SSJKV
Terminal
Group 2

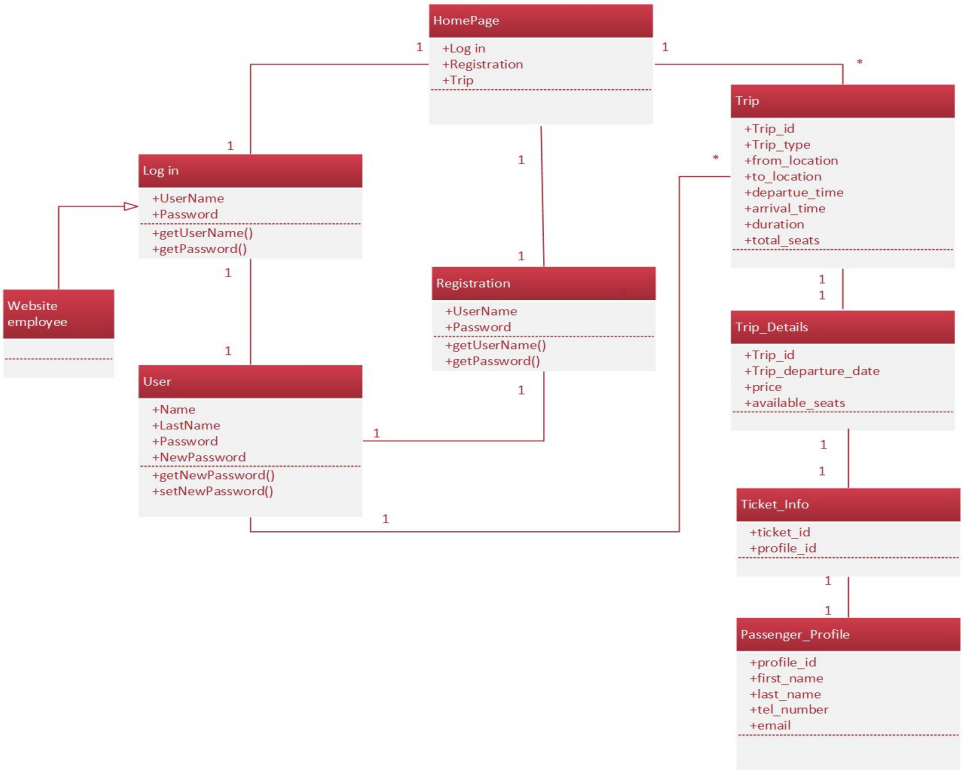
Kamal Rimal, Sarah Swilley, Saleh Alhassan, Juan Martinez, Viktoriya Rasuli

Problem statement

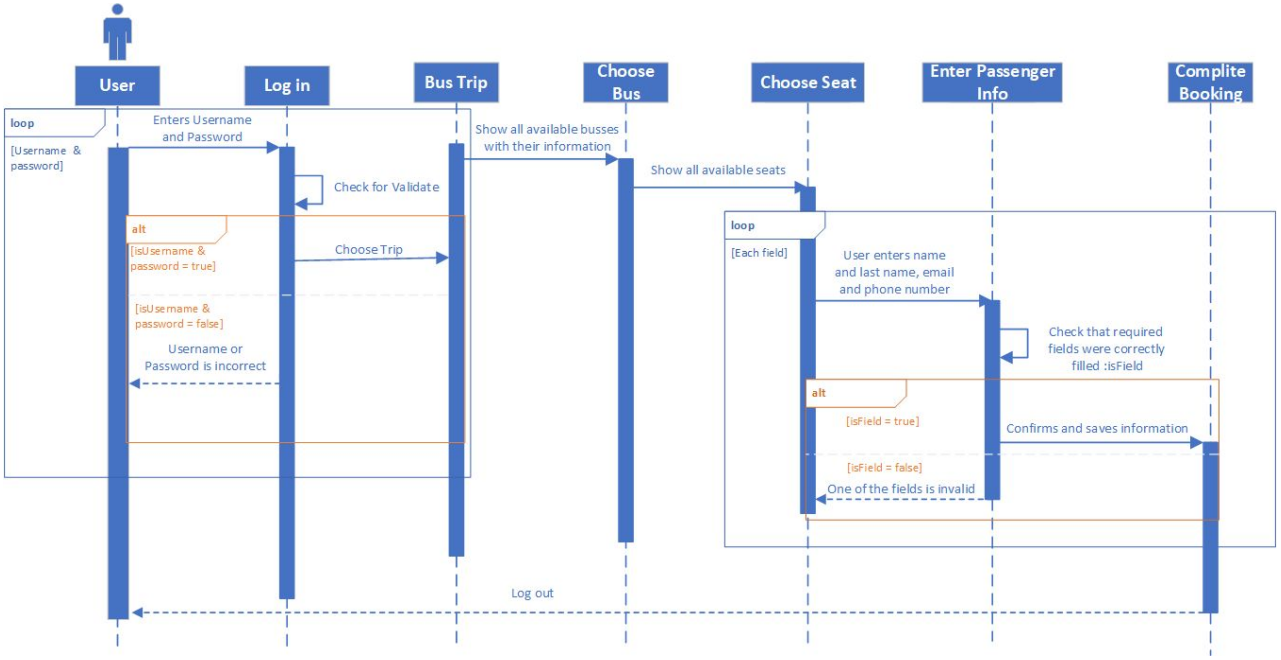
The application we developed is called the ticketing system. It takes the user's travels dates, they are taken to a system where the user can choose from a flight, bus, or train and see the different travel options regarding price, time departure, and a number of people traveling. We decided to work on this particular application because it was something that, to our knowledge, hadn't been done exactly like this quite yet, so in a sense, it was new, however after further research, we did find that it had been done before. There are a couple of applications on the market providing similar services, they are liligo, wanderu, fromatob, combtrip. There aren't many differences between our website and the others. One would be that you can choose whether you take a flight, bus, or train, and it will direct you to a new webpage that draws from a database of all of the listed flights, buses, or trains.

Architecture

Class diagram:



Sequence diagram:



Front-end:

AngularJS and Bootstrap 3 CSS framework were the main front end design frameworks used in building the website. The main reason was the ease of use of the frameworks in building stylistic modern websites, coupled with previous experience building websites using those frameworks. The website has a total of 10 pages as follows: Home, Sign In, Register, Bus, Train, Flight, Profile, Change Password, Enter Info, and Ticket Confirmation. The home page contains information pertaining to the purpose of the website and links a user to the three trip pages (bus, train, and flight) in the middle, as well as the sign in and register pages on the top right of the page. If a user clicks the sign in link, they are sent to the sign in page where they can sign in. There are two links to the registration page on the sign in page: at the top right and below the sign in submit button. If the user happens to not have previously registered, they can be navigated to the registration page by clicking on one of the links mentioned. Once a user either signs in or registers, they would be sent to the home page from where they can select one of the trip pages or the profile page. The profile page contains information the customer entered upon registering. A user can navigate to the change password page to change their password by clicking on the link to the page on the left. If the bus page is clicked in the home page, the user is navigated to a page where they can view a list of available buses that can be booked. If a user clicks on a trip to book, they will be sent to the enter info page where they can select the bus seat and enter passenger information that would be on the ticket. When the information is submitted, the user is presented with the ticket confirmation page and can choose to print the ticket or send the ticket to their email. Selecting either option will close the confirmation page and send the user to the home page. From there a user can select any of the other trip pages (train or flight) and go through the same process gone through with the bus page.

Back- end:

Node.js, Express.js, MongoDB, and Mongoose were used as the primary frameworks/platforms used to design the backend. These frameworks were chosen mainly because they are lightweight platforms that are not particularly resource heavy, as well as

previous experience using the frameworks. As we used a NoSQL Database in MongoDB, naturally our database schema would not include any tables as there is no such structure in a NoSQL database. Instead, there are data collections which are comprised of multiple documents. There are two main data collections: user and trip collections. The user collection has user documents with the following data attributes and type pair: (user_id: ID type), (user_name: string), (password: string), and a nested user_tickets document having: (ticket_id: ID type), (trip_id: ID type). The trip documents' pair is: (trip_id: ID type), (trip_type: string), (location_to: string), (location_from: string), (day: string), (departure_time: string), (arrival time: string), (total_seats: integer), and (price: integer). The equivalent of this database structure in a MySQL database would have produced four tables: User, Trip, Ticket, and Location tables. But again, since MongoDB was used, the database structure is different. The primary keys for the user, ticket, and trip documents are user_id, ticket_id, and trip_id respectively.

API:

The frameworks that we used to build an interface between our front end and back end is Node.js, Express.js, and Mongoose. We used them over choosing other API's because we have experience using these frameworks, they are easy to use, and lightweight. The API is used to retrieve, post, edit, and delete the user and trip data. The API allows for getting all users or a single user data; signing in and registering; editing user data, including user password and tickets; deleting the user. The API also allows for getting a trip, posting trip, editing trip information, and deleting the trip.

Testing:

The test environments we used the hardware Asus F556U laptop, operation system Windows 10 and Jasmine and Karma javascript testing frameworks. Also, no other application was running during the testing on a laptop. There were five significant bugs and errors encountered in our project. The first one was a location.path not recognized as a function. The second one the phone number field was able to accept any characters including letters. The third one ticket id not showing in the confirmation page. The fourth one the user attempts to print ticket but it not printing out. The fifth one the user attempts to send a ticket to the user email but not receiving it. The first problem we resolved by injecting $location to the main program scope. The second problem we solved by using a try-catch exception to make sure that only numbers are passing. The third problem we resolved by binding the ticket id to the website scope. The last two problems we tried to fix, but unfortunately couldn't.