CMPUT 291 Mini Project 2 Report

Group Members:
obiokoye, Chidinma Obi-Okoye
diepreye, Diepreye Charles-Daniel
akpulonu, Ugonna Noble Jr Akpulonu

Fall 2025

1 System Overview & User Guide

1.1 System Overview

Our system is a robust document store application designed to process and analyze large datasets of news and blog articles using MongoDB and Python. The system architecture consists of two primary components:

1. **Data Loader (`load-json.py`):** A high-performance batch processing tool that ingests raw JSON data (supporting both line-delimited and array formats) into a MongoDB collection. It automatically manages database connections, ensures clean collection states, and generates performance-critical indexes.
2. **Query Engine (`phase2_query.py`):** An interactive Command Line Interface (CLI) that enables users to perform complex analytic queries on the document store, implementing strict validation logic for accurate results.

1.2 User Guide

Phase 1: Loading Data
To initialize the database, the user executes the loader script with the input file and port number. The system provides real-time progress indicators.

```
python load-json.py <filename.json> <port>
```

Phase 2: Querying
To launch the analytics engine, the user executes:

```
python phase2_query.py <port>
```

The interactive menu supports four key operations:

- **Word Frequency Analysis:** Analyzes top alphanumeric words for a given media type ("News" or "Blog").
- **Article Volume Comparison:** Compares publication counts between News and Blogs for a specific date provided by the user.
- **Top News Sources:** Identifies the top 5 news sources by volume for the year 2015.
- **Recent Articles Fetcher:** Retrieves the 5 most recent articles for a specified source.

2 Assumptions & Design Decisions

To ensure robustness and strict adherence to the project specifications (specifically the Nov 17 and Nov 21 clarifications), we implemented the following design decisions:

1. **Definition of a "Word" (Query 1):** The specification defined a word as "any contiguous sequence of alphanumeric characters, including hyphens and underscores."
    - *Decision:* We rejected a "blacklist" approach (removing specific punctuation) in favor of a "whitelist" Regex strategy. We utilized the regex `[a-zA-Z0-9_-]+` within the MongoDB Aggregation Pipeline via `$regexFindAll`. This ensures that mixed-character words (e.g., "3D", "R2-D2") are correctly identified while strictly ignoring other punctuation.
2. **Tie-Breaking Logic (Nov 17 Clarification):** The requirement stated that all items tied at the 5th position must be included.
    - *Decision:* We deliberately avoided using a hard database limit (e.g., `.limit(5)`). Instead, our application fetches a larger result set and implements Python-side logic to identify the count at the 5th rank, subsequently including all additional items that match this count.
3. **Date Parsing Robustness (Query 2):** User input for dates varies significantly (e.g., "Sept 1, 2015" vs "2015-09-01").
    - *Decision:* We implemented a multi-format date parser in Python. For the database lookup, we utilized a Regex prefix match (e.g., `^2015-09-01`) on the `published` field. This provides greater robustness than constructing ISO timestamp ranges, effectively bypassing potential timezone offsets or formatting inconsistencies.
4. **Performance Optimization:**
    - *Decision:* We automated the creation of compound indexes (specifically `source + published: -1`) at the end of the Phase 1 loading process. This ensures that Phase 2 queries perform efficiently without triggering full collection scans.

3 Group Work Break-down Strategy

3.1 Communication Method

To ensure the project stayed on track, our group adhered to the following communication strategy:

- **In-Person Meetings:** We met twice weekly on campus for pair programming sessions and architectural planning.
- **Daily Updates:** We utilized a Discord server for daily status checks and file sharing.
- **Version Control:** We used GitHub for merging code and tracking changes.

3.2 Work Breakdown

All group members contributed equally to the project's success. The distribution of tasks is detailed below:

Partner	Tasks Assigned	Est. Time	Status
Chidinma	Developed `load-json.py` (batch insertion, streaming); Performed testing & performance optimization; Wrote README Installation instructions.	4 hrs	Complete
Ugonna	Developed `phase2_query.py` structure & menu system; Implemented Query 1 (Word Freq) & Query 2 (Date Diff); Handled user input parsing.	4 hrs	Complete
Diepreye	Implemented Query 3 (Top Sources) & Query 4 (Recent Articles); Compiled Report.PDF; Wrote README Group Info & Sources sections.	4 hrs	Complete

Table 1: Distribution of Work Items