Flight Booking System

Design and Implementation Report

Arpana Sripathi, Alan Soto, Bethany Green, and Prem Sathisha Etagi

Department of Science, University of Arizona

CSC 337: Web Programming

Reyan Ahmed

December 9, 2025

Introduction

For our final project, we decided to build a flight booking system that works like the airline reservation websites we've all used before. The application lets users search for flights, browse different destinations, and complete their bookings through a straightforward web interface. We built everything using Node.js, Express, and MongoDB, bringing together all the concepts we learned this semester, from client-side JavaScript to server-side processing and database management. The whole system is organized around three main parts: user accounts, destination management, and the actual booking process. These pieces work together to create what we hope is a smooth and realistic flight reservation experience.

Motivation

Choosing what to build for this project was actually pretty straightforward for us. Everyone has booked a flight at some point, so we all understood what the system needed to do. More importantly, we realized that a flight booking platform involves a lot of interesting technical challenges that would let us showcase what we've learned. It also felt like the kind of project we wouldn’t get bored with halfway through.

Another reason we picked this idea was that it's visual and interactive. We could create pages for browsing destinations, selecting from multiple flight options, and seeing confirmation details, which made the project more engaging to work on than something more abstract. The fact that airline booking systems are used by millions of people every day made this project feel relevant and gave us a concrete goal to aim for.

Modules Overview

We organized our application into three main modules:

1. **User Module:** This handles user registration and login. We hash all passwords using SHA-256 before storing them in memory. Login validation checks credentials against the in-memory user array. The user module is implemented as part of the main Express server.
2. **Destinations Module:** This manages airport data with two views: an admin page for adding new airports and a browsing page where users can search by city or country. The system inserts five default airports into MongoDB on the first run if the collection is empty.
3. **Bookings Module:** The booking workflow is implemented in the main Express. Users enter travel details, then we generate five random flight options. When someone selects a flight, we save everything to a text file and show a confirmation page using URL parameters to pass the data.

Functionalities

Our flight booking system includes several key features:

1. **Flight Search and Selection:** Users enter their names, travel date, and cities on a form. They then get five flight options to choose from, each with its own flight number, route, and randomized price. It demonstrates how the workflow would operate in a real system.
2. **Destination Management:** We built separate interfaces depending on user needs. There's an admin page for adding new airports with three-letter codes, and a browsing page where anyone can search destinations by city or country.

3. **Booking Confirmation:** Once someone selects a flight, we capture everything and append it to a text file. The confirmation page shows a complete summary.
4. **User Authentication:** We hash all passwords using SHA-256 and store them in memory.
5. **Navigation System:** Every page has the same navigation bar with links to all main sections, making it easy to understand how the site is organized and move around.

Technical Details

We used Node.js with Express running on port 8080. MongoDB stores destination data in flightDB and users in userDB. Bookings are appended to a text file using Node's fs module.

We track logged-in users through a server-side session list and client-side localStorage, passing usernames through URLs. For the frontend, we used JavaScript with DOM manipulation to dynamically create flight options and URL parameters to pass data between pages.

Passwords get hashed with SHA-256 before storage, and we use encodeURIComponent() to prevent injection attacks in URLs. The whole system follows server-side rendering, forms trigger Express routes that interact with MongoDB or files, and return HTML.