CS 499 Milestone Four - Databases

For this artifact, I am focusing on the creation and enhancement of the MongoDB database that powers the backend of my Political Bias Tracker application. The database is hosted online using MongoDB Atlas, allowing seamless access from both the backend Node.js server and the data ingestion scripts. To maintain security and modularity, the database connection URI was stored securely in a .env file and accessed programmatically via process.env.MONGO\_URI. This ensured sensitive credentials were not hard-coded into the project and could be easily reused across multiple scripts and environments.

To populate the initial dataset, I sourced a publicly available CSV file from the AllSides GitHub repository (https://github.com/favstats/AllSideR/blob/master/data/allsides\_data.csv). This dataset contains bias ratings for hundreds of news publications. After downloading the CSV, I defined a MongoDB schema using Mongoose in a file called publication.js, which included fields such as news\_source, rating, rating\_num, type, and url (for the AllSides page). I then wrote a custom script, importPublications.js, to convert the CSV data into JSON. This involved trimming unnecessary whitespace, validating fields (e.g., ensuring rating\_num was numeric), and filtering out any incomplete records before inserting the data into the MongoDB collection.

Initially, the dataset only included the AllSides profile URL for each publication. However, in order to support user queries based on actual news article URLs (e.g., from bbc.com, cnn.com, etc.), the database needed to include the publication's root domain. To solve this, I designed a second script that leveraged Puppeteer, a Node.js library for headless browser automation, to visit each AllSides profile page and scrape the linked official website URL for each news source. This involved launching a headless browser, navigating to the AllSides page, and extracting the href from the "Website" row of the info table.

Once the raw website URL was scraped, I passed it through a utility function called cleanUrl.js, which stripped prefixes such as https:// and www., and removed trailing slashes to normalize the domain (e.g., turning https://www.bbc.com/ into bbc.com). This cleaned domain was then saved to the database using Mongoose's document update methods.

To avoid scraping blocks by Cloudflare, I used the puppeteer-extra library along with the stealth plugin to mask the headless nature of the browser. I also implemented a delay between each scrape to avoid rate limiting. This full automation loop enabled me to enrich the original dataset with valid domain data, ultimately supporting the core feature of the application: allowing users to paste any news article URL into a search bar and receive an accurate political bias rating based on the domain.

I selected this artifact because it showcases a substantial improvement over the original static database used in the Travlr CRUD app. While that database contained only manually entered fields, this artifact demonstrates dynamic ingestion, transformation, enrichment, and queryability of real-world data. It reflects my ability to integrate multiple tools and techniques—including schema design, environment configuration, automation with Puppeteer, and API development—to create a robust and functional backend service.

Through the process of building and enhancing this database, I gained practical experience in asynchronous scripting, troubleshooting MongoDB schema errors, working with CSV-to-JSON transformations, and scraping dynamic websites. One of the most valuable lessons I learned was how to creatively solve obstacles, such as bypassing Cloudflare's bot detection using stealth plugins. This reinforced the importance of persistence and adaptability in software development—where there's a will, there's often a technical way forward. The end result is a scalable and extensible backend that significantly improves the user experience of the app and showcases my full-stack capabilities.