Milestone Three: Enhancement Two: Algorithms and Data Structures

The artifact I selected for this milestone is a full-stack web application originally developed as a console-based Java Program for managing rescue animals. It enhances a Java-based console program I originally developed in IT-145, which used arrays and procedural logic to handle user input and manage animal data in memory. The original application operated entirely through a terminal-based interface, using ArrayList collection to manage separate lists for dogs, monkeys, and available animals. Each operation, such as adding a new animal or searching for one to reserve, required the program to manually iterate through lists using basic for-loops. For example, the reserveAnimal() method used a linear search though the dogList or monkeyList to find a match based on name and country, which resulted in O(n) time complexity for those operations. This works fine with small datasets, but it was inefficient and lacked scalability.

To align with the Algorithms and Data Structures category, I enhanced the application to include dynamic filtering, sorting, and efficient querying capabilities. In this updated version, animal records are stored in MongoDB, and the backend is built using Spring Boot while the frontend uses Angular. These changes allowed me to redesign the filtering logic using database queries instead of in-memory loops, and to incorporate multiple filters, such as animal type, size, age range, and availability, along with sorting by name, age, or weight.

The filtering system was one of the most important enhancements. In the original program, searching for an animal to reserve involved scanning through each item in the dogList or monkeyList until a match was found, resulting in a linear search time of O(n). This means that if there are 10,000 animals it would check all 10,000. In the enhanced version, this is replaced by MongoDB queries using indexed fields. For example, instead of looping through each dog or monkey to find one that is in service and not reserved, I now send a single query to the database that includes conditions like trainingStatus: in service and reserved: false. If these fields are indexed, MongoDB can find results very quickly. MongoDB is built to handle these searches more efficiently. When these fields are indexed, the database can find results much faster, often in O(log n) time or even O(1) time for simple lookups. This means that even as the amount of data grows, the time it takes to search doesn’t grow nearly as quickly. The improvement makes the system much faster and more scalable. To implement the filtering functionality, I used a combination of checkboxes and numerical inputs that update the query parameters sent to the backend. The backend then dynamically constructs a MongoDB query object based on which filters are active. This approach avoids unnecessary filtering by ensuring that only selected criteria are included in the query.

Sorting was another improvement. In the original program, animals were listed in the order they were added. There was no way to sort by name or age. Sorting was implemented using a sortBy parameter that is passed along with the filter query. In the backend controller, I used Spring Data’s Sort object to apply sorting by name, age, or weight. I mapped each sorting option to the appropriate MongoDB field and direction. For example, selecting “Youngest to Oldest” triggers a sort by the age field in ascending order. The sorting is done on the server side so that the full dataset is already ordered before it reaches the frontend. This reduces the amount of work the Angular frontend needs to do by letting MongoDB handle sorting more efficiently before sending the results back to the user. In this web version, I added a dropdown that lets users choose how they want the list sorted. Once selected this information is passed to the backend, and the backend uses Spring Data’s built-in sort functions to return the right order.

To further improve the user experience, I added pagination and a reset button. Pagination ensures that users only see a limited number of animals per page, which keeps the application responsive and prevents slow load times. The reset button clears all filters and restores the full list, making it easier to navigate though the data even after multiple filters applied.

This enhancement meets the course outcome I initially planned to address in Module One, which focuses on solving problems using algorithmic principles and computer science practices. Through this enhancement, I demonstrated an ability to design efficient data handling logic that delivers a fast and responsive user experience while minimizing backend load. I have not made changes to the outcome I targeted, but I have developed a deeper understanding of how to apply data structure knowledge in practical scenarios. One thing I learned was the importance of maintaining clean and manageable query logic. At one point, overlapping conditions in the filter system cause unpredictable behavior. When users selected multiple filter options along with an age range and availability toggle, the backend received conflicting or incomplete parameters. This happened because the frontend sent the filters as a set of optional query strings, and my original backend logic tried to apply them all without properly validating each one first. I fixed this by restructuring how the query parameters were parsed and validated. Instead of applying all filters at once, I introduced a step-by-step check that first confirmed which parameters were actually present before building the MongoDB query. This restructuring improved accuracy and also made the filtering system easier to extend in the future.

The enhancement process helped me refine my skills in both frontend and backend logic. I learned how to coordinate between user interface events and backend processing, ensuring that all filters and sorting mechanisms stayed in sync and worked. I also deepened my understanding of Big O notation and how it relates to everyday application performance. I considered both the computational cost of data operations and the user experience impact, especially when dealing with potentially large lists.

This work supports course outcome 3, as I used algorithmic thinking and practical computing standards to solve a complex filtering and performance problem. I also met course outcome 4 by applying professional tools like MongoDB, Spring Boot, and Angular to deliver a modern and scalable application. By implementing dynamic filtering, sorting, and pagination, I demonstrated a solid understanding of how to process and optimize data in a real-world context. This enhancement not only made the application more usable for end users but also reflects my growth as a computer science professional capable of designing efficient, responsive, and user-friendly software.