

Traveler

# **CS 465 Project Software Design Document**

Version 2.0

## Table of Contents

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

[Table of Contents](#bookmark1) 2

[Document Revision History](#bookmark2) 2

[Instructions](#bookmark3) 2

[Executive Summary](#bookmark4) 3

[Design Constraints](#bookmark5) 3

[System Architecture View](#bookmark6) 3

[Component Diagram](#bookmark7) 3

[Sequence Diagram](#bookmark8) 4

[Class Diagram](#bookmark9) 4

[API Endpoints](#bookmark10) 4

[The User Interface](#bookmark11) 4

## [Document Revision History](#bookmark12)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 08/10/23 | Nick Schiffman | Final Copy |

## [Executive Summary](#bookmark13)

This web application is built using the MEAN stack (MongoDB, Express, Angular, Node) during development. Angular would be used for the customer-facing side, providing a dynamic and interactive user interface. Express would handle RESTful APIs for communication between the front-end and back-end. Node would serve as the server-side runtime environment, ensuring scalability and high performance. MongoDB would be used as the NoSQL database for efficient data storage for user data. The administrator single-page application would follow a similar architecture, with Angular used for the interface and Express and Node for handling API requests and server-side applications.

## [Design Constraints](#bookmark14)

Design constraints for the Travlr Getaways web application impact development with cross-platform compatibility that requires thorough testing. Performance and scalability involve code optimization and caching(not reading JSON for each request). Security measures to protect user data. Usability needs user research. Integration with APIs requires planning and unknown error handling. Addressing these constraints will ensure a reliable application. The MEAN stack offers a versatile and flexible environment for building web applications; however, its learning curve and full stack complexity might be challenging for developers new to its components, and considerations like performance optimization and security practices are essential for successful development.

## [System Architecture View](#bookmark15)

### Component Diagram



The web application's overall system architecture comprises several significant components that interact with each other. The Client component serves as the customer-facing side of the application. The Server handles client requests and interacts as the middleman between the Database for the users and company data. The Database component serves as the storage for all company and user data, keeping it well organized for the server’s requests. Together, they form a cohesive system that enables quick and efficient development that enhances the user experience.

### Sequence DiagramCopy of Sequence Diagram_Travlr Getaways w_o Sequence Labels.png

<Describe the flow of logic in the web application based on the sequence diagram. Be sure to describe the interactions between the layers, or tiers, of the full-stack application. It will be helpful to include significant processes such as Sign In, Trips, and Admin interactions when referring to the sequence diagram.>

The Flow of logic starts when the user requests for the webpage, this then gets translated into an HTTP request then the browser handles the request and rates it to the server the server looks up the controller then routes to the database, then this request moves back up the chain in the reverse order and the browser allow the HTTP client and controller to interact with the user.

## Class DiagramCopy of Travlr Getaways Class Diagram w_ Labels.png

The JS classes are private methods that require getters to retrieve the data. These classes are split into segmented categories for easy expansion and allow the system to only call classes that are specific to the user's current trip. Mitigating overhead to allow for a better user experience.

## [API](#bookmark16) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| CREATE | <add a thing> | </api/things/:thingId> | <add thing to DB> |
| DELETE | <Delete a thing> | </api/things/:thingId> | <delete a thing from DB> |
| UPDATE | <Update a thing> | </api/things/:thingId> | <update a thing in the DB> |

## The User Interface

Angular projects are structured for organizations with components, services, and modules, promoting a modular approach to building the front end. In contrast, Express follows a middleware-driven architecture, focusing on routing and data handling on the backend. Single Page Applications in Angular can offer more dynamic content and create updates without full page reloads, delivering a seamless user experience while also using less server overhead. Testing the SPA's interaction with the API needed additional testing, to ensure data communication between Angular and Express is correct and functional.