# Tickets Reservation – Delta, Greyhound, Amtrak Terminal

Spring 2019

SSJKV

Group 2

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

3/14/2019

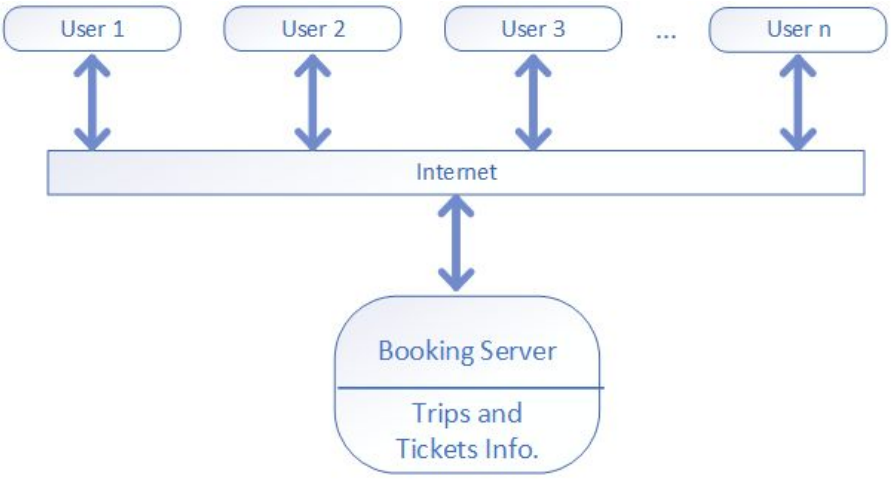
**Work Breakdown Structure**

| Assignee Name | Email | Task | Duration (hours) | Dependency | Due date | Note |
|---|---|---|---|---|---|---|
| Saleh Alhassan | salhassan1@student.gsu.edu | Implement the Database Design.<br>Implement the Test case | 8 hours | None | 2/13/2019 | 100%<br>Should be ready within 24 hours before a deadline |
| Juan Martinez | Jmartinez41@student.gsu.edu | Test Case Script.<br>Test Documentation and create new Test cases. | 6 hours | Implement The test case should be done first. | 3/13/2019 | 100%<br>Should be ready within 24 hours before a deadline<br>Work with Kamal on test case script. |
| Sarah Swilley (coordinator) | sswilley1@student.gsu.edu | Implement the Class Diagram Design (Frontend and Logic).<br>Put all parts together. | 8 hours | Implement the Database Design.<br>This should be done first.<br>All parts of the report should be ready. | 3/14/2019 | 100%<br>Implementation part should be ready with 24 hours.<br>Work with Viktoriya on Implement the Class Diagram Design.<br>The report should be ready within 7 hours before a deadline. |
| Kamal Rimal | krimal1@student.gsu.edu | Do Revise and Refine System.<br>Architecture modeling.<br>Test Case Script. | 6 hours | None | 3/12/2019 | 100%<br>Should be ready within 48 hours before a deadline.<br>Work with Juanon test case script. |
| Viktoriya Rasuli | Vrasuli1@student.gsu.edu | Creates a new To do; In Progress and Done in the GitHub as described in an assignment.<br>Implement the Class Diagram Design | 7 hours | None | 3/12/2019 | 100%<br>Behavioral should be ready within 48 hours before a deadline.<br>Work with Sarah on Implement the Class Diagram Design. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | (Frontend and Logic). Behavioral Modeling. | | | | |

**Revise and Refine your System**

In the Client-server architecture, we showed that there are multiple users can access the booking server not only three as we showed last time.




We decided to change our test case 2 which was a booking bus ticket to the change password because the front end is complete for that part.

**Test Case ID: 2.1 Change password**
**Description:** User attempts to change the password on his account.
**Test Inputs:** User enters old password and new password into assigned fields.
**Expected Results:** Change of password is successful. The user redirects to user profile page.
**Dependencies:**A user should log in first before be able to change old password.
**Initialization:** Database with information about user password and login name.
**Test Steps:**

1. Submit valid old password and then new password into the provided fields.
2. Click button save changes that locates at the center-bottom page.

**Test Case ID: 2.2 Change password**
**Description:** User attempts to change the password on his account.
**Test Inputs:** User enters invalid old password and then new password into assigned fields.
**Expected Results:** Change of password is unsuccessful. The user stays on the same page and sees a notification that the old password is not matching with entered one.
**Dependencies:** A user should log in first before be able to change old password.
**Initialization:** None
**Test Steps:**

1. Submit invalid old password and then new password into provided fields.
2. Click button save changes that locates at the center-bottom page.

## 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)

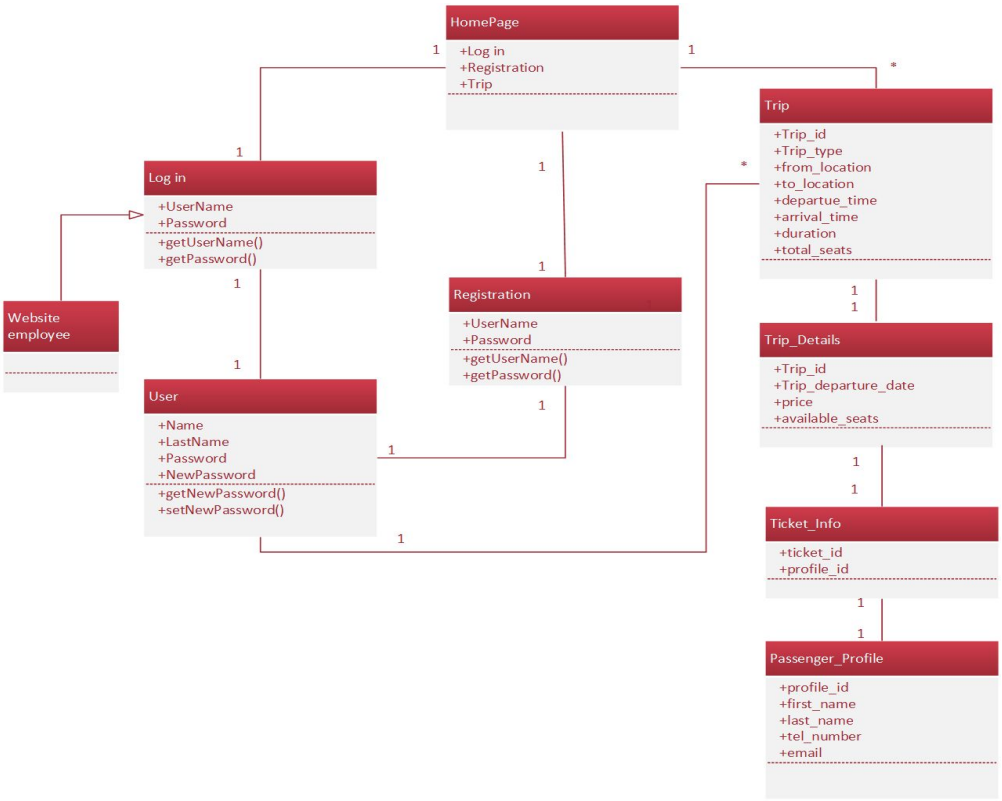
**Class Diagram:**

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:

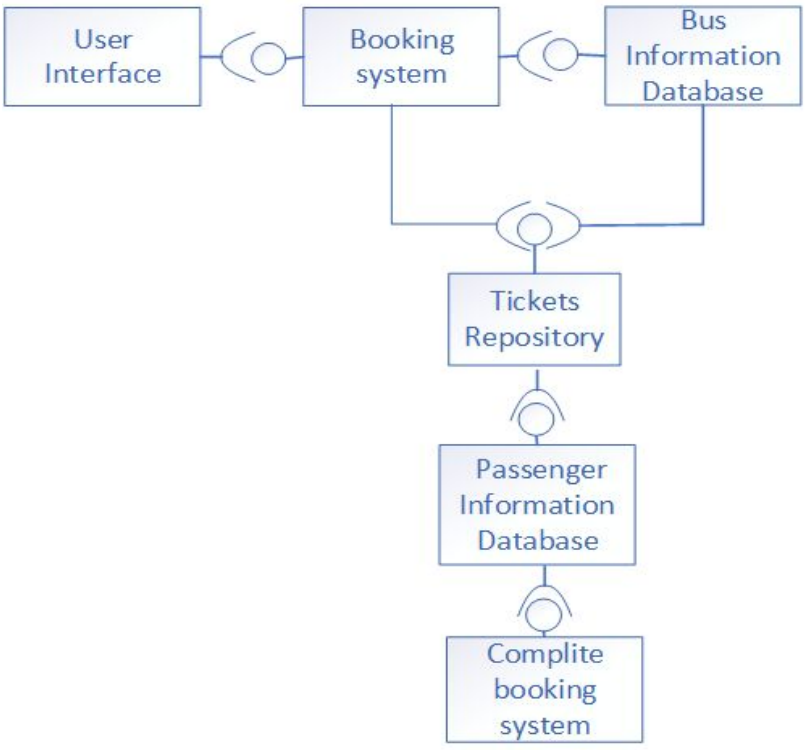
We will use a MongoDB No-SQL database, and as such, the structure of the database and terminology used will be different from a normal SQL database. This means that there will be no "tables" like there would be if we were using a MySQL database. Instead, there will be data collections which are comprised 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.

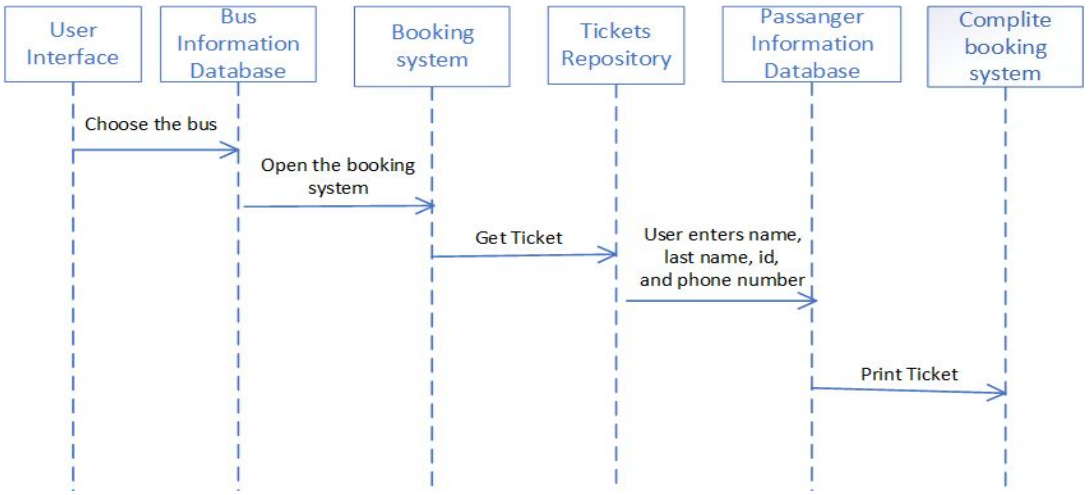

```
_id: ObjectId("5c74c5ba02932c20b0c256f8")
name: "Newer User"
password: "password"
__v: 0
tickets: Object
   ticket_id: ObjectId("5c7a0c0b7d203a0000e66ad0")
   trip_id: ObjectId("5c75c1fe22b7e431643bc2a7")
    _id: ObjectId("5c75c1fe22b7e431643bc2a7")
    trip_type: "Bus"
    location_to: "Boston"
    location_from: "Atlanta"
    day: "Monday"
    departure_time: "1:00 pm"
    arrival_time: "6:00 pm"
    total_seats: 37
    price: 500
    __v: 0
```


**Architecture modeling:**

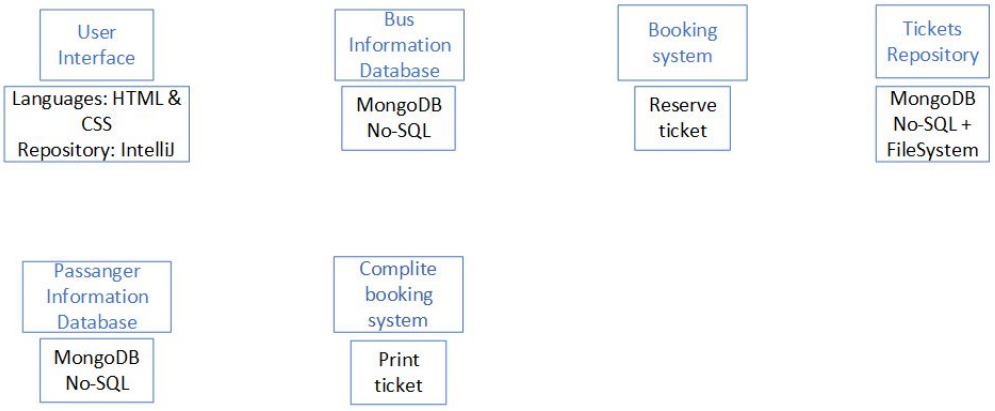
**Logical View**



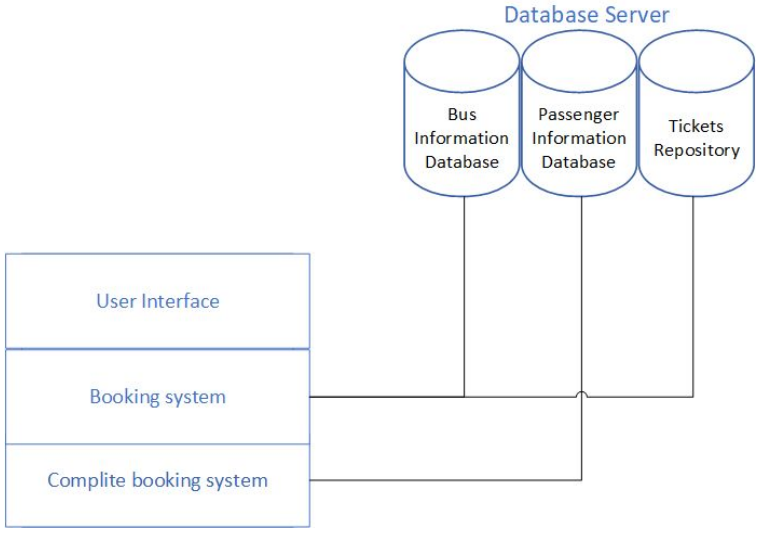
## Process View




## Development View

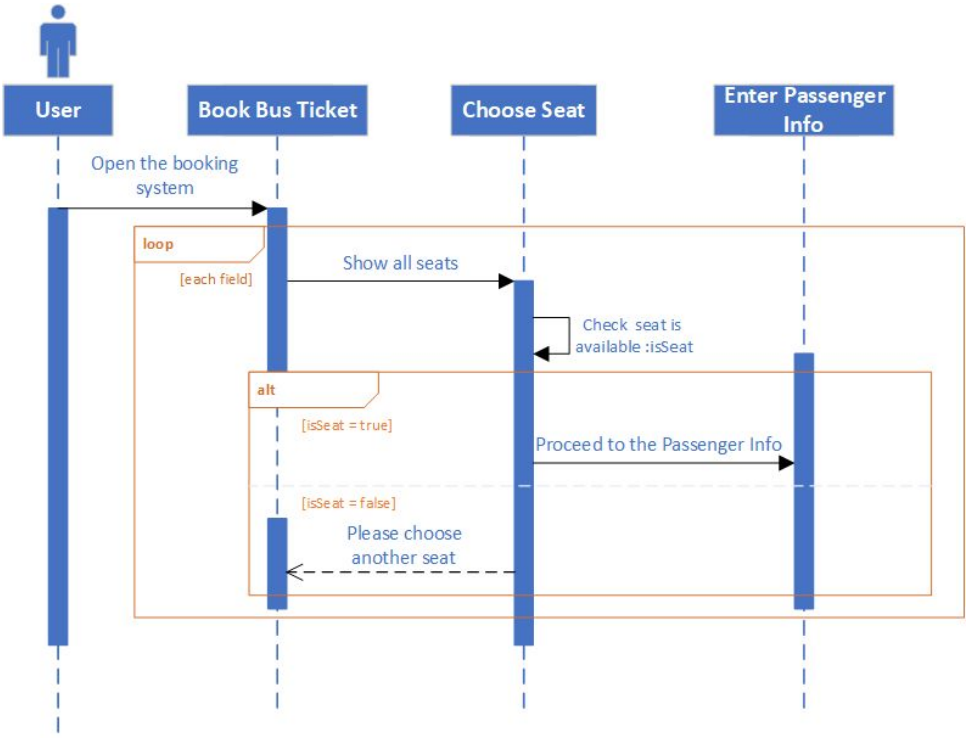



## Physical View

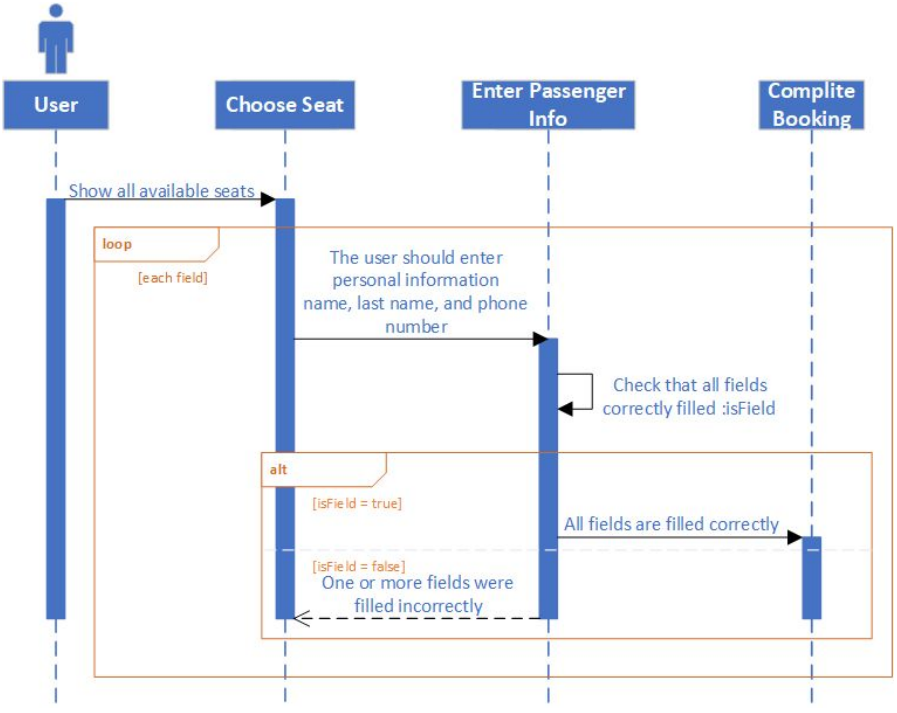



## Behavior modeling:

**Sequence diagram of Choose Seat**




**Sequence diagram of Entering Passenger Information**




## 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

**Creating Testing environment: Installed javascript testing frameworks jasmine and karma through npm install. The tests cases were written in JavaScript.**

**Test Case:**

**Test Case ID: 4.1 View bus information**
**Description:** Attempt of the user views bus information with valid information.
**Test Inputs:** Place of departure, place of arrival, date of departure and return to get the valid bus information.
**Expected Results:** Search is successful. To a user provides a dashboard with bus information.
**Dependencies:** A user should log in first before access the page with bus information.
**Initialization:** Database with information about place departure, place arrival and date of departure and return.
**Test Steps:**

1. Submit valid information into the provided fields about the place of departure, place of arrival, date of departure and return.
2. Click button find at the center-bottom page.

**Test Case ID: 4.2 Viev bus information**
**Description:** Attempt of the user views bus information with invalid information.
**Test Inputs:** Place of departure, place of arrival, date of departure and return are incorrect.
**Expected Results:** Search is unsuccessful. A user sees a notification that search was unsuccessful and ask to try again.
**Dependencies:** A user should log in first before access the page with bus information.
**Initialization:** None
**Test Steps:**

1. Submit invalid information into provided fields about the place of departure, place of arrival, date of departure and return.
2. Click button find at the center-bottom page.

**Test Case ID: 5.1 Enter Passenger Information**
**Description:** User attempts to enter valid passenger information to continue booking tickets.
**Test Inputs:** Name, last name, phone number, and email.
**Expected Results:** All fields filled successfully. A user sees a notice that a ticket is booked.

**Dependencies:** A user should log in first, choose the bus and then seat before to get on this page.
**Initialization:** None
**Test Steps:**

1. Enter the passenger information into the provided fields.
2. Click button continue at the center-bottom page.

**Test Case ID: 5.2 Enter Passenger Information**
**Description:** User attempts to enter invalid passenger information (enters the phone number in the name field or name in the phone number) to continue booking tickets.
**Test Inputs:** Name, last name, phone number, and email.
**Dependencies:** A user should log in first, choose the bus and then seat before to get on this page.
**Initialization:** Catch try system should be already implemented to check the fields.
**Test Steps:**

1. Enter the passenger information in the provided fields.
2. Click button continue at the center-bottom page.

**Implementing the Test case:**

```
describe('signinController', function() {
  beforeEach(module('myAngApp'));

  var $controller, loggedOnService;

  beforeEach(inject(function(_$controller_, _loggedOnService_) {
    $controller = _$controller_;
    loggedOnService = _loggedOnService_;
  }));

  describe('$scope.signin', function() {

    var $scope, controller, $cookies;

    beforeEach(function() {
      $scope = {};
      $cookies = {};
      controller = $controller('signinController', { $scope: $scope, $cookies: $cookies});
    });

    it('does not allow the the user to sign in', function() {
      $scope.userEmail = 'notrightemailformat';
      $scope.userPassword = 'shouldnotmatter';
      $scope.signin();
      expect($cookies['uName']).not.toBeDefined();
    });
    it('allows the the user to sign in', function() {
      $scope.userEmail = 'salhassan1@student.gsu.edu';
      $scope.userPassword = 'pass';
      $scope.signin();
      expect(loggedOnService.signedIn).toBeDefined();
    });
  });

});
```

```
describe('passwordController', function() {
  beforeEach(module('myAngApp'));

  var $controller;

  beforeEach(inject(function(_$controller_) {
    $controller = _$controller_;
  }));

  describe('$scope.chgpass', function() {

    var $scope, controller;

    beforeEach(function() {
      $scope = {};
      controller = $controller('passwordController', { $scope: $scope });
    });

    it('ensure correct current user password is entered', function() {
      $scope.userPassword = 'shouldnotmatter';
      expect($scope.userPassword).toBe('shouldnotmatter');
    });
    it('ensure the current user password is changed', function() {
      $scope.newpass = 'pass';
      expect($scope.newpass).toBe('pass');
    });
  });

});
```

Above are four tests cases relating to sign-in and change-password.

**Test Case Scrip:**

| **Test ID** | 1.1 Log in |
|---|---|
| **Purpose of Test** | Make sure that the user can log in on the website with valid information. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the testing on a laptop. |

| | |
|---|---|
| | Operation system: Windows 10 |
| **Test Steps** | In the log in page user should do:<br><br>User clicks with the mouse on the field that says Email and enters the valid Email address into the field.<br>User clicks with the mouse on the field that says Password and enters the valid Password into the field.<br>User click button log in. |
| **Test Inputs** | Valid user email and password into assigned fields. |
| **Expected Result** | Login Successful. User returning to the main page where he/she can choose for which transport will book a ticket. |
| **Likely Problem/Bugs Revealed** | None |

| | |
|---|---|
| **Test ID** | 1.2 Log in |
| **Purpose of Test** | Make sure that the user can't log in on the website with invalid information. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the testing on a laptop.<br>Operation system: Windows 10 |
| **Test Steps** | In the log in page user should do:<br><br>User clicks with the mouse on the field that says Email and enters the valid Email address into the field.<br>User clicks with the mouse on the field that says Password and enters the valid Password into the field.<br>User click button log in. |
| **Test Inputs** | Wrong user email and password into assigned fields. |
| **Expected Result** | Login Unsuccessful. The user stays at the same page and sees notification that Email and/or Password is incorrect. |

| **Likely Problem/Bugs Revealed** | None |
|---|---|

| **Test ID** | 2.1 Change Password |
|---|---|
| **Purpose of Test** | Make sure that the user can change his/her password when is it needed. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the testing on a laptop.<br>Operation system: Windows 10 |
| **Test Steps** | In the user, profile page tester should :<br><br>Clicks with the mouse on the field that says Enter an old password and enters the valid old password there.<br>Clicks with the mouse on the field that says Enter an old password and enters the new password there.<br>Click button save changes that locates at the center-bottom page. |
| **Test Inputs** | User enters old password and new password into assigned fields. |
| **Expected Result** | Change of password is successful. The user redirects to user profile page. |
| **Likely Problem/Bugs Revealed** | None |

| **Test ID** | 2.2 Change Password |
|---|---|
| **Purpose of Test** | Make sure that the user can change his/her password when is it needed. Also, the system can recognize when the wrong old password was entered. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the testing on a laptop.<br>Operation system: Windows 10 |
| **Test Steps** | In the user, profile page tester should :<br><br>Clicks with the mouse on the field that says Enter an old password and enters the invalid old password there. |

| | |
|---|---|
| | Clicks with the mouse on the field that says Enter an old password and enters the new password there.<br>Click button save changes that locates at the center-bottom page. |
| **Test Inputs** | User enters a wrong old password into assigned field.<br>User enters new password into assigned field. |
| **Expected Result** | Change of password is unsuccessful. The user stays on the same page and sees a notification that the old password is not matching with entered one. |
| **Likely Problem/Bugs Revealed** | None |

| **Test ID** | 3.1 Log out |
|---|---|
| **Purpose of Test** | Make sure that the user can log out from the website after finishing his booking tickets and his personal information can't be accessed after log out. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the testing on a laptop.<br>Operation system: Windows 10 |
| **Test Steps** | Click the logout button at the top right-hand side of the page. |
| **Test Inputs** | User must be already logged in. |
| **Expected Result** | Logout successful. |
| **Likely Problem/Bugs Revealed** | None |

| **Test ID** | 3.2 Log out |
|---|---|
| **Purpose of Test** | Make sure that the user can log out from the website after finishing his booking tickets and his personal information can't be accessed after log out. |
| **Test Environment** | Ures hardware : Asus F556U laptop.<br>No other application was running during the |

| | |
|---|---|
| | testing on a laptop.<br>Operation system: Windows 10 |
| **Test Steps** | Click the logout button at the top right-hand side of the page.<br>Go back to the main page and try clicking the button again. |
| **Test Inputs** | User must be already logged in. |
| **Expected Result** | Logout unsuccessful. |
| **Likely Problem/Bugs Revealed** | None |

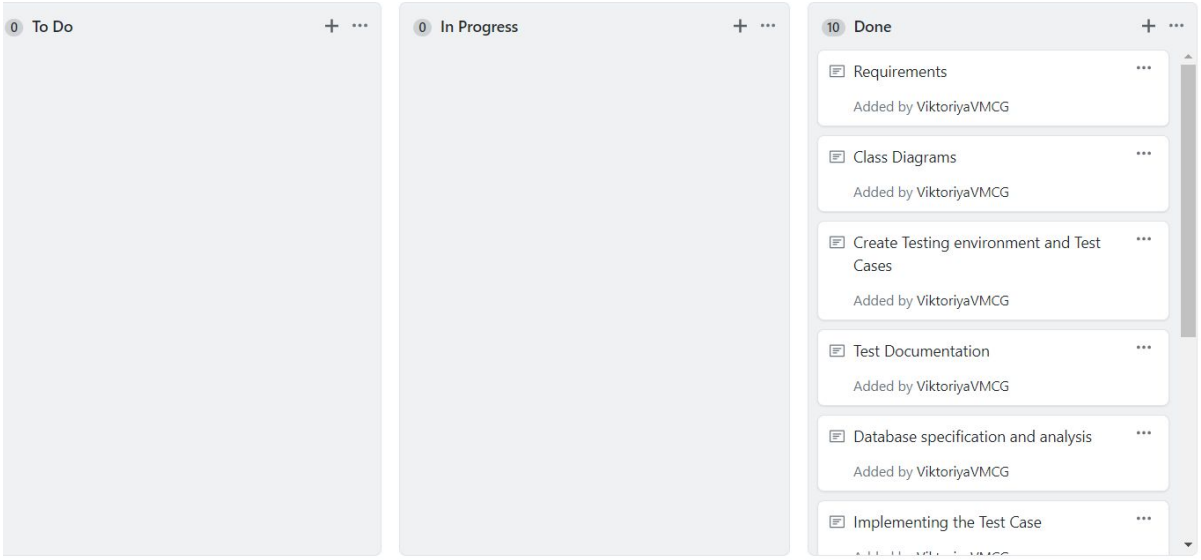
**Test Documentation:**

During the implementation test, any errors didn't appear.

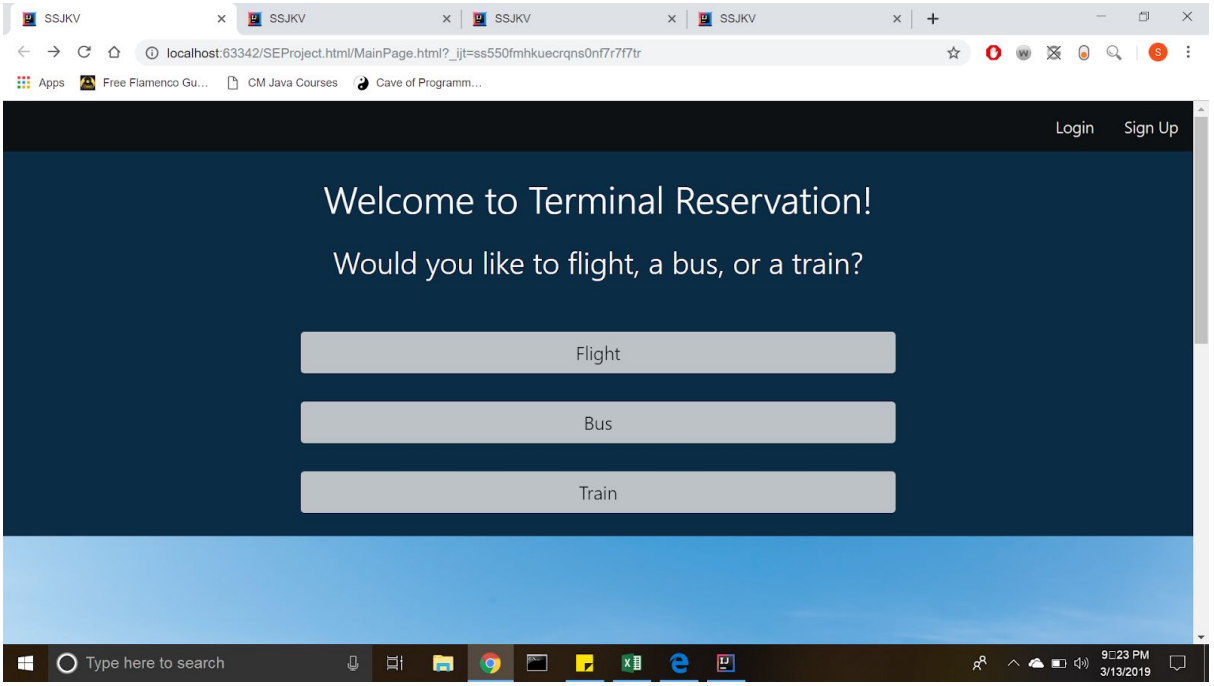
| Bug | The test uncovered the bug | Description of the bug | Action was taken to fix the bug |
|---|---|---|---|
| None | | | |
| None | | | |
| None | | | |
| None | | | |

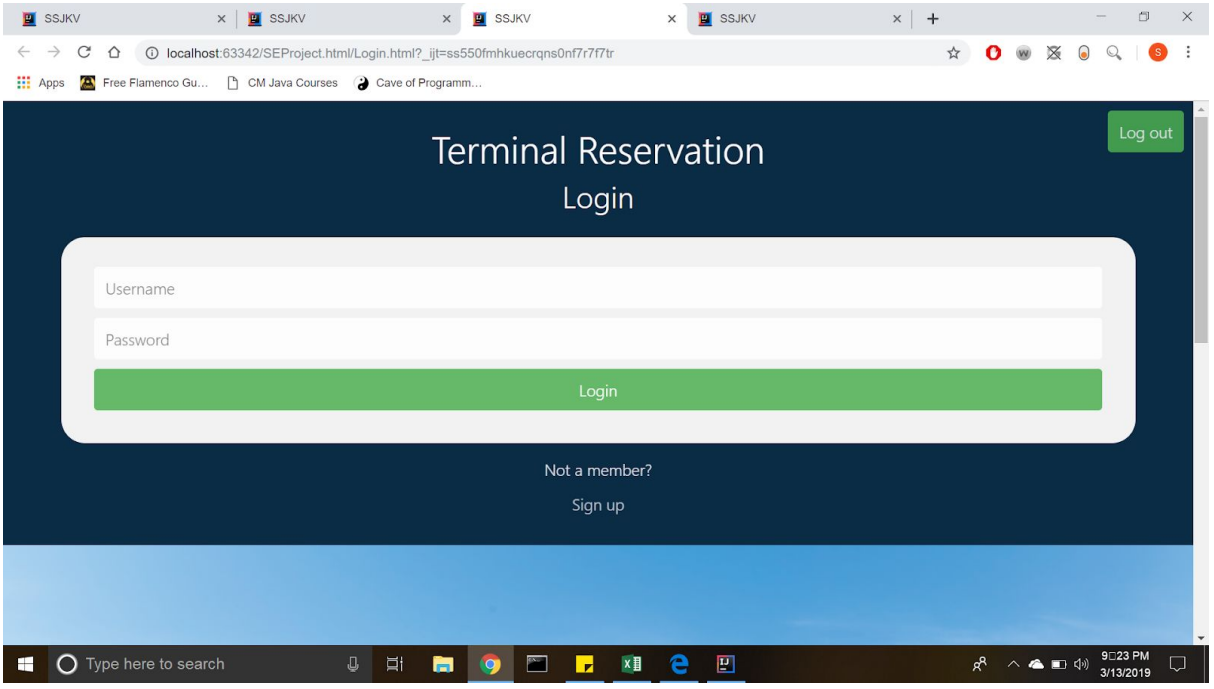
## Appendix

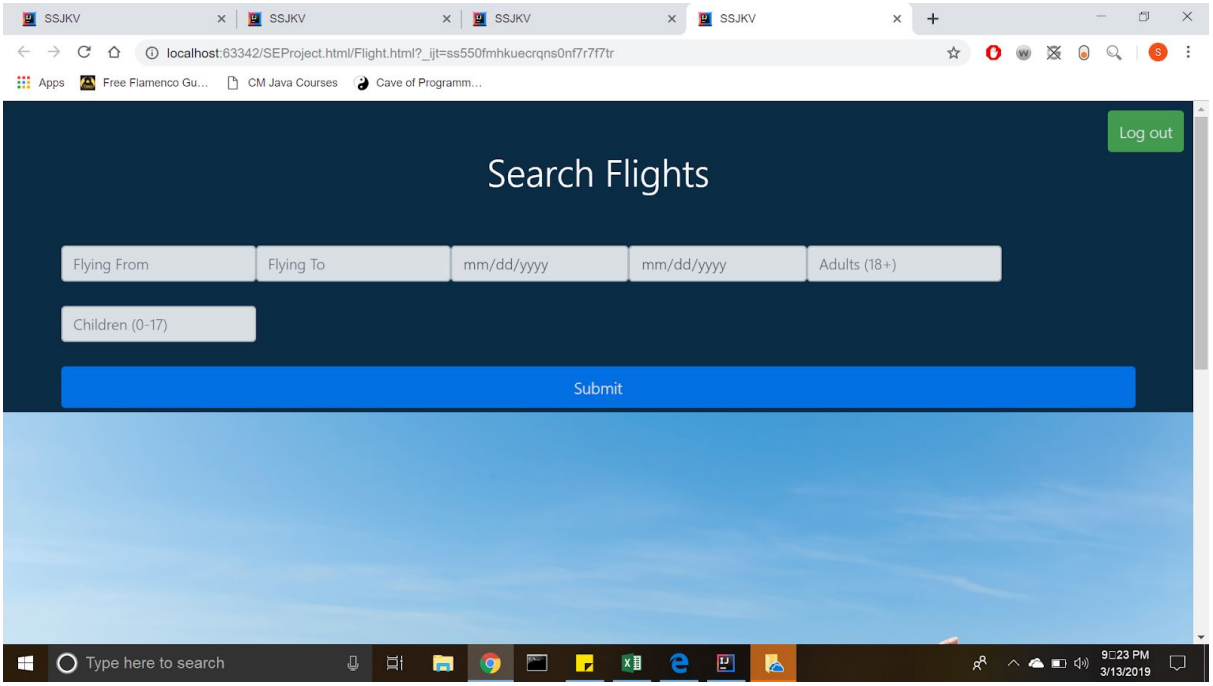
https://github.com/5aleh/book-on-time/projects/4

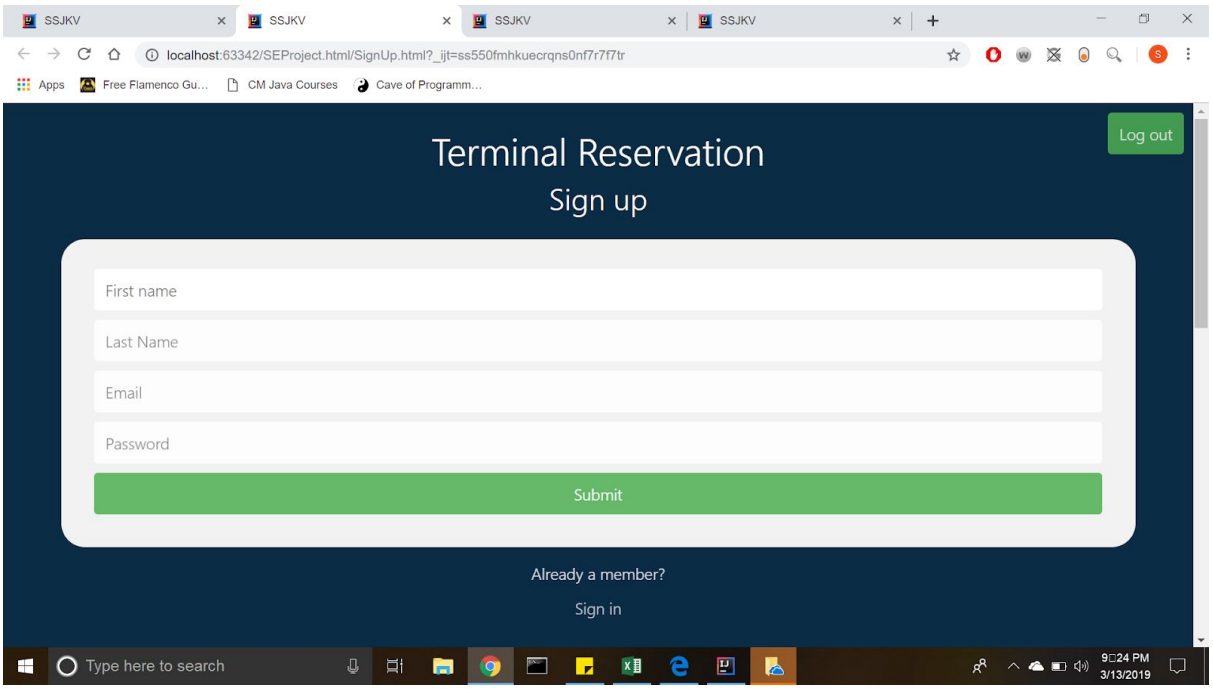


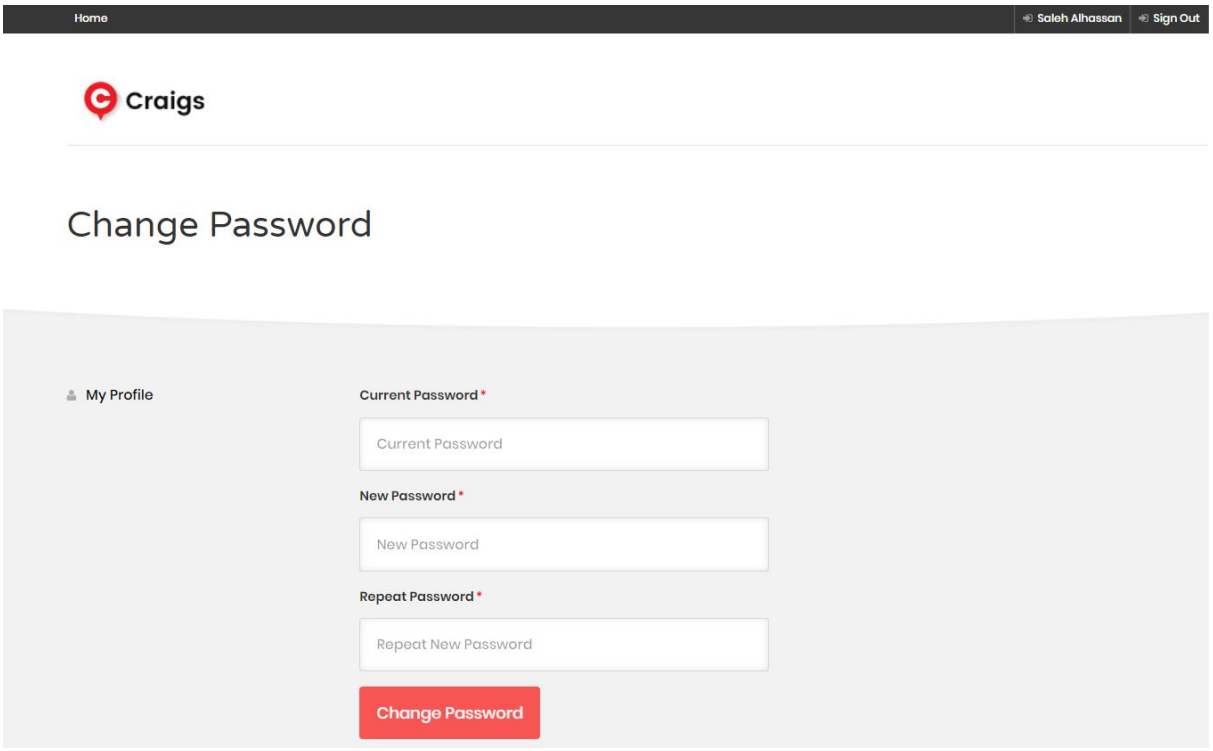
## Implemented Fronted




SSJKV
localhost:63342/SEProject.html/Flight.html?_ijt=ss550fmhkuecrqns0nf7r7f7tr
Apps
Free Flamenco Gu...
CM Java Courses
Cave of Programm...
Log out
Search Flights
Flying From
Flying To
mm/dd/yyyy
mm/dd/yyyy
Adults (18+)
Children (0-17)
Submit
Type here to search
9:23 PM
3/13/2019


SSJKV
localhost:63342/SEProject.html/SignUp.html?_ijt=ss550fmhkuecrqns0nf7r7f7tr
Apps
Free Flamenco Gu...
CM Java Courses
Cave of Programm...
Log out
Terminal Reservation
Sign up
First name
Last Name
Email
Password
Submit
Already a member?
Sign in
Type here to search
9:24 PM
3/13/2019




The change password page above is part of what what was presented in assignment 3 for the implementation section. We intend to update at least the color pattern design in the future.