Terminal Reservation – Delta, Greyhound, Amtrak
Terminal
Spring 2019
SSJKV
Group 2
Kamal Rimal, Sarah Swilley, Saleh Alhassan, Juan Martinez, Viktoriya Rasuli
3/2/2019

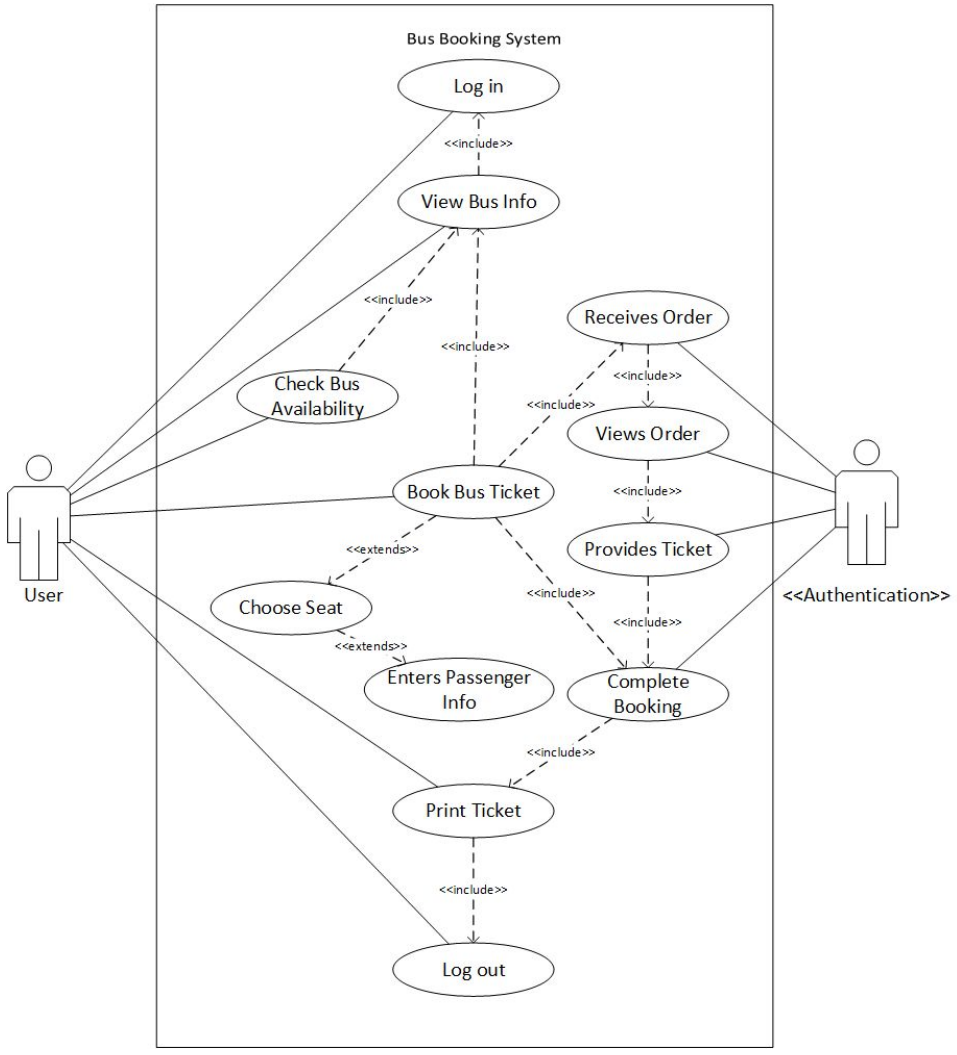
Work Breakdown Structure

Assignee Name	Email	Task	Duration (hours)	Dependency	Due date	Note
Saleh Alhassan	salhassan1@student.gsu.edu	Implement the Database Design. Fix Database specification and analysis.	7 hours	None	2/28/ 2019	100% Should be ready within 48 hours before submissions deadline. Work on the Implement the Database Design. Fix specification and analysis from the project 2.
Juan Martinez	Jmartinez41@student.gsu.edu	Fix Requirements Implement the Class Diagram Design (Frontend and Logic).	8 hours	Implement the Database Design. This should be done first.	3/1/ 2019	100% Should be ready within 24 hours before the deadline. Work with Sarah on the Implement the Class Diagram Design (Frontend and Logic). Add missing Requirements.
Sarah Swilley	sswilley1@student.gsu.edu	Implement the Class Diagram Design (Frontend and Logic). Testing.	8 hours	Implement the Database Design. This should be done first.	3/1/ 2019	100% Should be ready within 24 hours before submissions deadline. Work with Juan on the Implement the Class Diagram Design (Frontend and Logic).
Kamal Rimal	krimal1@student.gsu.edu	Do Revise and Refine System. Putting the report together. Check the Class	6 hours	All parts of the report should be ready.	3/2/ 2019	100% Report fully should be ready with 7 hours before the deadline.

		Diagrams and fix it if needed.				
Viktoriya Rasuli (coordinator)	Vrasuli1@student.gsu.edu	Creates a new To do; In Progress and Done in the GitHub as described in assignment. Behavioral modeling. Architecture modeling.	7 hours	None	2/28/ 2019	100% Behavioral and Architecture modeling should be ready within 48 hours before a deadline.

Revise and Refine your System.

In the user case diagram, we added the missing user cases. Also, showed where happened the extends between user cases.



Requirements

Requirement: Log In
Introduction: We will have a login page that will ask for users username and password.
Rationale: Make it easier to track transactions and know who is buying the tickets.
Input: User's username and password
Requirements Instructions:

1. The user must have completed registration in order to have a login.
2. The system must have the information from the username and password stored and ready to be checked

Output: Profile
Test Cases: TBD

Requirement: View bus info
Introduction: The clear and colorful page which will provide a list of specific and clear understandable information about each bus' trip, according to the information in the search gave by the user.
Rationale: A user must be able to easily find a trip to their destination by bus.
Input: Trip information such as city, date, and time.
Requirements Instructions:

1. The user must complete the information required to find a trip.
2. The system must be able to find and display the time and destination.

Output: Times and available buses to their destination.
Test Cases: TBD

Requirement: Check Bus availability
Introduction: The page must be able to have a clear way of navigation to the previous page. There should also be a link between the page and the database that pulls up the bus information.
Rationale: Make it easy for the user to know what busses are available.
Input: Information about the travel dates and the city from the previous page.
Requirements Instructions:

1. The page must receive information about the time and city.
2. The system will then go to the database, find the information that is looking for.

Output: Availability of the trips based on the city and date given.
Test Cases: TBD

Requirement: Book Bus ticket
Introduction: The user will be taken to a new page where they can confirm the bus trip they have selected with options to cancel or go back.
Rationale: To make it easy for the user to know what bus trip they have selected and clear as to what they will be getting.
Input: The bus trip from the database with time and city.
Requirements Instructions:

1. Receive information from the database.
2. Show the user the time and date selected.

Output: Date and time selected.
Test Cases: TBD

Requirement: Receives Orders
Introduction: The system will receive an order for what bus trip the user has selected
Rationale: To start the authentication process and give the user a confirmation code.
Input: The bus trip that was selected by the user.
Requirements Instructions:

1. Information selected by the user

Output: Selected bus trip.
Test Cases: TBD

Requirement: View Orders
Introduction: The system will check the database for any conflict with the information given by the user.
Rationale: To make sure that the ticket that the user has selected is available.
Input: The information selected by the user for the bus trip selected.
Requirements Instructions:

1. Compare data selected with the database and look for conflict.

Output: Ticket information.
Test Cases: TBD

Requirement: Provides Ticket
Introduction: After confirming that there is no conflict the system will but all the information together on the ticket.
Rationale: Make and deliver a ticket for the user
Input: Information received from the database that was selected by the user.
Requirements Instructions:

1. Receive information from the database.
2. Show the user the time and date selected in a ticket.

Output: Date and time selected.
Test Cases: TBD

Requirement: Choose seats
Introduction: The user will be able to select a seat from available seats on a bus.
Rationale: Organizing and knowing who is where and to keep a tally of available seats.
Input: Row number and seat letter.
Requirements Instructions:

1. Read the date given by the user for seat row and number.
2. Search the data in the database and receive the information on available seats.
3. Show the user the seat that was chosen.

Output: The seat that they have selected or tell the user there is no more seats available**.**
Test Cases: TBD

Requirement: Enter passenger info
Introduction: The user will be taken to a new page where they can give us their information and the people that they are traveling with.
Rationale: To know which person is seating in what seat and to keep the inventory of available seats
Input: Name, address, phone, email.
Requirements Instructions:

1. The user must know all the information and must submit it to continue.
2. The system will add the information and put it on the ticket.

Output: The information that was given.
Test Cases: TBD

Requirement: Complete booking
Introduction: The user will select to complete the booking and there will be a confirmation number given to them.
Rationale: Confirming that the user has selected the right bus trip and for the user to keep trap of their upcoming trip.
Input: 'Complete booking' icon.
Requirements Instructions:

1. The user will click to confirm the trip
2. The system will provide a confirmation number.

Output: Confirmation number.
Test Cases: TBD

Requirement: Print ticket
Introduction: The user will be able to print a ticket or save it on their files.
Rationale: To authenticate the transaction.
Input: The ticket info from the previous screen.
Requirements Instructions:

1. The system will give them a ticket to print out.
2. The user will be able to print or save their ticket.

Output: Ticket.
Test Cases: TBD

Requirement: Logout
Introduction: The user will be able to save and exit out of the website
Rationale: To keep track of people that are using and booking tickets
Input: Click the logout button
Requirements Instructions:

1. The user needs to be signed
2. The system will log the user out after 30 minutes of no use

Output: Back to Home screen
Test Cases: TBD

System Modeling (Analysis).

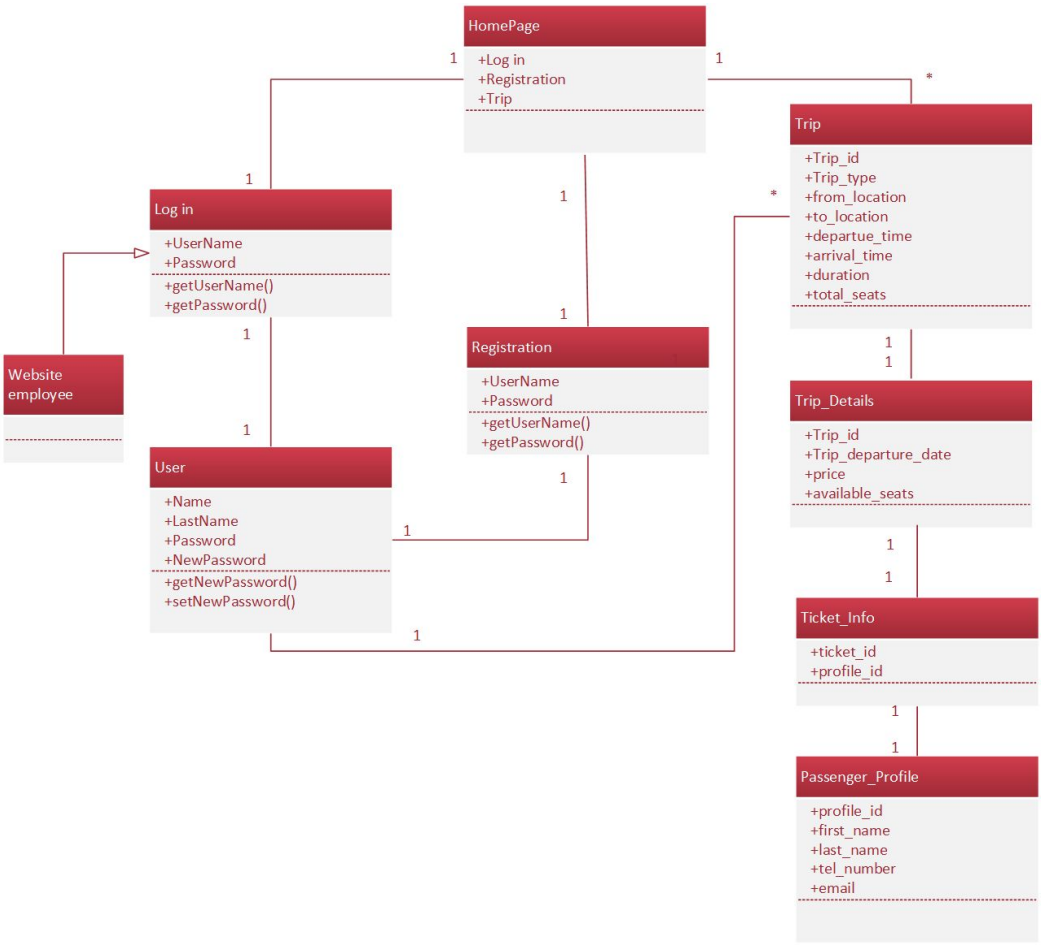
Class Diagram:

Our class diagram has nine objects such as HomePage, Log in, Trips, Registration, Trip_Details, Ticket_info, Passengare_Profile, User, Website Employee. All of the objects in our diagram have an association with other objects except for Website Employee object that has a directed association with Log in object. The multiplicity almost between every object is the same and equal to one to one. Only in two cases, we have different multiplicity that is not equal one to one, it between objects HomePage and Trips as also between objects User and Trips. In both cases, it equals one to many.

All object have several attributes in our class diagram, except one which is Website Employee, because it object playing the role of the person. The attributes for HomePage

object are log in, registration and trips. In the Log in and Registration object, attributes that we are going to use are username and password. In the User object, we will use name, lastname, password and newpassword attributes. The Trip object will have attributes such as trip_id, trip_type, from_location, to_location, departure_time, arrival_time, duration, and total_seats. In the Trip_Details object, we will have four attributes trip_id, trip_departure_data, price and available_seats. The Ticket_Info object has attributes ticket_id and profile_id. The last object is Passenger_Profile object. It will have attributes such as profile_id, first_name, last_name, tel_number and email.

We decided in our class diagram that only three objects out of nine will have operations. These three objects are Log in, User and Registration where Log in and Registration have identical operations which are getUserName() and getUserPassword(). The User object has getNewPassword() and setNewPassword().



Database Specification and Analysis:

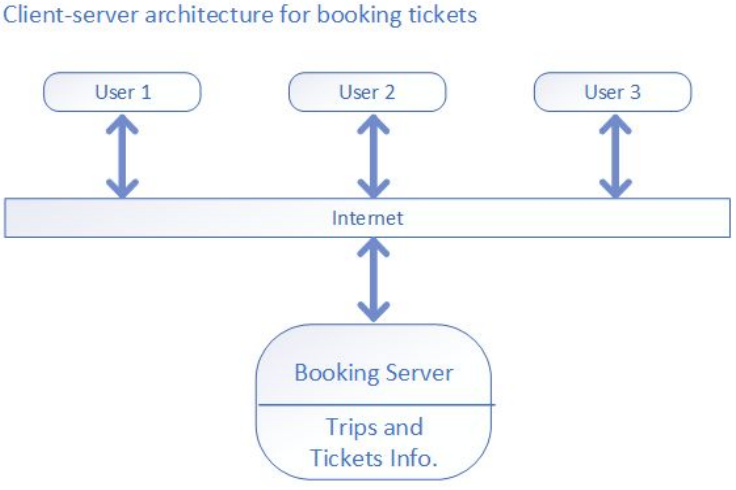
We will use a MongoDB No-SQL database, and as such, the structure of the database and terminology used will be different from a normal SQL database. This means that there will be no "tables" like there would be if we were using a MySQL database. Instead, there will be data collections which are comprised of multiple documents. We will have two main data collections: user and trip collections. The user collection will have 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 will be: (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). It is important to note that if this was a MySQL database, there would have been four tables: User, Trip, Ticket, and Location tables. But again, since we are using MongoDB, we have a different database structure. The primary keys for the user, ticket, and trip documents are user_id, ticket_id, and trip_id respectively. Below are two images illustrating the structure of the two database documents as discussed above.

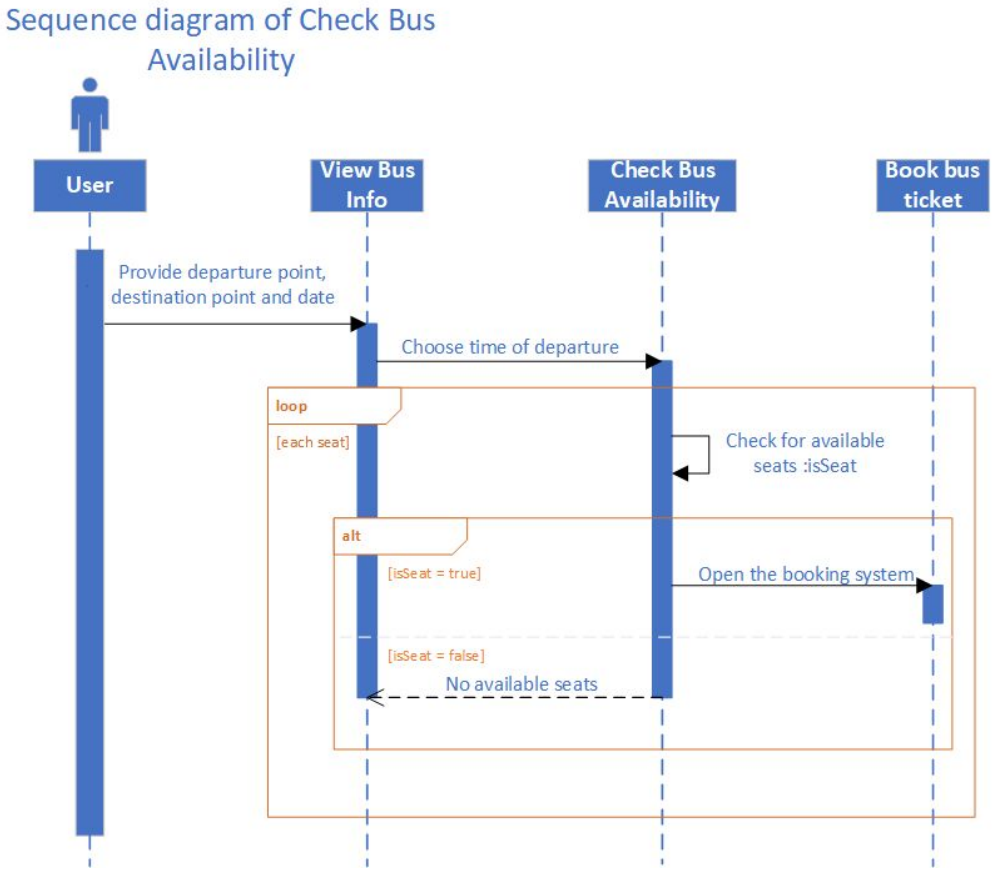

```
_id: ObjectId("5c74c5ba02932c20b0c256f8")
name: "Newer User"
password: "password"
__v: 0
tickets: Object
    ticket_id: ObjectId("5c7a0c0b7d203a0000e66ad0")
    trip_id: ObjectId("5c75c1fe22b7e431643bc2a7")

_id: ObjectId("5c75c1fe22b7e431643bc2a7")
trip_type: "Bus"
location_to: "Boston"
location_from: "Atlanta"
day: "Monday"
departure_time: "1:00 pm"
arrival_time: "6:00 pm"
total_seats: 37
price: 500
__v: 0
```

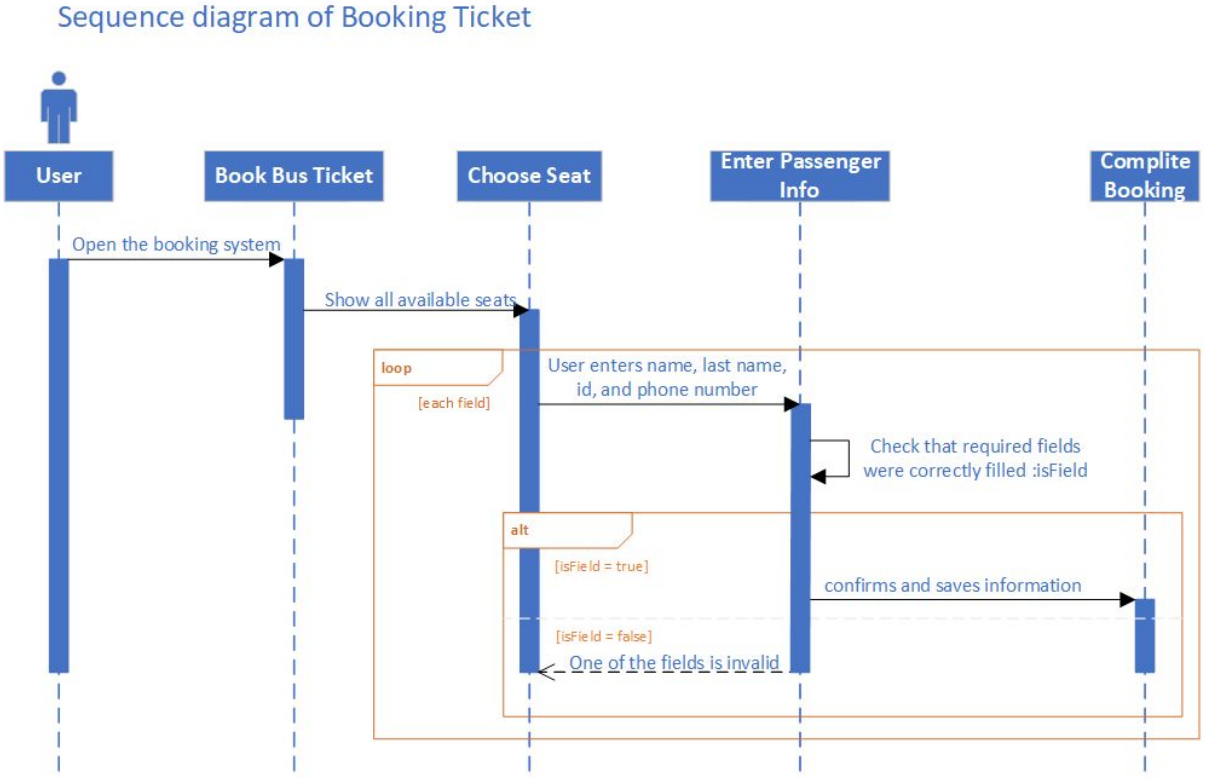

Architecture modeling:

For our project, we decided to use a client-server architecture because it has a simple architecture and it allows clients to access the website whenever a person needs it. Also, this architecture was designed to not have difficulty providing needed service to clients.



Behavior modeling:





Implementation

Implement the Database Design (Tables, Backend):

For the backend, we used JavaScript to create an API that communicates with the MongoDB database we created. As it stands, the API can connect to 10 endpoints dealing with getting, posting, patching, and deleting user and trip data. The backend currently allows for the implementation for all the use cases. Node.js was used as the run time environment to execute the developed JavaScript code.

Implement the Class Diagram Design (Frontend and Logic):

For the frontend of the project, I have chosen to use IntelliJ. Because it is an actual IDE, I have capabilities that I wouldn't with a common text editor. I have also decided to code this project using HTML, CSS, and JavaScript because all browsers support it, there is a multitude of web development tools, it's search engine friendly, and it's not overwhelmingly complicated. HTML also has the bootstrap library which helps create a more aesthetically pleasing website.

Testing

Test Case ID: 1.1

Description: Attempt login on the main page of the website.
Test Inputs: Email and password corresponding to a valid user account.
Expected Results: Login Successful. User is then presented with another page that will show the bus information.
Dependencies: None
Initialization: Users account must already have been created.
Test Steps:

1. Submit valid username and password on a login page.

Test Case ID: 1.2

Description: Attempt to login without incorrect user information.
Test Inputs: Email and password are not correct.
Expected Results: Login Unsuccessful. The user will be notified that the user name in password doesn't match.
Dependencies: None
Initialization: Database information that matches the correct user information.
Test Steps:

1. Submit invalid username and password on a login page.

Test Case ID: 2.1

Description: User attempts to book a bus ticket.
Test Inputs: Preferred date, location, and time of departure.
Expected Results: Booking of ticket successful.
Dependencies: None
Initialization: Database information for time and location are reserved within the database.
Test Steps:

1.Input the desired date, time, and location.

2.After the above are selected, the user will be presented with a "book bus ticket" link.

3.Click link.

Test Case ID: 2.2
Description: User attempts to book a bus ticket, but put in wrong information. For example, using numbers for a location.
Test Inputs: Preferred date, location, and time of departure.
Expected Results: Booking of the ticket unsuccessful.
Dependencies: None
Initialization: None
Test Steps:

1.Input the desired date, location, and time.

Test Case ID: 3.1
Description: User attempts to logout after the ticket is booked.
Test Inputs: User must be already logged in.
Expected Results: Logout successful.
Dependencies: Must be logged in. Must have a logout button.
Initialization: Database accesses user information, and then disconnects from it.
Test Steps:

1.Click the logout button at the top right-hand side of the page.

Test Case ID: 3.2
Description: User attempts to logout after the ticket is booked.
Test Inputs: User must be already logged in.
Expected Results: Logout unsuccessful.
Dependencies: Must be logged in. Must have a logout button.
Initialization: Database accesses user information, and then cannot disconnect from it.
Test Steps:

1. Click the logout button at the top right-hand side of the page.
2. Go back to the main page and try clicking the button again.

Appendix

https://github.com/5aleh/book-on-time/projects/3