the web page.
- **CSS** : Used for styling the web page and adding visual effects.
- **JavaScript** : Used for implementing interactive features and manipulating data.
- **Python** : Used for data processing and analysis.
- **Bootstrap** : Used for responsive design and layout.

The goal of this task was to showcase how amCharts can be used to build compelling and interactive visual representations of data. This involved integrating amCharts into the web interface and exploring its range of features, including chart customization and interaction options. Through this project, I demonstrated how data visualization can enhance user experience and make complex data more accessible and comprehensible.

Task 04 – Advanced Data Analysis and Insights

https://github.com/sayedalimohamad/task04.git

Objective:

Task 4 focused on performing advanced data analysis and extracting meaningful insights from the data collected, stored, and visualized in the previous tasks. This involved applying data science techniques such as sentiment analysis, entity recognition, and trend analysis. Additionally, I enhanced the Flask API to serve these advanced analytics and insights. The prerequisites for this task included the completion of Task 3, ensuring that visualizations had been created and embedded using amCharts. A basic understanding of Natural Language Processing (NLP) was required, particularly with concepts like sentiment analysis and entity recognition. Additionally, knowledge of Python libraries such as NLTK, spaCy, or TextBlob was essential for text processing.

Prerequisites

- Completion of Task 3: Ensure that you have created and embedded visualizations using amCharts based on your data.
- Basic Understanding of NLP (Natural Language Processing): Familiarity with concepts like sentiment analysis and entity recognition.
- Knowledge of Python Libraries: Experience with libraries like NLTK, spaCy, or TextBlob for text processing.

Flask Web Application with MongoDB and Authentication

The project includes a Flask web application that connects to a MongoDB database and provides various routes for retrieving and analyzing articles. Additionally, it includes user authentication to manage access to certain functionalities.

➢ Imports and Initialization:

The necessary modules and libraries are imported and initialized:

- Flask (along with jsonify, request, Response, render_template, redirect, url_for, and flash) is imported to handle web requests and templates.
- MongoClient from pymongo is used to interact with the MongoDB database.
- Standard Python libraries like datetime and json are used for handling dates and data formats.
- ObjectId from bson is used to work with MongoDB ObjectIds.
- Flask-Login is utilized for user authentication, importing LoginManager, UserMixin, login_user, login_required, and logout_user.

The Flask app is initialized and connected to a MongoDB client, enabling interaction with a local MongoDB instance.

➢ Routes

- Authentication
  - /login: Handles user login.
  - /logout: Logs out the user.
- Admin Credentials:
  - ADMIN_USERNAME = 'admin'
  - ADMIN_PASSWORD = 'password'
- Home and Static Pages
  - /: Renders the home page with a list of available routes.
  - /gui.html, /advanced.html: Render static HTML pages.
- Data Retrieval and Aggregation
  - /top_keywords: Returns the top keywords used in articles.
  - /top_authors: Returns the top 20 authors by article count.
  - /all_authors: Returns all authors sorted by article count.
  - /articles_by_date: Groups articles by publication date.
  - /articles_by_word_count: Groups articles by word count.
  - /articles_by_language: Groups articles by language.
  - /articles_by_classes: Groups articles by classes.
  - And More …

➢ Requirements

- Python 3.6+
- MongoDB
- Required Python packages (specified in requirements.txt):
  - Flask==3.0.3
  - pymongo==4.8.0
  - stanza==1.8.2
  - nltk==3.9.1
  - ftfy==6.2.3
  - flask_login==0.6.2

➢ Configuration

The MongoDB connection details are hardcoded within the script. Modify the connection string if your MongoDB instance is running on a different host or port.

Functions

- ensure_model_downloaded(lang_code): Ensures that the Stanza model for the specified language is downloaded.
- safe_decode(text): Safely decodes text using ftfy to fix encoding issues.
- process_document(document, idx): Processes a single document by:
    - Decoding the text.
    - Performing sentiment analysis.
    - Performing named entity recognition (NER).
    - Updating the document in MongoDB with the analysis results.
- analyze_texts(): Fetches documents from MongoDB and processes them concurrently.

CHALLENGES IN THE CODE:

1. **Handling Concurrent Requests Efficiently**
    - The application performs multiple data retrieval and aggregation tasks that may involve querying large datasets from MongoDB. Managing and optimizing these concurrent queries without overloading the server or causing performance issues was a significant challenge.
    - **Solution**: Implemented ThreadPoolExecutor to process documents concurrently in the advanced_data_analysis.py script, improving performance for tasks like sentiment analysis and entity recognition.
2. **MongoDB Query Optimization**
    - Aggregating large collections of articles and performing tasks like grouping by date, keywords, or authors can result in slow database queries, especially when dealing with real-time requests.
    - **Solution**: Added appropriate indexes on fields frequently queried, such as author, publication date, and keywords, to enhance query performance. Used MongoDB's aggregation framework to improve efficiency for complex queries.
3. **User Authentication and Security**
    - Implementing a secure user authentication system, especially with hardcoded admin credentials, posed a security risk.
    - **Solution**: Plan to implement hashed passwords using libraries like bcrypt and move away from hardcoded credentials to a user database for improved security. Flask-Login was integrated to manage user sessions securely.
4. **Handling Data Inconsistencies**
    - The web scraping task extracted data from the Al Mayadeen website, but the structure of the scraped data (e.g., missing metadata, inconsistent formats) posed difficulties when storing it in MongoDB and performing subsequent analyses.
    - **Solution**: Wrote data validation and cleaning scripts to ensure the data was consistent before storing it in the database. Used regex and error handling to manage unexpected data formats.
5. **Integration of Sentiment Analysis and NER for Arabic Text**
    - Sentiment analysis and entity recognition (NER) for Arabic text using libraries like Stanza was challenging due to the complexity of the language and the need for specialized models.

- **Solution**: Carefully selected and downloaded the appropriate Stanza models for Arabic. Ensured that the NLP models were accurate and performed well on the dataset, although this required considerable tuning and testing.

6. **Cross-Language Support for Articles**
   - Articles in different languages required special handling, especially for tasks like sentiment analysis and keyword extraction. This created challenges when trying to apply a one-size-fits-all solution.
   - **Solution**: Implemented language detection mechanisms to route articles to the appropriate analysis pipeline (e.g., Arabic articles through Stanza). Used separate logic for multilingual support when handling keywords, entity recognition, and sentiment.
7. **Maintaining API Consistency and Response Formats**
   - With many API routes for querying articles by various attributes, it was important to maintain consistent response formats (e.g., JSON) and error handling.
   - **Solution**: Created utility functions like convert_objectid_to_str to ensure ObjectId fields were properly serialized in the JSON responses. Applied error handling at each route to return informative error messages in case of failure.
8. **Managing Model Downloads for NLP Tasks**
   - Ensuring the right NLP models (e.g., Stanza for Arabic) were available during runtime was a logistical challenge, especially on different systems or deployment environments.
   - **Solution**: Built the ensure_model_downloaded function to check and automatically download required models at runtime, avoiding issues with missing models.
9. **Implementing Data Visualizations Using amCharts**
   - While creating interactive charts with amCharts, handling dynamic data from the MongoDB backend required careful integration with JavaScript on the frontend.
   - **Solution**: Developed API routes in Flask that provide data in the required format for amCharts. Managed the flow of data between the Flask backend and the frontend efficiently to ensure smooth chart rendering.
10. **Error Handling for Web Scraping**
    - The web scraping process was prone to failures due to unpredictable changes in the target website's structure or connection issues, which led to incomplete or erroneous data.
    - **Solution**: Implemented robust error handling with requests exceptions (e.g., RequestException) and retry mechanisms to handle transient errors and ensure the stability of the scraping process.