

Travlr Getaways

# **CS 465 Project Software Design Document**

Version 3.0

Table of Contents

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

[Table of Contents 2](#_30j0zll)

[Document Revision History 2](#_1fob9te)

[Instructions 2](#_3znysh7)

[Executive Summary 3](#_2et92p0)

[Design Constraints 3](#_tyjcwt)

[System Architecture View 3](#_3dy6vkm)

[Component Diagram](#_1t3h5sf) 4

[Sequence Diagram](#_17dp8vu) 5

[Class Diagram](#_3rdcrjn) 6

[API Endpoints](#_26in1rg) 6

[The User Interface](#_1ksv4uv) 7-8

## [Document Revision History](#_44sinio)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/19/21 | Abigail Mattingly | Plan and discussing constraints found so far and how to approach them. |
| 2.0 | 10/03/2021 | Abigail Mattingly | Add flow diagrams |
| 3.0 | 10/24/2021 | Abigail Mattingly | To discuss the finished product |

## 

## [Executive Summary](#_2jxsxqh)

Travlr GetAway is a website used to provide access to help customers find and book travel packages. The customers must be able to create an account, search for travel packages by location and price point, and book reservations with our travel agency. The website will be built using the MEAN stack which includes essential tools such as MongoDB, Express, Angular, and Node.js. These allow the applications to link together to complete different aspects of the website. There will also be an admin only site where the administrator can log in to make updates to the site's content. The customer-facing side of the application will be programmed using html, CSS, and javascript and will be the main part of the website where everything needs to be coded. The admin side will be a single page to allow them to make updates to pricing, availability, and client needs.

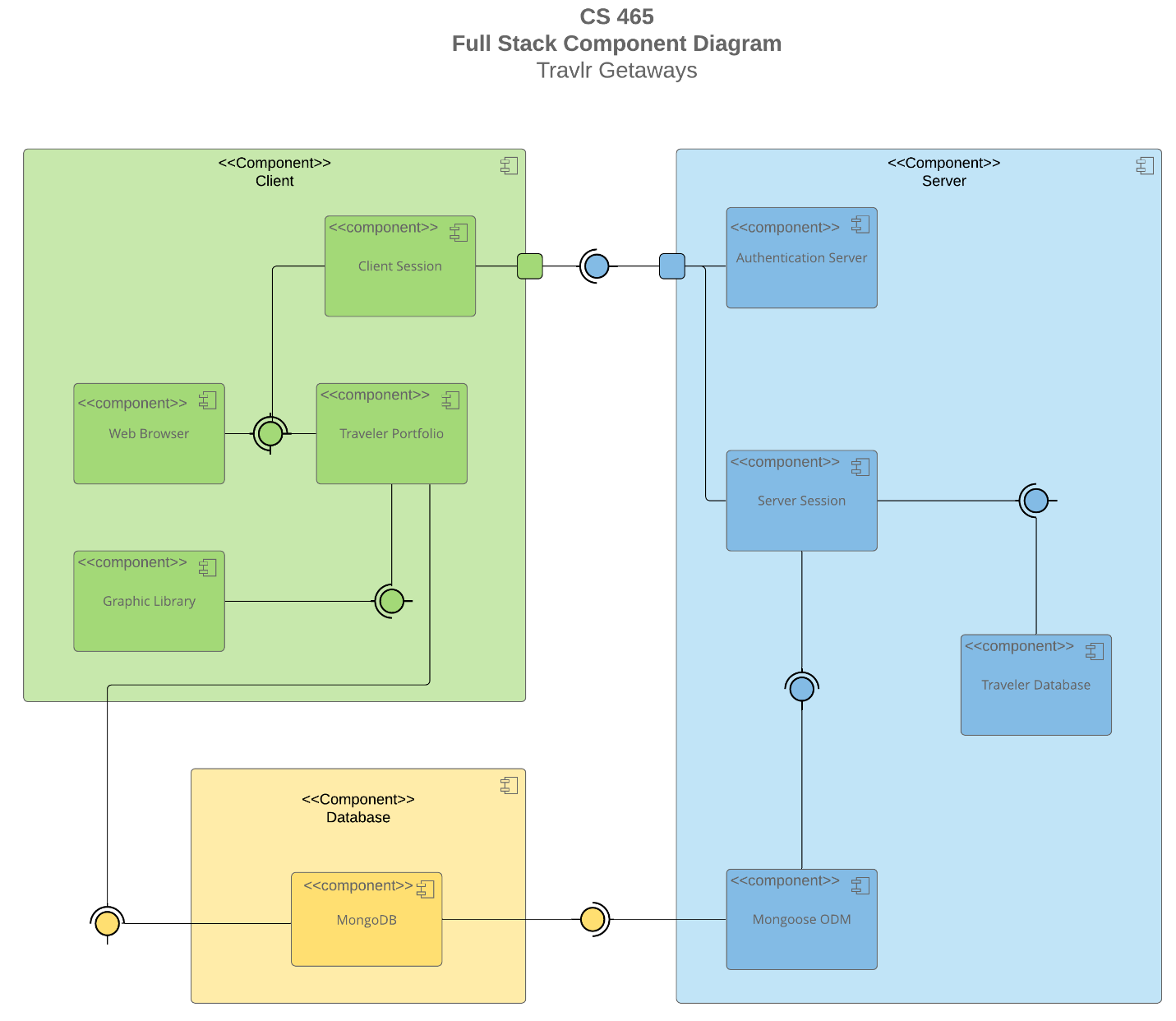
## [Design Constraints](#_z337ya)

Constraints that have been noticed so far in the project is the need for an admin side single page. When designing the customer side the developer will need to think about administrator interface, ller search criteria, and collecting login identification that will come later when programming.

## [System Architecture View](#_3j2qqm3)

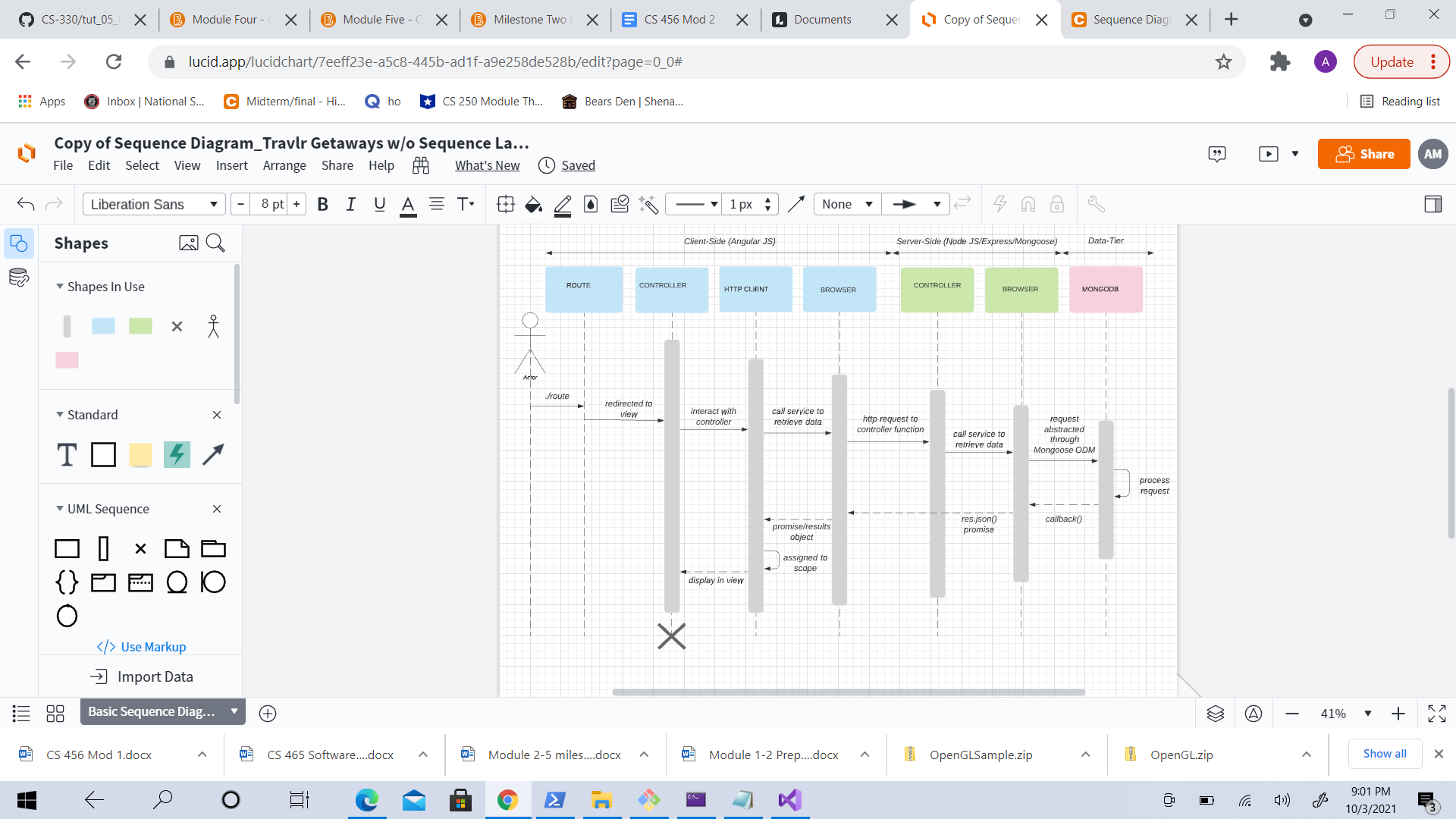
The website will open up to the customer view and access the web browser to show the traveler portfolio using what is stored in the graphic library and the database in MongoDB. While the client session is opening on the browser the server is authenticating the user and creating their session within the database as well. It is important that all of these tie together because they use aspects of different components to bring together a working website.

### Component Diagram



The Component diagram shows the databases and how they connect between the serves to create the website. The Client creates the session and it is authenticated by the server. The server creates a session while the client receives a web browser and traveler portfolio and graphics on the website. In the background the server is accessing the database to pull the information such as the trip info to present to the client.

### Sequence Diagram



The sequence diagram above shows the flow of the web application and how it interacts between the layers of the full stack information. It will start with the route and then the controller will take over and display the sign in page. The user will interact with the controller and trigger the http client that will call service to retrieve the data. MongoDB will be processing the request service and retrieving the data through mongoose. It will callback and display the results for the user to view.

## Class Diagram

## 

The classes above show a complicated but organized diagram about how all the classes interact with each other and store and gather information between them. For example the itinerary pulls information from every class to create a day by day for the user's trip such as accessing flight, booking, and cruise details from their individual booking and info classes. All of these classes work together to help a customer login, find and book a trip, and build an itinerary of the trip. As shown the trip can be either a hotel or a cruise and it can have flight if necessary.

## [API](#_1y810tw) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | code  name  length  start  resort  image  description  perPerson | </api/>  trips | Returns all active instances |
| **GET** | <Retrieve single code > | </api/trips/:tripsId> | Returns single instance, identified by the thing ID passed on the request URL |

## 

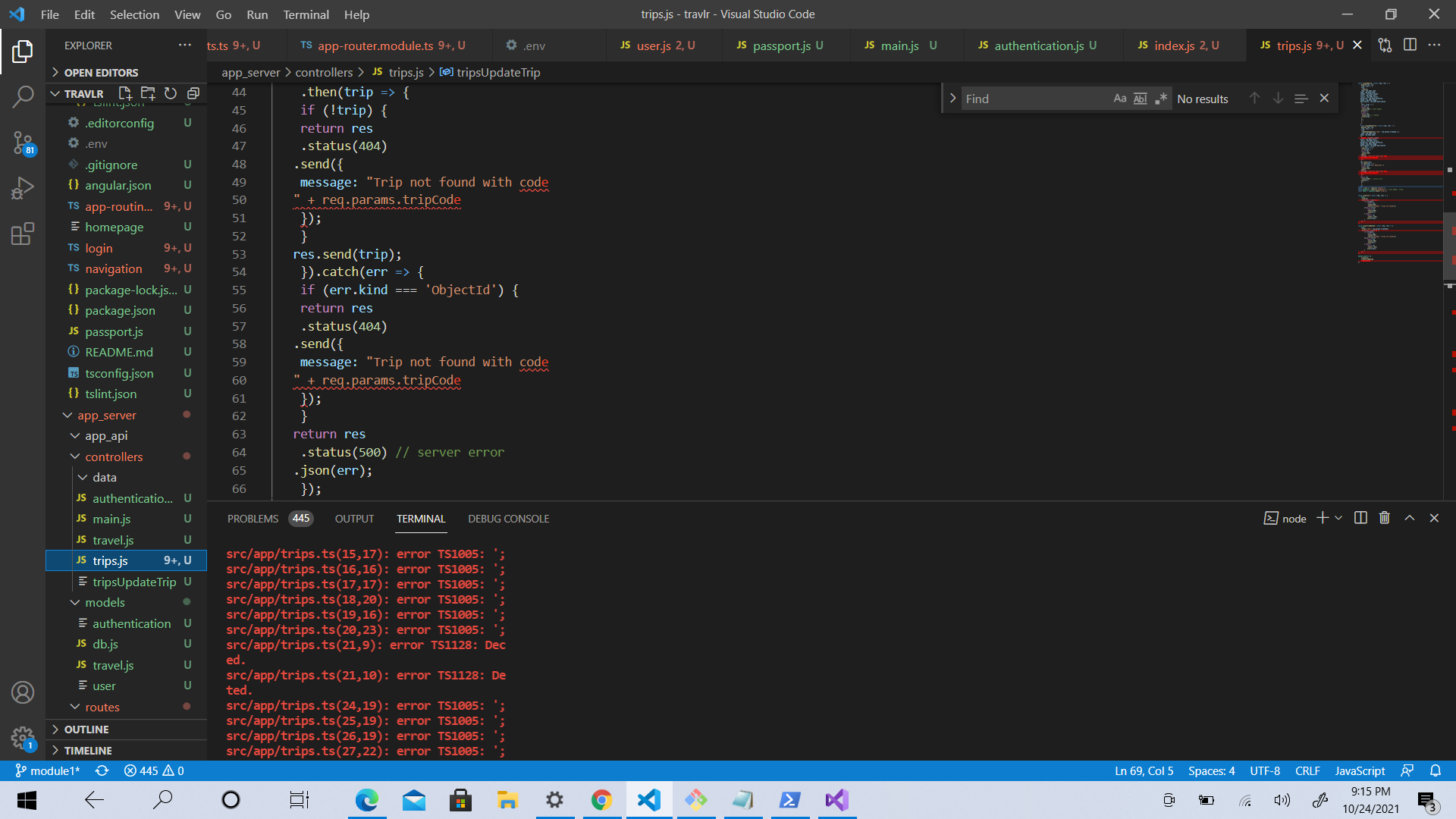
## 

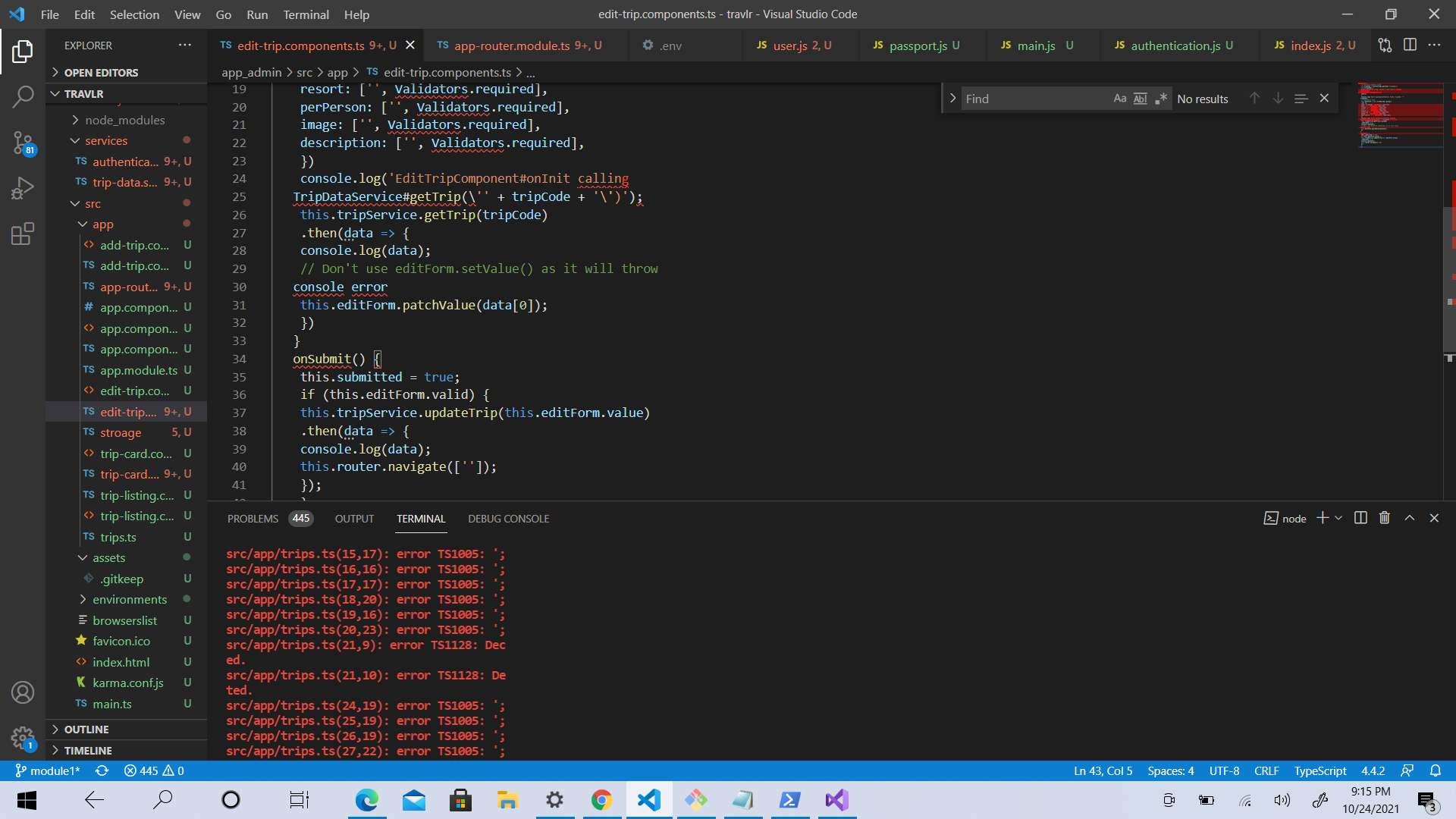
## 

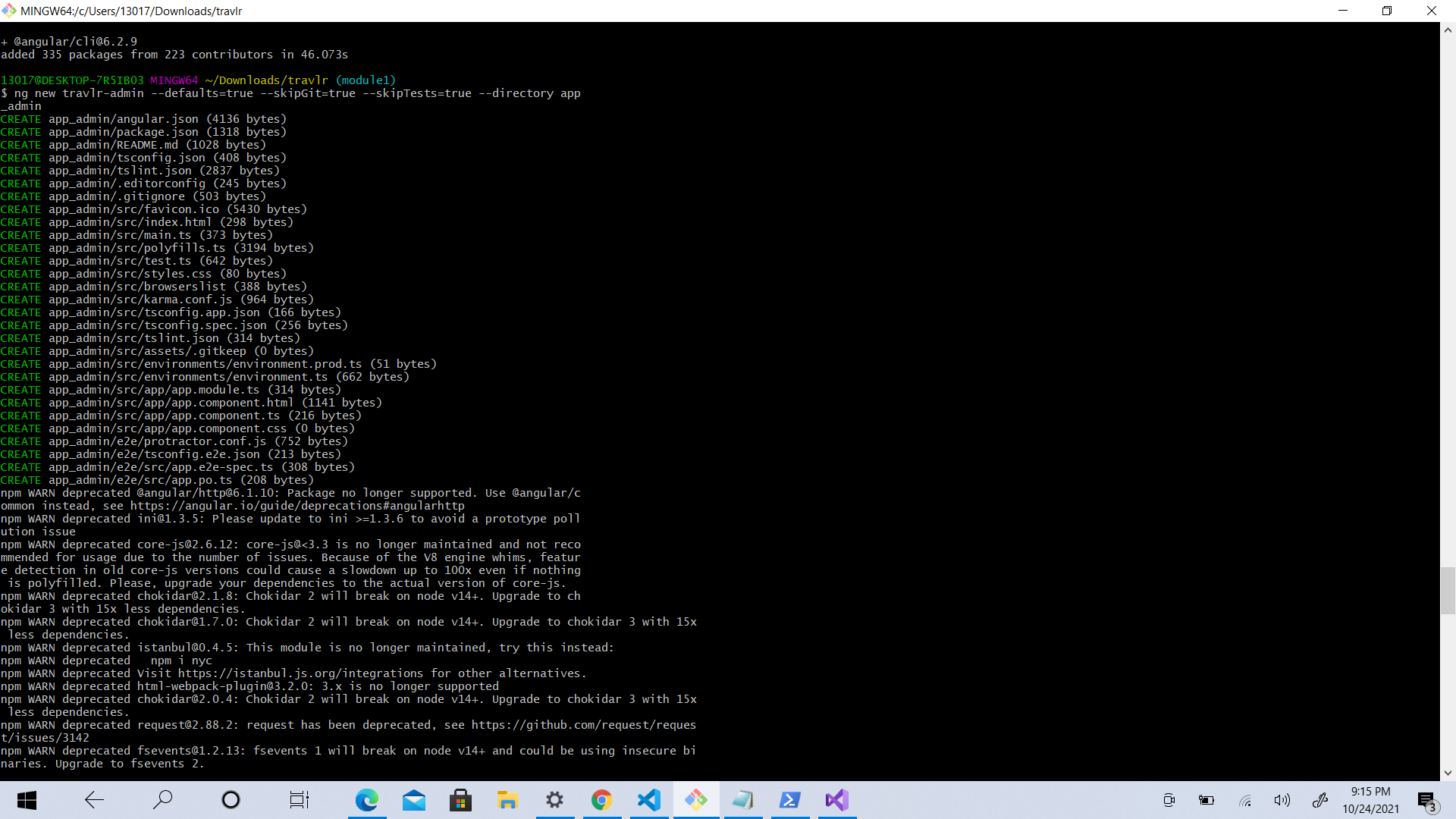
## 

## 

## The User Interface







Angular uses HTML to define the website's user interface. HTML is more intuitive and less convoluted than defining the interface procedurally in JavaScript. HTML is less likely to break. Plus you can bring in many more UI developers when the view is written in HTML. HTML is also used to determine the execution of the website. Special attributes in the HTML determine which controllers to use for each element. Angular is beneficial because instead of a specific program flow and the load process, you define the Angular dependencies.

In order to test it I used a local host to see if it would execute. I also moved files around to make sure they were in the correct folder which affects a lot. Without proper folders and organized code the program can have errors.