

Travlr Getaways Full Stack Travel Booking App

# **CS 465 Project Software Design Document**

Version 1.0

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc36198462)

[Table of Contents 2](#_Toc36198463)

[Document Revision History 2](#_Toc36198464)

[Instructions 2](#_Toc36198465)

[Executive Summary 3](#_Toc36198466)

[Design Constraints 3](#_Toc36198467)

[System Architecture View 3](#_Toc36198468)

[Component Diagram 3](#_Toc36198469)

[Sequence Diagram 4](#_Toc36198470)

[Class Diagram 4](#_Toc36198471)

[API Endpoints 4](#_Toc36198472)

[The User Interface 4](#_Toc36198473)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 08/17/25 | Jordan Swe | Initial version: all <> fields filled, diagrams inserted, API documented. |

## Instructions

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_heading=h.35nkun2)

This project delivers a production-style full stack travel booking application for Travlr Getaways, implemented using the MEAN stack: MongoDB for document storage, Express and Node.js for the REST API and server-rendered public pages, and Angular for the administrator single-page application (SPA). The public customer site is an Express application using Handlebars (HBS) to render trips, details, and itineraries dynamically from JSON and database data. The admin experience is a secured Angular SPA that consumes the same REST API to create, update, and delete trips and manage users. Authentication is a single layer of security using username/password login that exchanges credentials for a signed JSON Web Token (JWT). Subsequent API requests from the SPA include the JWT in the Authorization header to access protected endpoints.

## [Design Constraints](#_heading=h.1ksv4uv)

• Technology constraints: MEAN stack (MongoDB, Express, Angular, Node.js). No relational database; server framework is Express only.  
• Runtime: Local development with Node.js LTS and MongoDB Community; API served on http://localhost:3000. Angular CLI used for the SPA.  
• Security scope: Single-layer authentication (JWT) without MFA; protect administrative UI and mutating API routes.  
• Data shape: Trip schema {code, name, length, start, resort, perPerson, image, description}; basic User schema for auth. Pricing retained as string to match seed data from the course guide.  
• UX: Public site follows provided wireframe; SPA exposes list/create/edit/delete for trips with form validation.  
• Time and assessment: Delivered in course milestones emphasizing demonstrability and correctness over breadth.  
Implications: These constraints favor clear separation of concerns: server-rendered public pages for simplicity/SEO and a token-secured SPA for admin operations. Fixed schemas and local runtime simplify seeding, testing, and grading.

## [System Architecture View](#_heading=h.44sinio)

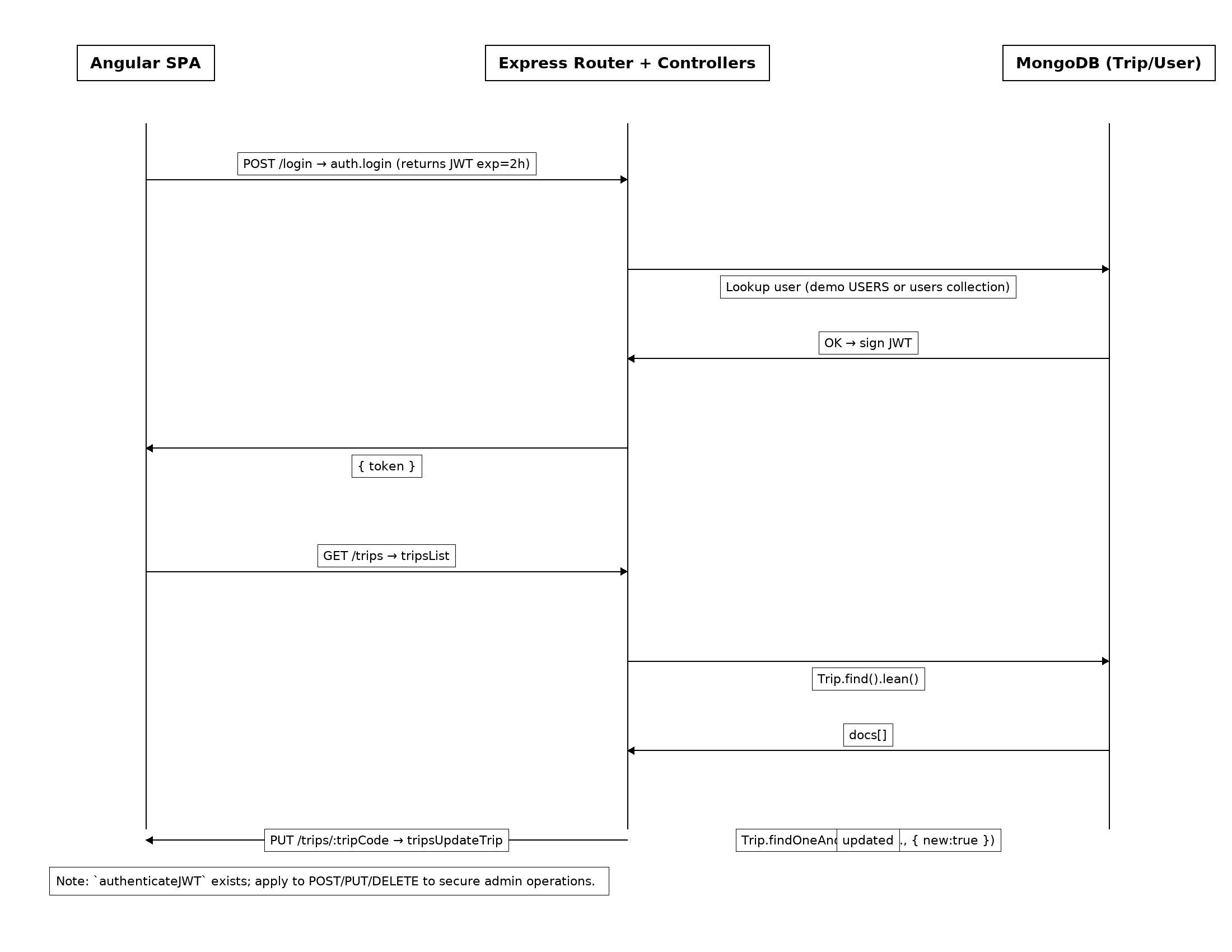
### Component Diagram



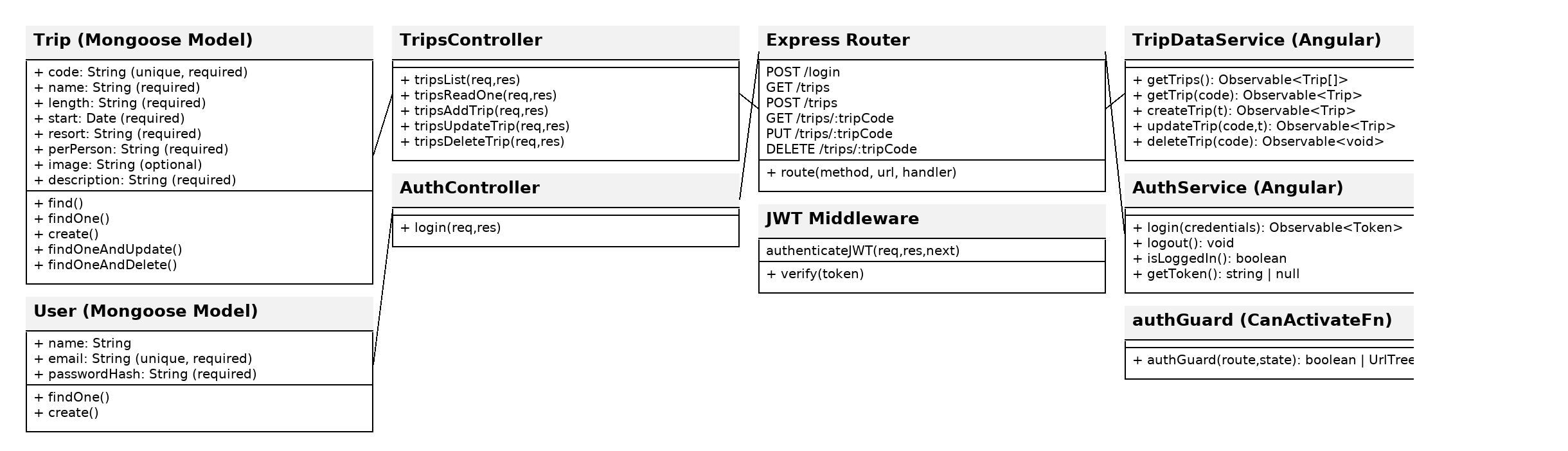
A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

The system uses a layered architecture:  
• Client Layer — (1) Public customer site: Express + HBS views rendered on the server; (2) Admin SPA: Angular app loaded in the browser.  
• API Layer — Express app\_api exposes REST endpoints under /api (e.g., /api/trips, /api/trips/:code, /api/login). Controllers validate input, invoke data access, and return JSON.  
• Data Access Layer — Mongoose models (Trip, User) implement schema, validation, and CRUD.  
• Database — MongoDB stores collections (trips, users). Seed scripts populate initial data for testing.  
Relationships: The public site uses app\_server controllers to fetch data from the API and render HBS templates. The Angular SPA calls the same API via HttpClient. JWT middleware guards admin routes. This aligns with the wireframe (public pages: Landing/Travel/Login; admin: Travel CRUD, Users).

### Sequence Diagram



## Class Diagram



| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | List all trips | /api/trips | Returns Trip[]; public read. |
| **GET** | Get single trip by code | /api/trips/:tripCode | Returns Trip; 404 if not found. |

## The User Interface

Code: Angular Trip service (app\_admin/src/app/services/trip-data.service.ts)

Code: Angular Auth Guard (app\_admin/src/app/guards/auth.guard.ts)

import { CanActivateFn, Router } from '@angular/router';  
import { inject } from '@angular/core';  
import { AuthService } from '../services/auth.service';  
  
export const authGuard: CanActivateFn = (route, state) => {  
 const auth = inject(AuthService);  
 const router = inject(Router);  
 if (auth.isLoggedIn()) return true;  
 router.navigate(['/login'], { queryParams: { redirectTo: state.url } });  
 return false;  
};

createTrip(trip: Partial<Trip>): Observable<Trip> {  
 return this.http.post<Trip>(this.baseUrl, trip);  
}  
  
updateTrip(code: string, changes: Partial<Trip>): Observable<Trip> {  
 return this.http.put<Trip>(`${this.baseUrl}/${code}`, changes);  
}  
  
deleteTrip(code: string): Observable<void> {  
 return this.http.delete<void>(`${this.baseUrl}/${code}`);  
}

Screenshots (insert where indicated):  
• Unique Trip Added: e.g., BRR1 – Blue Ridge Retreat (new trip you added)  
• Edit Screen: TripEditComponent with reactive form validation  
• Update Screen: refreshed list with confirmation message

Angular vs. Express Structure:  
• Express (public): MVC in app\_server (routes/, controllers/, views/partials/ with HBS) and app\_api for REST controllers.  
• Angular (admin): CLI layout under src/app with components/, services/, models/, routing module, and reactive forms.  
  
SPA Advantages vs. simple web pages:  
• Richer UX (client routing, validation), fewer full-page reloads, shared state. Trade-offs: larger initial bundle and explicit client auth.  
  
Testing SPA ↔ API:  
1) Postman: test GET /api/trips, GET /api/trips/NOR1 (valid), negative cases, and /api/login auth (positive/negative).  
2) Angular HttpClient: verify TripService GET/POST/PUT, inspect network calls in DevTools, confirm guards block unauthenticated access.  
3) MongoDB Compass: confirm create/update/delete persist in trips collection.  
4) Error handling: 404/empty states show friendly messages in HBS pages and SPA.