# Artifact 2 — CS 465 Narrative

Course: CS 465 – Full Stack Development

Artifact: MEAN Full-Stack Travel Booking App

## Brief Description (What/When)

The artifact chosen from CS 465 is a MEAN (MongoDB, Express, Angular, Node.js) single-page application for travel booking. The original project allowed a user to log in, view a list of trips, and edit existing trips. It included both a frontend (Angular SPA) and a backend (Node.js/Express API connected to MongoDB).  
  
The project was selected because it demonstrates my ability to design, build, and enhance full-stack applications, showcasing not only front-end user experience but also backend database integration and security considerations.

## Why This Artifact? (Justification)

This artifact was chosen because it demonstrates the Databases and Security program outcomes. It highlights the ability to:  
- Design and implement schemas with validation and indexing  
- Apply authentication and authorization to protect routes  
- Handle user input securely through sanitization and error handling  
- Connect frontend Angular services with backend RESTful APIs  
  
It also complements my CS 350 artifact by showing my growth in web application development and database-driven applications, contrasting the embedded systems work with scalable web systems.

## Enhancements Implemented

The artifact was enhanced by integrating database improvements from CS 340 (Databases) and applying security best practices:

1. Database Enhancements  
- Designed schemas for users, trips, and bookings with required fields, validation, and reference integrity.  
- Added indexes on common query fields (trips.slug, bookings.userId) for performance.  
- Implemented pagination for efficient trip listings.  
  
2. API Hardening  
- Added centralized error handling middleware for consistent status codes and responses.  
- Sanitized inputs before inserting/updating MongoDB documents.  
- Limited exposure of sensitive fields in API responses.  
  
3. Authentication & Security  
- Added JWT-based authentication with Angular AuthInterceptor.  
- Applied route guards to restrict access to booking and editing functions.  
- Enforced role-based access where needed (admin vs user routes).  
  
4. Frontend Improvements  
- Angular services updated to include auth tokens automatically.  
- Clear error messages displayed on failed login or unauthorized access.  
- Improved UI responsiveness with paginated results.

## Reflection (Process, Challenges, Feedback)

- What I Learned: Enhancing the artifact deepened my understanding of how careful schema design simplifies application logic and improves performance. I also learned how frontend route guards must coordinate with backend authorization for a consistent and secure user experience.  
- Challenges: The most difficult part was connecting Angular’s HttpInterceptor with the backend’s JWT validation while keeping the app functional and user-friendly. Testing across multiple scenarios (expired token, invalid token, unauthorized role) required careful debugging.  
- Feedback Incorporated: I improved documentation of API endpoints, standardized error messages, and added DTOs (data transfer objects) for cleaner communication between frontend and backend.

## Outcomes Met

This artifact demonstrates:  
- Databases: Schema design, indexing, validation, and secure query patterns.  
- Software Design & Engineering: Separation of concerns across frontend, backend, and database layers.  
- Security Mindset: Authentication, authorization, and sanitized inputs.  
- Professional Communication: Clear documentation and structured code review.

## Example Code Snippet (Enhanced API with Pagination & JWT Check)

// trips.controller.js  
router.get('/api/trips', requireAuth, async (req, res, next) => {  
 try {  
 const page = Math.max(1, parseInt(req.query.page || '1', 10));  
 const limit = Math.min(50, Math.max(1, parseInt(req.query.limit || '10', 10)));  
 const cursor = Trip.find({})  
 .sort({ startDate: 1 })  
 .skip((page - 1) \* limit)  
 .limit(limit);  
  
 const [items, total] = await Promise.all([  
 cursor.exec(),  
 Trip.countDocuments()  
 ]);  
  
 res.json({ items, page, pages: Math.ceil(total / limit), total });  
 } catch (err) {  
 next(err);  
 }  
});

## Closing

This enhanced MEAN travel booking app demonstrates my ability to develop and secure full-stack web applications that are scalable, maintainable, and user-friendly. It highlights growth in database design, API security, and front-end integration, fulfilling the Databases and Security program outcomes of the Computer Science degree.