** Al Maaref University**

**Faculty of Sciences**

**Department of Computer Science**

Summer 2023 - 2024

Practical Training (CSC 497) Report

**Student ID** : 10120048

**Student Name :** Mohamad Ali MOnzer

Al Mayadin data analysis

**Bootcamp Data Science (Python)**

Company Coordinator : Kassaem Shhade

University Advisor : DR. Kassem Danash

Contents

[Introduction 4](#_Toc177747573)

[Data Collection Process 4](#_Toc177747574)

[1. Crawling the Sitemaps 4](#_Toc177747575)

[2. Storing Data: 4](#_Toc177747576)

[3. Python for Automation: 4](#_Toc177747577)

[4.Network Speed Monitoring: 4](#_Toc177747578)

[Hardware and Software Overview 5](#_Toc177747579)

[Hardware Used: 5](#_Toc177747580)

[Linux Server: 5](#_Toc177747581)

[Laptop for Crawling: 5](#_Toc177747582)

[Software Used: 5](#_Toc177747583)

[Linux OS: 5](#_Toc177747584)

[Python: 5](#_Toc177747585)

[MongoDB: 6](#_Toc177747586)

[Network Speed Monitoring Tools: 6](#_Toc177747587)

[Data Crawling Implementation 6](#_Toc177747588)

[requests and Beautiful Soup: 6](#_Toc177747589)

[concurrent. Futures: 6](#_Toc177747590)

[MongoDB and pymongo: 6](#_Toc177747591)

[Resiliency with tenacity: 6](#_Toc177747592)

[Data Structure: 7](#_Toc177747593)

[Crawling Strategy and Efficiency 7](#_Toc177747594)

[Utilizing Multiple CPUs: 8](#_Toc177747595)

[Avoiding Data Duplication in MongoDB: 8](#_Toc177747596)

[Efficient Crawling by Time Period: 8](#_Toc177747597)

[Web Application and Data Visualization 8](#_Toc177747598)

[Static Data from JSON Files: 8](#_Toc177747599)

[Real-Time Data from MongoDB: 8](#_Toc177747600)

[Efficient Data Crawling and Processing: 9](#_Toc177747601)

[Articles by Coverage: 9](#_Toc177747602)

[Most Popular Keywords in the Last 7 Days: 9](#_Toc177747603)

[Articles by Published Month: 9](#_Toc177747604)

[Articles by Word Count Range: 9](#_Toc177747605)

[Articles with Specific Keyword Count: 9](#_Toc177747606)

[Articles by Specific Date: 9](#_Toc177747607)

[Advanced Data Analysis and API Enhancements 10](#_Toc177747608)

[Sentiment Analysis 10](#_Toc177747609)

[Entity Recognition 10](#_Toc177747610)

[Trend Analysis 10](#_Toc177747611)

[API Enhancements for Advanced Analytics 10](#_Toc177747612)

[Creating a Comprehensive Dashboard 11](#_Toc177747613)

[User Authentication and Role-Based Access 11](#_Toc177747614)

[Real-Time News Integration 11](#_Toc177747615)

[Conclusion 11](#_Toc177747616)

[1. Explore Historical and Real-Time Data: 12](#_Toc177747617)

[2. Perform Data Analysis: 12](#_Toc177747618)

[o Sentiment Trends: 12](#_Toc177747619)

[o Entity Mentions: 12](#_Toc177747620)

[o Keyword Usage: 12](#_Toc177747621)

[Software Engineering Best Practices 12](#_Toc177747622)

[Code Refactoring and Best Practices 12](#_Toc177747623)

[ Code Refactor 12](#_Toc177747624)

[SOLID Principles: 12](#_Toc177747625)

[ DRY (Don't Repeat Yourself): 13](#_Toc177747626)

[Design Patterns 13](#_Toc177747627)

[Configuration and Environment Management 13](#_Toc177747628)

[Testing and Technical Debt Management 13](#_Toc177747629)

[ Tests: 13](#_Toc177747630)

[ Technical Debt: 13](#_Toc177747631)

[Deployment and Hosting Options 14](#_Toc177747632)

[1. Cloud Platforms: 14](#_Toc177747633)

[2. NGINX for Local or Server-Based Hosting: 14](#_Toc177747634)

# Internship Information

Training Title : Al Mayadin data analysis

Company Department : computer science

Internship Duration (hours) : > 50 hours

Company Coordinator : Kassem Shhade

University Advisor : dr. Kassem Danash

# Internship Abstract

During my internship, I developed a comprehensive data analysis pipeline and web application focused on extracting and visualizing insights from online news sources, particularly Al Mayadin. The project involved the use of **Python** for data crawling, **MongoDB** for data storage, and **Flask** for building an interactive dashboard that serves both historical and live news data.

I utilized **multi-threading** and **process pooling** to optimize the data crawling process, ensuring efficient use of CPU resources. By organizing the data collection process by year and month, I minimized redundancy and improved the efficiency of the crawling operation. The application also includes advanced data visualizations created with **amCharts** and **Highcharts**, allowing users to explore data through visual representations like stacked bar charts, line charts, and pie charts.

For **natural language processing (NLP)**, I employed tools such as **Khaled analysis** for sentiment analysis and **Stanza** for entity recognition, enabling deeper insights into the text of the articles. The dashboard allows users to filter data based on sentiment, keyword trends, and entity mentions. Moreover, I implemented **user authentication**, ensuring that different levels of access are granted to users, with root privileges reserved for managing the full range of dashboard functionalities.

# Introduction

In this project, I am conducting data analysis using a Linux server equipped with Python, MongoDB, and network speed metrics. To gather the data, I am using another laptop to crawl and extract information from the Al Mayadin news website. The primary goal is to collect and analyze news articles across different months.

For crawling the data, I am utilizing the sitemap provided by Al Mayadin at [https://www.almayadeen.net/sitemaps/all.xml](https://www.almayadeen.net/sitemaps/all.xml), which contains links to monthly sitemaps. Each monthly sitemap follows the format `https://www.almayadeen.net/sitemaps/all/sitemap-YYYY-MM.xml`. By accessing these monthly sitemaps, I can retrieve all news articles published during that specific month.

# Data Collection Process

## 1. Crawling the Sitemaps

- I started by accessing the main sitemap link, which contains references to all monthly sitemaps.

- From there, I looped through each monthly sitemap (e.g., `https://www.almayadeen.net/sitemaps/all/sitemap-2023-09.xml`), extracting all article URLs listed for that particular month.

## 2. Storing Data:

- I am storing the extracted URLs and corresponding metadata in a MongoDB database. This allows for efficient querying and retrieval of the news articles for further analysis.

## 3. Python for Automation:

- The crawling and data extraction are automated using Python scripts. These scripts fetch the sitemap data, parse it, and store it in the database. I am also utilizing libraries like `requests` and `BeautifulSoup` to handle the HTTP requests and data parsing.

## 4.Network Speed Monitoring:

- Since the data extraction process involves fetching large amounts of information over the web, I am also tracking the network speed to ensure that data is being retrieved efficiently. Monitoring the network speed helps in optimizing the crawling process by adjusting timeouts and managing retries if the connection slows down.

# Hardware and Software Overview

In this project, both hardware and software components play critical roles in enabling the data analysis and web crawling process.

## Hardware Used:

### Linux Server:

* + I am using a Linux-based server that serves as the primary environment for processing and storing data. The server handles running Python scripts and managing a MongoDB instance.
  + The server’s specifications include a multi-core processor, ample storage, and sufficient RAM to handle large datasets efficiently. This ensures smooth execution of scripts and supports fast database queries.
  + I chose Linux for its stability and resource management capabilities, which are ideal for running data-intensive tasks over long periods without frequent reboots.

### Laptop for Crawling:

* + I am using a separate laptop to perform the web crawling tasks. The laptop is connected to the internet and runs Python scripts that access and download data from the Al Mayadin website.
  + The laptop’s specifications are more modest compared to the server, but it’s powerful enough to handle HTTP requests, parse large XML sitemaps, and transfer the data to the server for further analysis.

## Software Used:

### Linux OS:

* + Both my server and laptop run Linux. On the server, the Linux OS helps manage system resources efficiently. I am also using Linux’s built-in tools to monitor system performance, especially when running multiple tasks concurrently.

### Python:

* + I am using Python as the primary language for scripting. Python’s extensive library support makes it ideal for web scraping, data processing, and working with databases.
  + Libraries like requests and BeautifulSoup are used for crawling and parsing HTML/XML data from the sitemaps. Python’s simplicity and versatility make it easy to automate repetitive tasks, like fetching monthly sitemaps.

### MongoDB:

* + MongoDB is used as the database for storing the crawled data. I chose MongoDB because it’s a NoSQL database that stores data in a flexible JSON-like format, which works well for handling web-scraped data that often contains irregular or nested structures.
  + MongoDB is installed on the Linux server, and the data collected from the web crawler is automatically inserted into the database for future queries and analysis.

## Network Speed Monitoring Tools:

* + Since I am working with large amounts of data and frequently transferring data between the laptop and server, I am using network speed monitoring tools to keep track of connection performance. Tools like speedtest-cli allow me to measure download and upload speeds, helping me optimize the data crawling process.

## Data Crawling Implementation

To efficiently crawl and structure the data from the Al Mayadin website, I developed a Python-based web scraping script using several key libraries. The following components were utilized:

### requests and Beautiful Soup:

* + I used the requests library to send HTTP requests to the website and retrieve HTML/XML content.
  + Beautiful Soup from the bs4 library was used to parse the HTML/XML structure, which allowed me to navigate the document tree and extract specific pieces of data (e.g., URLs, article titles, and metadata).

### concurrent. Futures:

* + To improve performance and reduce crawling time, I used Python’s ThreadPoolExecutor and ProcessPoolExecutor for concurrent execution of multiple web requests. This enabled fetching and processing multiple articles simultaneously, speeding up the data retrieval process.

### MongoDB and pymongo:

* + The pymongo library is used to connect to the MongoDB database where the crawled data is stored. MongoDB’s schema-less structure allows flexible storage of article information, even when different articles have varying attributes.

### Resiliency with tenacity:

* + I incorporated the tenacity library to handle retry logic for failed web requests. This ensures that if a request fails due to network issues or server timeouts, the program retries the request up to a specified number of times before stopping.

### Data Structure:

* + I defined an Article dataclass to store the structured information of each crawled article. This class captures various metadata attributes extracted from the news articles.

Here’s a breakdown of the fields in the Article dataclass:

@dataclass

class Article:

url: str # The URL of the article

postid: Optional[str] = None # Unique identifier for the article

title: Optional[str] = None # Title of the article

keywords: List[str] = field(default\_factory=list) # List of keywords related to the article

thumbnail: Optional[str] = None # URL of the article's thumbnail image

video\_duration: Optional[str] = None # Duration of the video if the article contains one

word\_count: Optional[str] = None # The word count of the article

lang: Optional[str] = None # Language of the article

published\_time: Optional[str] = None # Time when the article was first published

last\_updated: Optional[str] = None # Last updated time of the article

description: Optional[str] = None # A brief description of the article

author: Optional[str] = None # Author of the article

classes: List[Dict] = field(default\_factory=list) # List of classes or categories assigned to the article

text: Optional[str] = None # Full text of the article

filename: Optional[str] = None # Local filename where the article is saved (if applicable)

Each field represents a specific aspect of the article that is important for analysis. For example, the url holds the link to the article, title captures the title, keywords contain a list of SEO-related terms, and published\_time records the article’s release date.

This structured approach using the Article dataclass ensures that all the necessary data points for analysis are captured and organized efficiently for further processing and storage in the MongoDB database.

## Crawling Strategy and Efficiency

My primary goal in this project is to optimize the data crawling process, ensuring that it is both fast and reliable. To achieve this, I focused on several key areas:

### Utilizing Multiple CPUs:

* + One of my aims was to leverage the multiple CPU cores available on my laptop to parallelize the data crawling process. By using Python’s concurrent.futures.ThreadPoolExecutor and ProcessPoolExecutor, I ensured that multiple URLs could be processed simultaneously, making full use of my system's computational power. This approach significantly reduces the total time required to crawl large datasets, especially when working with a large number of articles.

### Avoiding Data Duplication in MongoDB:

* + When sending crawled data to MongoDB, I implemented mechanisms to prevent data duplication. Each article has a unique identifier (typically derived from the URL or post ID). Before inserting new data, my script checks if an article with the same identifier already exists in the database. This ensures that only new articles are added, maintaining data integrity and avoiding redundant entries.

### Efficient Crawling by Time Period:

* + To further improve the efficiency of the crawling process, I structured the crawler to allow data retrieval based on specific time periods, such as by year or by month. This approach prevents the need to start from the very beginning of the sitemap every time. For example, if I only need articles from a particular year or month, I can directly target that specific sitemap (e.g., sitemap-2023-09.xml) rather than re-crawling the entire site. This flexibility makes the crawling process more manageable, especially for long-term projects where new data is added regularly.

## Web Application and Data Visualization

To present the collected data in an interactive format, I am using **Flask** to develop a web application. Flask serves as the backend, providing the necessary API endpoints and data management features. The web app connects to **MongoDB** in two ways:

### Static Data from JSON Files:

* + For older data (e.g., articles from 2014 to 2020), I use pre-extracted JSON files, which were generated from the data collected during the crawling process. These files serve as a fixed data source, allowing the app to quickly load and visualize historical data.

### Real-Time Data from MongoDB:

* + For more recent data, I use MongoDB pipelines to query the database directly. This allows for real-time updates in the data, ensuring the latest articles are always available for analysis. By implementing this two-way data connection, I ensure that the system is both scalable and efficient.

### Efficient Data Crawling and Processing:

* To optimize the crawling process, I am utilizing **multiple CPUs** on my laptop. This allows me to fetch large datasets efficiently by parallelizing the crawling tasks.
* When inserting the crawled data into MongoDB, I ensure that no duplicates are introduced, keeping the database clean.
* Additionally, I crawl data by year or month, so I don’t have to restart the process from the beginning every time, which saves time and computing resources.

**Data Visualization**: To analyze and present the data, I created a variety of visualizations using **AM4Chart** and **Highcharts**. Here are some examples of the key charts implemented:

### Articles by Coverage:

* + **Recommended Visualization**: Stacked Bar Chart
  + **Explanation**: A stacked bar chart displays articles categorized by their coverage, allowing for easy comparison across multiple categories.

### Most Popular Keywords in the Last 7 Days:

* + **Recommended Visualization**: Line Chart
  + **Explanation**: This chart tracks the frequency of keywords used over the past week, helping to highlight trends and spikes.

### Articles by Published Month:

* + **Recommended Visualization**: Column Chart
  + **Explanation**: This chart shows the number of articles published per month, identifying seasonal or monthly trends.

### Articles by Word Count Range:

* + **Recommended Visualization**: Histogram
  + **Explanation**: A histogram is used to show the distribution of articles based on their word count, providing insight into the length of articles over time.

### Articles with Specific Keyword Count:

* + **Recommended Visualization**: Pie Chart
  + **Explanation**: A pie chart displays the proportion of articles that contain a specific number of keywords, helping to analyze keyword density.

### Articles by Specific Date:

* + **Recommended Visualization**: Line Chart
  + **Explanation**: A line chart visualizes the number of articles published on a given date, making it easy to spot any significant activity on that day.

These visualizations help provide meaningful insights into the data, enabling deeper analysis of trends and patterns within the news articles.

## Advanced Data Analysis and API Enhancements

In **Week 4**, the focus is on performing advanced data analysis using techniques such as sentiment analysis, entity recognition, and trend analysis. This enables deeper insights into the news articles collected from the web.

### Sentiment Analysis

Using the **Khaled analysis** tool, I perform sentiment analysis on the text of the articles to determine whether they have a **positive**, **negative**, or **neutral** sentiment. The results of the analysis are then stored in MongoDB under a new sentiment field for each article.

* **API Enhancements**: I’ve extended the Flask API to allow sentiment-based queries. For example, users can filter articles by sentiment through endpoints like /articles\_by\_sentiment/positive to fetch articles with positive sentiment.

### Entity Recognition

To extract **named entities** (such as people, places, and organizations) from the articles, I use **Stanza**, which helps categorize and identify entities in the text. These entities are stored in MongoDB within an entities field, making it possible to query articles based on specific entities.

* **API Enhancements**: New API endpoints have been created to support entity-based queries, such as /articles\_by\_entity/<entity>. This allows users to fetch articles mentioning a specific entity, like /articles\_by\_entity/Israel.

### Trend Analysis

I also perform **trend analysis** to visualize how topics, sentiment, or entity mentions evolve over time. Using the **Stanza** tool for categorical analysis, I track changes in keyword usage or sentiment to provide a dynamic view of trends within the news.

* **Visualizations**: I generate charts such as **line charts** and **bar charts** to visualize these trends. These trends are made available via the API, and endpoints like /sentiment\_trends or /keyword\_trends can be used to retrieve this data.

### API Enhancements for Advanced Analytics

In addition to sentiment and entity recognition, I’ve added several new endpoints to the Flask API to handle advanced analytics, including:

#### Most Positive/Negative Articles:

* + Endpoint: /most\_positive\_articles and /most\_negative\_articles.
  + Description: Retrieves articles with the highest or lowest sentiment scores.

#### Articles by Entities:

* + Endpoint: /articles\_by\_entity/<entity>.
  + Description: Fetches all articles that mention a specific entity.

#### Trend Data:

* + Endpoint: /trends/<keyword> or /trends/<entity>.
  + Description: Provides trend data over time for specific keywords or entities, allowing visualization of topic evolution.

### Creating a Comprehensive Dashboard

I have also built a comprehensive dashboard that integrates all the **visualizations** and insights derived from the data analysis. This dashboard allows users to interact with the data through features such as filtering by:

* **Sentiment** (positive, negative, neutral)
* **Entities** (people, organizations, locations)
* **Date Ranges** (to explore trends over time)

The dashboard is built using **Flask** and **AM4Chart/Highcharts** for visualizations, providing an intuitive user interface for exploring news article trends, sentiment, and entities.

## User Authentication and Role-Based Access

To manage access to the dashboard, I’ve implemented a **login system** within the Flask application. This system provides **role-based privileges**, ensuring that different users have appropriate levels of access:

* **Root Users**: The root user has full access to the entire dashboard, including the ability to use all data visualizations, perform advanced data analysis, and access the API for sentiment analysis, entity recognition, and trend analysis.
* **Other Users**: Regular users can log in to view the dashboard but have limited functionality, allowing them to only explore the pre-generated data visualizations without making changes or accessing sensitive API endpoints.

## Real-Time News Integration

In addition to analyzing historical news data, I’ve added functionality to display **live news updates** directly in the dashboard. By integrating **Flask live streaming** features, users can now access real-time data from the **Al Mayadin** news channel, ensuring that they are always viewing the most current information alongside the historical analysis.

* **API for Live News**: A special Flask route is created to pull the latest news articles as they are published, allowing users to stay up to date with the news in real time, while continuing to analyze trends based on historical data.

# Conclusion

By implementing this system, users of the dashboard can:

1. Explore Historical and Real-Time Data: They can view both past and live news articles, and use the tools available in the dashboard to analyze sentiment, track entities, and observe trends over time.
2. Perform Data Analysis: The system allows for in-depth analysis of Al Mayadin's online news data, providing insights such as:
   * Sentiment Trends: Understanding how public sentiment has shifted over time in relation to certain news topics or keywords.
   * Entity Mentions: Observing which people, organizations, or places have been most frequently mentioned in the news.
   * Keyword Usage: Tracking how often specific keywords are used, identifying emerging topics, and predicting future trends.

Through this analysis, users can benefit from insights that are crucial for **media monitoring**, **public opinion analysis**, and **content strategy planning**. The ability to analyze and visualize this data helps users and researchers understand the impact of media coverage, the frequency of certain topics, and how public sentiment evolves in response to global events.

# Software Engineering Best Practices

To ensure that the system is robust, maintainable, and scalable, I have applied several key software engineering principles throughout the development of this project:

## Code Refactoring and Best Practices

* Code Refactor: The codebase has been refactored to improve readability, maintainability, and performance. This involves cleaning up redundant code, simplifying complex functions, and ensuring consistent code structure across the application.

### SOLID Principles:

* + Single Responsibility Principle (SRP): Each module, class, or function in the system has a clear, single responsibility. For example, separate classes handle user authentication, data fetching, and visualization generation, ensuring each component is focused and reusable.
  + Open/Closed Principle: The application is designed in such a way that it can be extended with new features without modifying existing code. This is crucial for adding new visualizations, API endpoints, or authentication features in the future.
  + Liskov Substitution Principle, Interface Segregation, and Dependency Inversion are also adhered to, promoting cleaner interfaces and reducing dependencies between components.
* DRY (Don't Repeat Yourself): I have eliminated redundant code throughout the project, ensuring that logic is not repeated. Shared functionality is centralized into utility functions or helper classes, reducing the possibility of bugs and making the system easier to update.

### Design Patterns

To structure the system effectively, I’ve used common **design patterns** such as:

* Factory Pattern: For generating different types of visualizations dynamically (e.g., bar charts, line charts, pie charts) based on the type of data analysis requested.
* Singleton Pattern: For managing database connections, ensuring that only one instance of the MongoDB connection is created and reused throughout the application.
* Observer Pattern: To manage real-time updates of live news data in the dashboard, ensuring that when new data arrives, the dashboard reflects those updates automatically.

### Configuration and Environment Management

The system is configured to use environment variables and configuration files, ensuring that sensitive information (such as API keys and database credentials) is not hardcoded into the application. This approach allows for easy environment switching between development, testing, and production setups without modifying the core application code.

* Environment Management: Separate configuration files (e.g., .env) are used for different environments to manage settings such as debug modes, database URLs, and API credentials securely.

## Testing and Technical Debt Management

* Tests: I have implemented both **unit tests** and **integration tests** to ensure that each component of the application works as expected. The testing framework validates that data fetching, API responses, and visualizations work correctly under various scenarios.
* Technical Debt: By regularly refactoring and adhering to best practices like **SOLID** and **DRY**, I have kept **technical debt** low, ensuring that future enhancements can be made without causing significant disruption to the codebase.

# Deployment and Hosting Options

To make this project accessible online, we can deploy the application using various platforms:

1. Cloud Platforms: The project can be hosted on cloud services like **AWS**, **Google Cloud**, or **Heroku** to ensure scalability, availability, and easy access for users across different regions. By storing the data online and using services such as **Amazon S3** for files and **Amazon RDS** or **MongoDB Atlas** for databases, the application can handle large datasets and real-time data efficiently.
2. NGINX for Local or Server-Based Hosting: A simpler approach would be to use **NGINX** as a reverse proxy to serve the Flask application on a local server or a small-scale cloud server. This is a lightweight solution that allows the application to run on smaller servers while maintaining performance and reliability.

By deploying the application, users can access the dashboard and perform real-time data analysis from anywhere, enhancing its practical utility and accessibility.

Finally, I have added the project to a GitHub repository for easy collaboration and version control: [dgPadBootcamps-MIS\_MohamadMonzer](https://github.com/Tuelles00/dgPadBootcamps-MIS_MohamadMonzer.git).