Michael Isenhour

February 2, 2025

CS 499 – Computer Science Capstone

**Enhancement 2 Narrative**

**Briefly Describe the Artifact**

The artifact I chose for this enhancement is the backend of my Travlr Getaways project. Originally, I built this application using the MEAN stack (MongoDB, Express.js, Angular, and Node.js), where users could browse trips while admins could create and edit them. However, due to following required PDF tutorials each week, it lacked essential functionality like search, filtering, and pagination, making it inefficient as more trips were added. For my capstone project, I transitioned the backend to Flask and PostgreSQL, adding search functionality, filtering options, and database indexing to improve overall performance and scalability.

**Justification for Inclusion in ePortfolio**

I included this artifact in my ePortfolio because it showcases my ability to optimize database performance, improve algorithm efficiency, and enhance search functionality. The original MEAN stack applicaion had no search feature, meaning users had to scroll through every trip manually, resulting in a time complexity of O(n). With this enhancement, I implemented search, filtering, pagination, and indexing, making the system much more efficient. This highlights my skills in algorithm design, query optimization, and applying data structures to real-world problems.

**Enhancement Plan and Implementation**

To improve search efficiency and usability, I implemented a search and filtering system that allows users to search for trips based on the trip’s destination, price range, and start date. In my previous MEAN application, users had to scroll through every available trip manually, which became very inefficient as the dataset grew. Since MongoDB does not support indexed text searches in the same way PostgreSQL does, every search would have required a full table scan, making the process significantly slower as more trips were added.

With my Flask and PostgreSQL implementation, my database currently contains only 10 trips, so PostgreSQL defaults to Sequential Scans (O(n)), since scanning a small number of rows is more efficient than using an index. However, as the dataset grows, PostgreSQL will automatically switch to Index Scans (O(log n)), which will drastically reduce query execution time. By futureproofing my application and proactively implementing PostgreSQL indexing, I ensured that the system is optimized for scalability and high-performance searches.

In addition, I also introduced pagination, which limits the number of trips displayed per page to 9, preventing database slowdowns and improving overall responsiveness. Unlike the MEAN application, which required the entire dataset to load at once (O(n) per page view), my new Flask implementation only loads one page at a time (O(1) per page view), making trip browsing much more efficient. To confirm that PostgreSQL optimizes search queries based on the size of the dataset, I ran EXPLAIN ANALYZE queries and verified that as the data volume increases, the system will automatically transition from Sequential Scans to Index Scans, significantly improving performance.

**Meeting Course Outcomes**

Through this enhancement, I successfully met the course outcomes I set out to achieve in Module One.

**Collaborative Environments**

By implementing Flask blueprints to modularize different parts of my application I made the project easier to maintain and scale. Additionally, I ensured that the search and filtering system was well-documented so future developers can easily understand and modify the feature when needed.

**Professional Communication**

I focused on writing clear, structured code with proper commenting to explain key functionalities like search filtering, pagination, and indexing. Additionally, I used flash messages to provide real-time feedback to users when they perform searches or encounter errors, improving overall user experience and communication.

**Computing Solutions**

I enhanced the application by implementing search and filtering functionality, allowing users to find trips based on destination, budget, and start date. To further optimize performance, I integrated PostgreSQL indexing to speed up search queries and implemented pagination to limit the number of trips displayed per page. These enhancements ensure the application remains efficient and responsive, even as the dataset grows. By improving query performance while maintaining a user-friendly interface, this project demonstrates my ability to apply algorithmic principles and computer science standards to build scalable and optimized solutions.

**Innovative Techniques**

By implementing PostgreSQL indexing, searching query optimization, and pagination, I significantly improved the overall efficiency and scalability of the application. In my original MEAN application, searches required a full-table scan (O(n)), making queries slow as the dataset grew. With PostgreSQL indexing, searches now transition to Index Scans (O(log n)), drastically improving performance.

Additionally, I added pagination to ensure only 9 trips load per page (O(1)), preventing slowdowns compared to the previous O(n) per page view approach. I verified these improvements using EXPLAIN ANALYZE, confirming that as more trips are added, PostgreSQL will automatically optimize query execution. These enhancements demonstrate my ability to apply efficient data structures and algorithmic techniques to improve database performance.

**Security Mindset**

Throughout this enhancement, I followed secure coding practices by validating all user inputs and ensuring that search queries were protected from potential SQL injection attacks. Additionally, I maintained Role-Based Access Control (RBAC) so that only admins could modify or delete trip data, while regular users were limited to viewing and searching trips.

**Reflection on the Enhancement Process**

When it came to this enhancement, it was both challenging and rewarding. Before starting, I had no experience with Flask’s search capabilities, and I didn’t know much about PostgreSQL indexing or query performance optimization. Learning these concepts required hours of research and trial-and-error experimentation until I was able to successfully improve the search efficiency of my application.

One of the biggest challenges I faced during this enhancement was ensuring that PostgreSQL would transition from Sequential Scans to Index Scans as the data volume increased. While it defaults to Sequential Scans for small datasets, I wanted to confirm that it would automatically switch to Index Scans (O(log n)) for larger datasets, allowing me to improve performance without needing additional code changes.

Another aspect that required careful planning was my implementation of pagination and query validation to ensure that the application's usability and scalability remained intact. Looking back, these enhancements significantly improved search performance, filtering efficiency, and pagination, making my application much more scalable and efficient compared to the original MEAN stack version.

This enhancement has made a major impact on search efficiency and clearly demonstrates my ability to design and optimize search algorithms, improve database performance, and implement scalable solutions. Compared to my original MEAN application, which had no search, filtering, or pagination, this enhancement greatly improves usability, efficiency, and overall system performance. By transitioning to Flask and PostgreSQL, I built a future-proof search system that will scale efficiently as the dataset grows.