Milestone Four Enhancement Narrative: Final Artifact Submission

# Artifact Overview

The artifact I selected for the final ePortfolio submission is the controller logic and route configuration of the Travlr web application, which demonstrates my skills in algorithms and data structures. The application allows users to view, filter, and paginate through a list of travel trips. It was originally created during my full-stack development course at SNHU, and I have since enhanced it as part of CS 499 to reflect more advanced technical abilities.

# Justification for Inclusion

I chose this artifact because it provides a strong platform to showcase algorithmic thinking and practical applications of data manipulation within a real-world full-stack context. The original logic was limited to fetching a static list of trips. I enhanced this by implementing pagination using query parameters (`page`, `limit`), adding filtering logic (`destination`, `minPrice`, `maxPrice`), and introducing sorting capabilities (`sortBy=price` or `date`). These enhancements required thoughtful structuring of control flow, input sanitization, and array manipulation, demonstrating my grasp of efficient algorithmic practices.

# Outcome Alignment

These enhancements demonstrate substantial progress toward the CS 499 course outcome: “Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices.”

Specifically, this artifact shows my ability to manipulate arrays through filtering, sorting, and slicing (for pagination), parse and validate input using `req.query`, and implement helper logic for data sanitation and safety.

# Reflection on the Enhancement Process

Working through this enhancement solidified my understanding of how to build scalable endpoints that gracefully handle dynamic inputs. It was challenging to implement logic that remains clean and maintainable while supporting multiple query parameters. I approached this by modularizing logic (e.g., into helper functions), applying defensive programming for invalid inputs, and consistently validating numeric query parameters. These efforts ensured a robust experience both in the backend and for the end-user viewing paginated travel results.

Additionally, I updated the routing layer with documentation and modern syntax (`const` instead of `var`) and confirmed that query parameters are parsed correctly via `req.query`. While Express handles query parsing automatically, I gained a deeper understanding of middleware design, route-level validation, and response shaping for paginated content. These enhancements demonstrate my readiness to handle real-world data flow and API response logic.

# Pagination Enhancement Screenshot

The following screenshot demonstrates the logic added to the `travel` controller in order to support pagination for API results. The controller now reads `page` and `limit` from the query string, calculates a slice of the results to return, and includes pagination metadata such as `totalPages`, `currentPage`, `hasNext`, and `hasPrevious` to support client-side navigation between pages. This change improves scalability and aligns with best practices for handling large datasets in web applications.

