Mobile to Cloud: Final Project Writeup

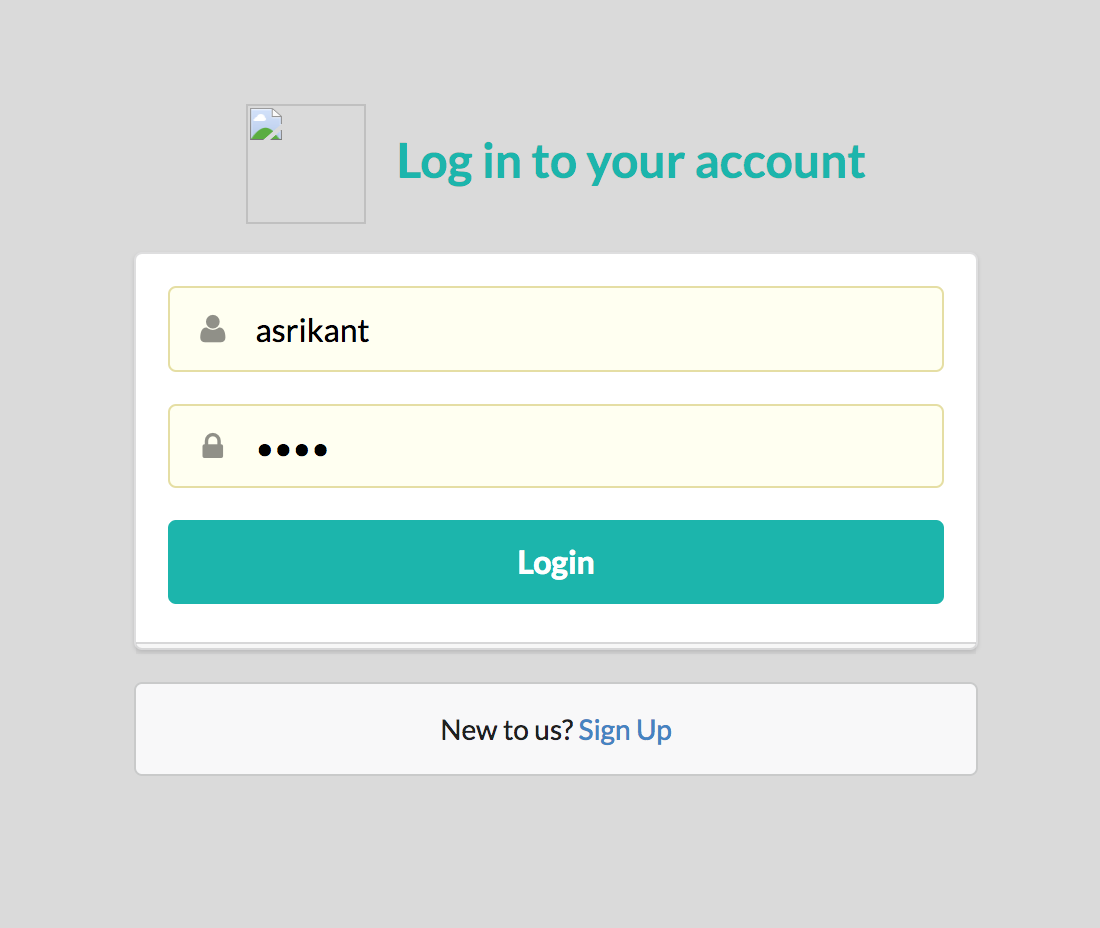
Anu Srikanth

1. **Implement a distributed application that does something of value.**
   * **Include a description with screenshots of user interaction**

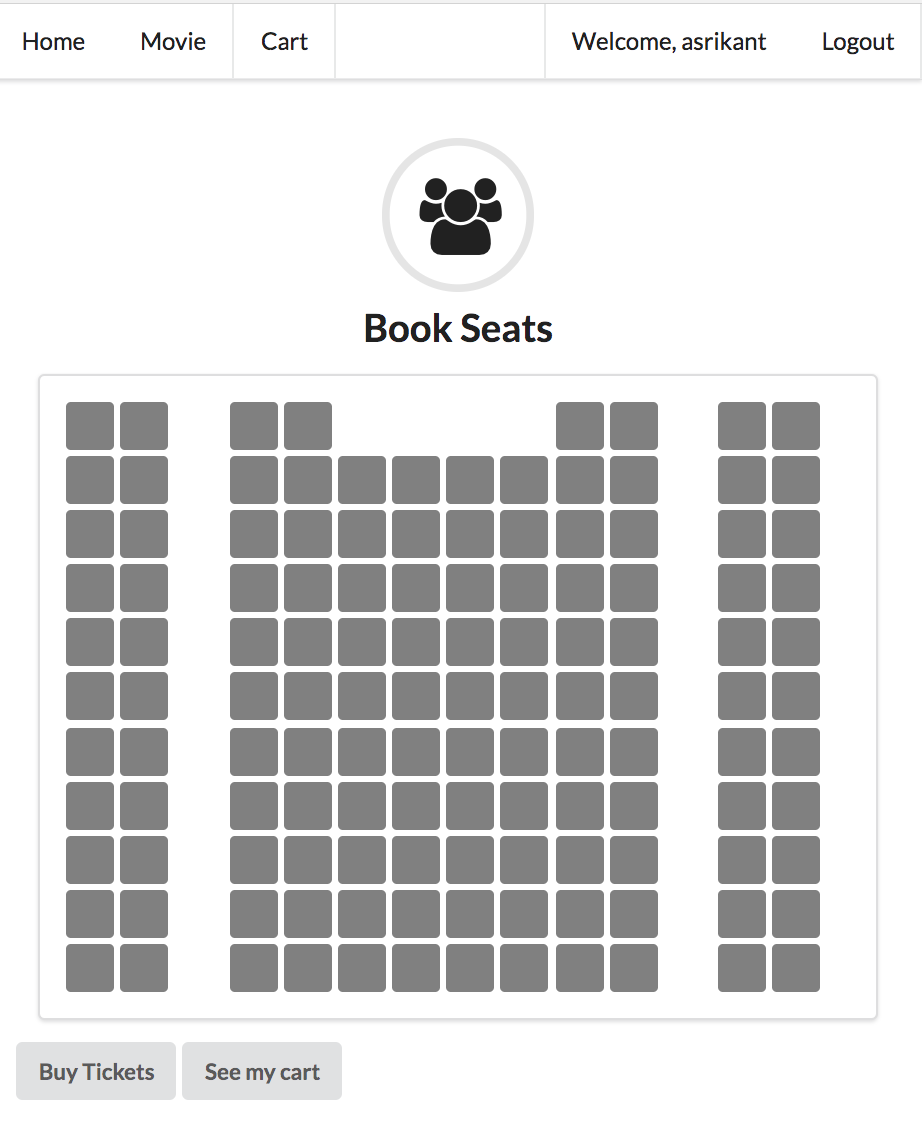
My application basically implements a ticket booking system. Currently the process of ticket booking is used in many different industries-movies, concerts, award ceremonies, wedding seating, etc. there is currently no unified application that can handle real time ticket booking without a manual component. This project is my attempt to fix this problem.

The following shows the different capabilities of the web application:

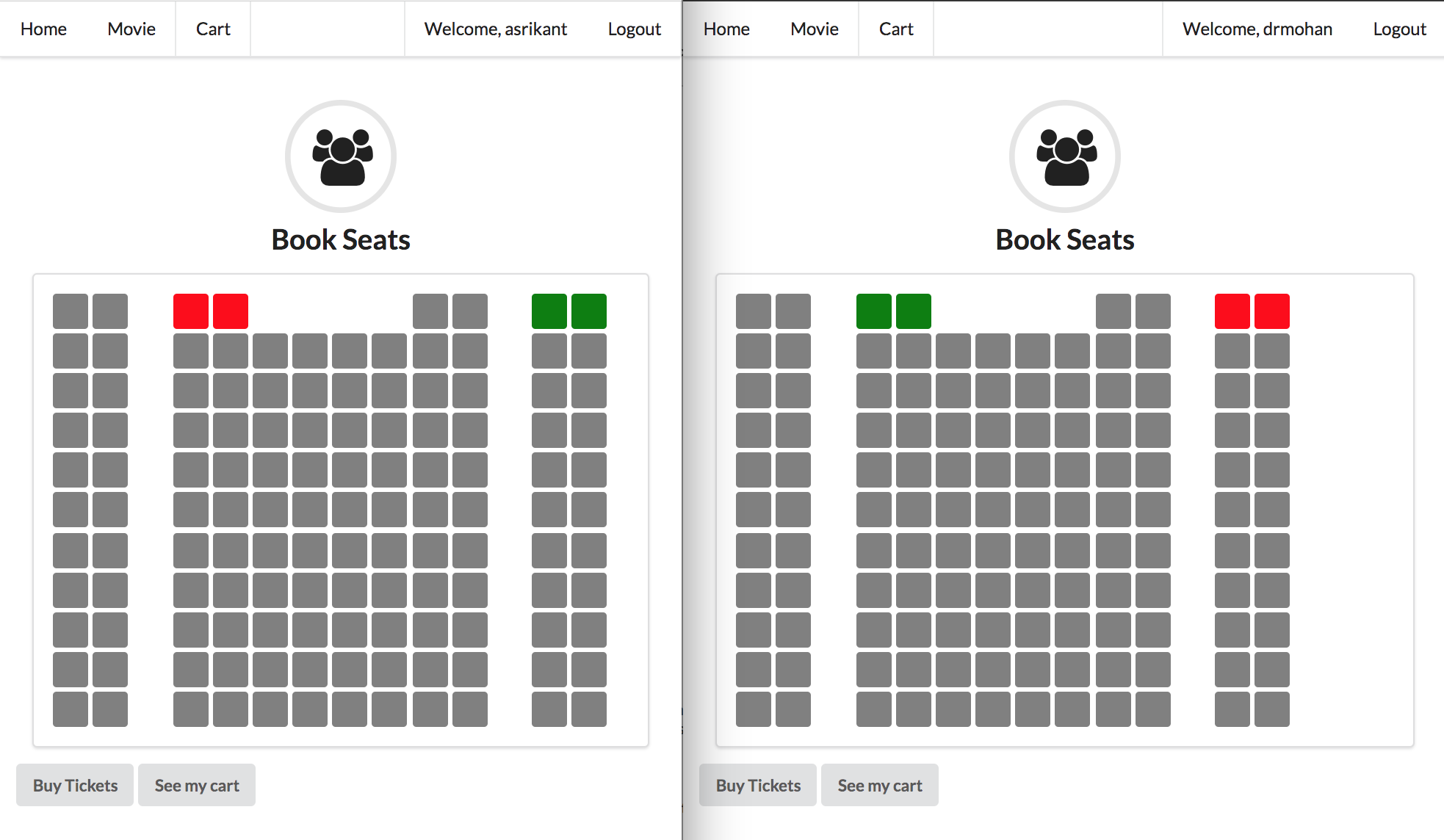
1. Login functionality – uses passport to carry out authorization and authentication on users that have created accounts on this web application



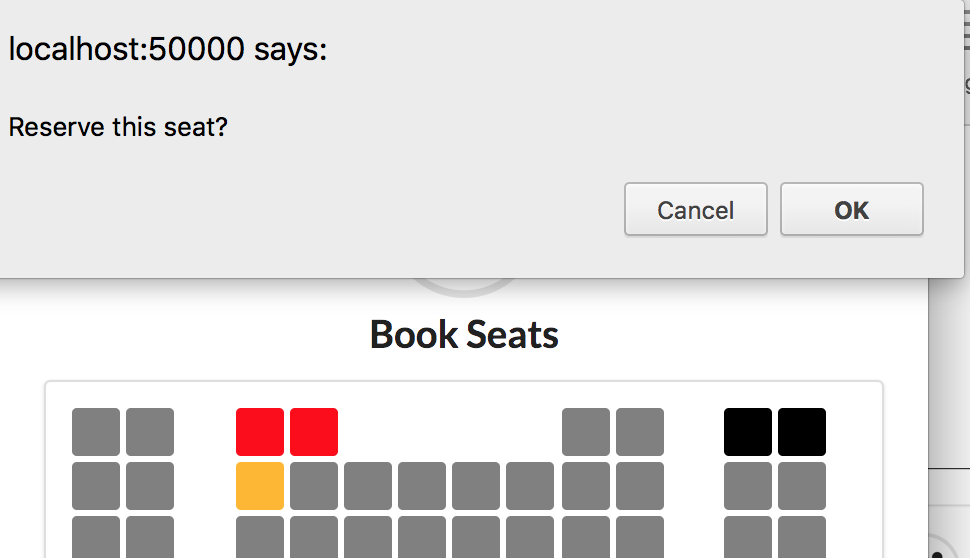
1. This is the home page of the application. This is where you can temporarily reserve tickets and book tickets.

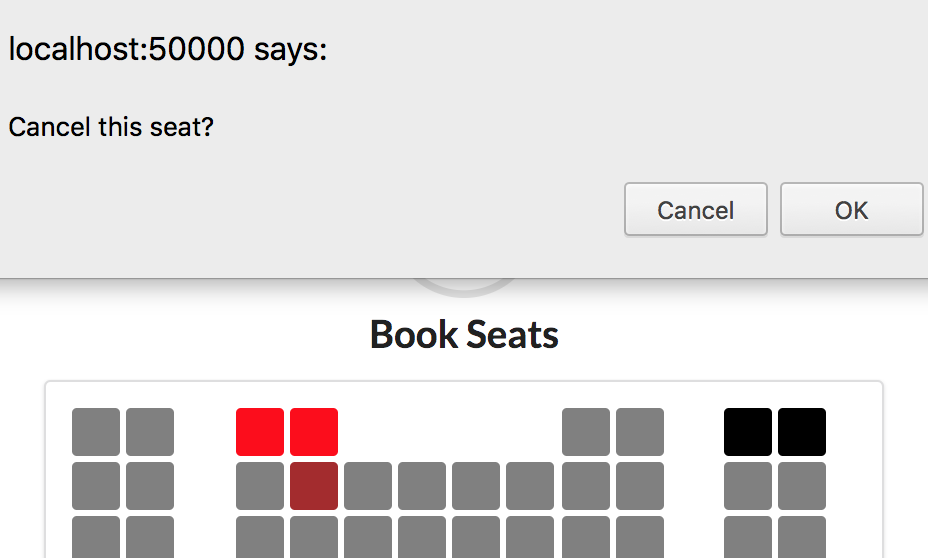


1. The following shows you the realtime nature of the application. It uses sessions and sockets to temporarily reserve tickets. As you can see below, the green squares show the tickets that have been reserved from that account. Those tickets are immediately unavailable to the other user (as demonstrated by the red square).

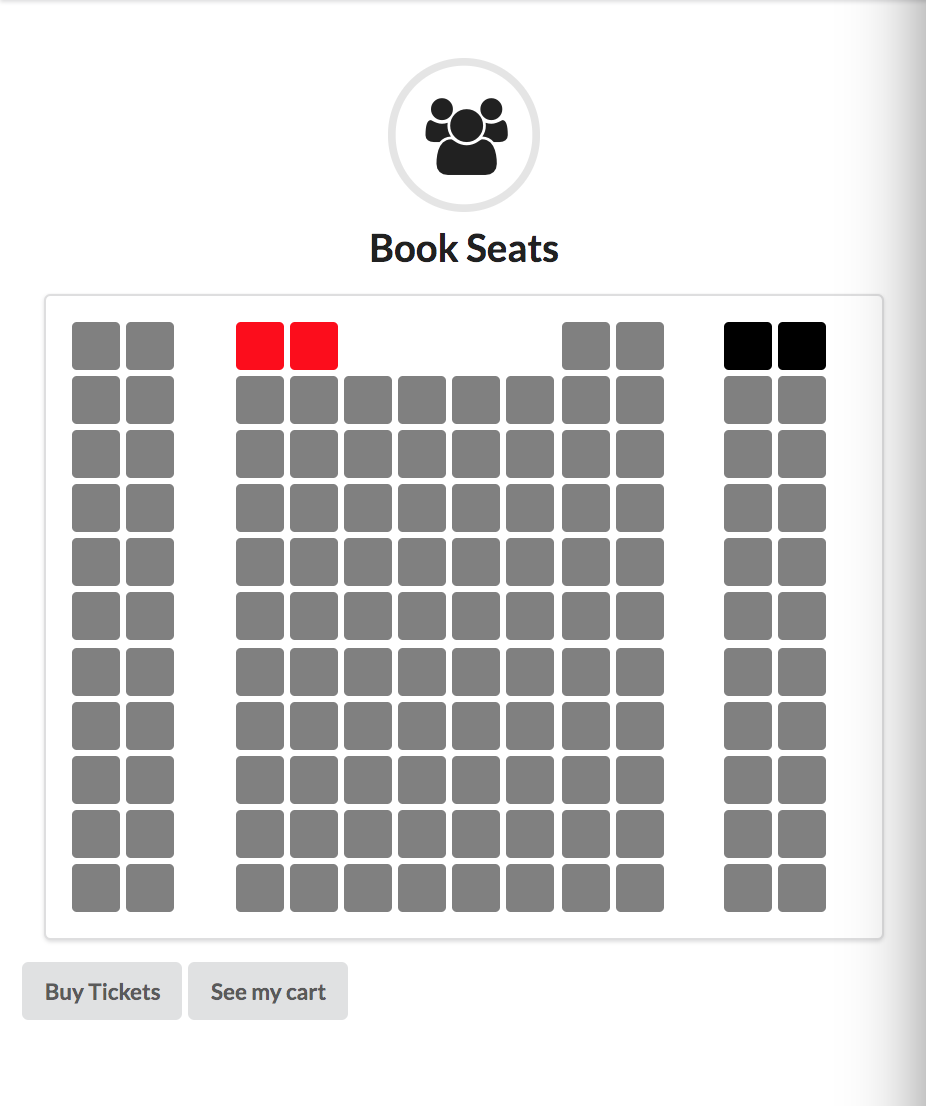


1. The following popups confirm that you want to reserve or cancel your reservations. Both of these are session based. If you refresh the page or log out, these seats will no longer be reserved. They are only to allow you to reserve all the seats that you want and then book the tickets permanently later, instead of one by one.

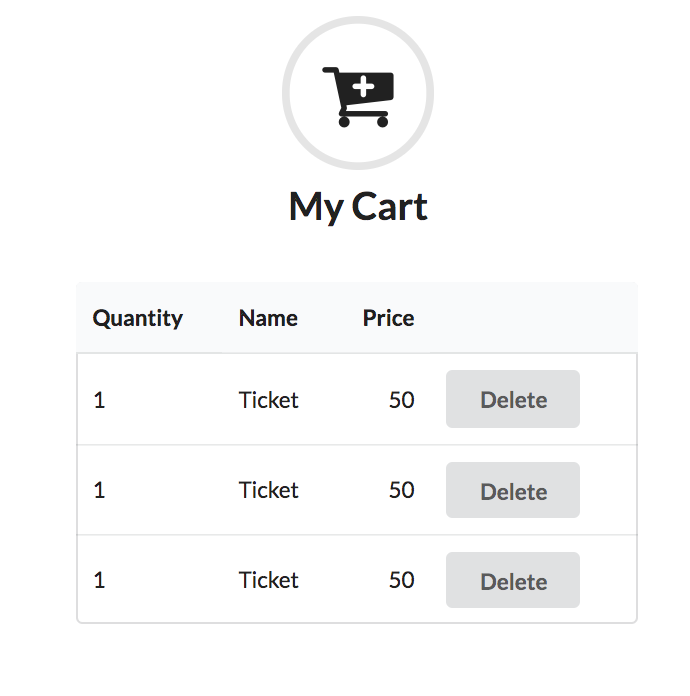




1. The tickets that you temporarily reserved are booked when you click on the book tickets button. These tickets are associated with your account and are saved to the account. You will be able to view them in your cart.



1. If you click on the view cart functionality, it will show you all of the tickets that are associated with your account.



Features to add in the future:

1. There would be functionality to create your own seats and spaces.
2. You could connect it to other frameworks and apis to specialize it. For example, it could be connected with a theatre’s movie showtimes api in order to book tickets for each of these shows.
3. You could attach a 3rd party bank transaction handling software such as square or stripe in order to pay for the tickets in your cart. You could also handle transactions inhouse with node modules that deal with credit card information and added models.
4. **Updates the content displayed on the client based on interaction with the server.**

The client side has the display to the seating arrangement. The server side has the connection to the database and the requests that are coming in. As shown above, when a ticket is booked, the front end colors immediately change. After you reserve a ticket, classes are removed from elements so you can no longer reserve those tickets. Those are all done on the client side after the server side reserves a ticket and gives the client side a request.

1. **Use Ajax style interaction or WebSockets (via**[**socket.io**](http://socket.io/)**)**
2. **The server-side application interacts with other systems by one or a combination of:**
   1. **Accessing one or more 3rd-party web services**
   2. **Coordinating live collaboration with other users**
   3. **Interacting with some physical (IoT) device**

**(These questions are related and I have answered them togeth below, describing the workflow.)**

I use both [socket.io](http://socket.io/) and ajax requests on this website. Ajax requests are used to handle requests that don’t relate to the session such as creating new users, etc.

Socket io is used to have a conversation between the client and the server regarding the ticket booking. The following shows a sample interaction between the client and the server using [socket.io](http://socket.io/):

* + - The client goes to the movies page, a socket connection is initiated. The client's username is connected to the socket connection.
    - The client clicks on different seats that he or she wants to book. This will update the server real time. This does not mean that the tickets are booked, they are just reserved temporarily. If another user is connected on a different socket connection, he or she will not be able to book that seat. Also, the seat is temporarily reserved so it can still be cancelled by clicking on the reserved seat and confirming the cancel.
    - The book tickets button sends another request to the backend with the seat information in order to book those tickets for the client. These tickets are then stored in the database and associated with the account.
    - There is live collaboration among users because people can see realtime updates of the seat reservations. But if these are not booked, they are not persistent.

1. **Demonstrate good separation of concerns with an MVC architectural style.**

There is a good separation of concerns because there are different folders in the project that handle different operations:

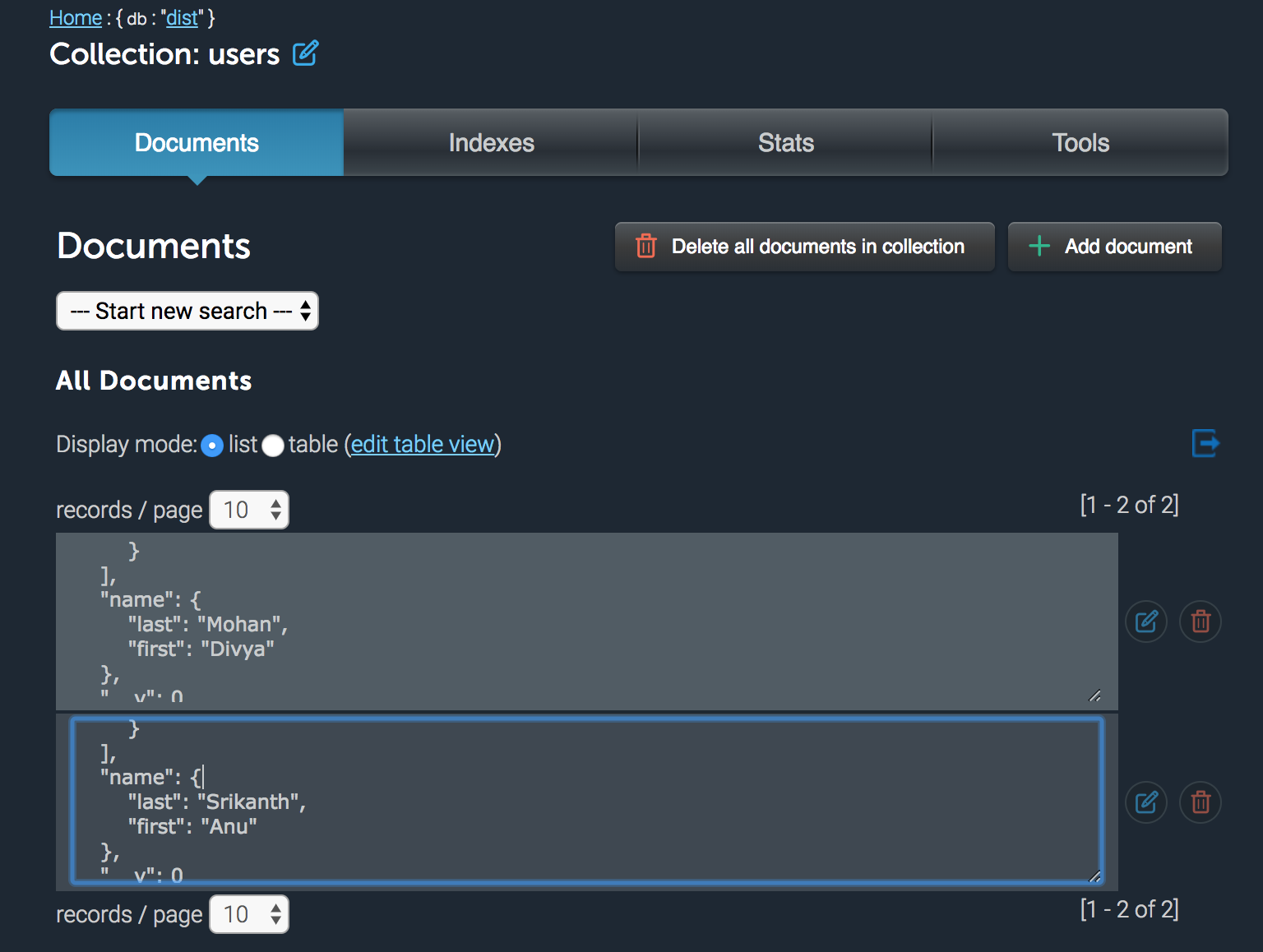
* 1. Models – this folder contains the model logic, namely the business logic for seats, tickets and users. These include the mongoose schema which requires the format the limitations that is placed on the data that is stored in this database.
  2. Routes – this folder contains the controller logic, routing logic that routes different urls to different blocks of code and front end pages and manipulate the backend based on user input.
  3. Views – this folder contains all the front end code. These use the ejs template so that they can embed blocks of JavaScript code in the front end and be responsive to changes in the backend.
  4. Public – this folder contains all of the static pages that need to be loaded and the static javascript that gets executed. For example, for ticket booking, static javascript loads the front end of the seats onto the page.

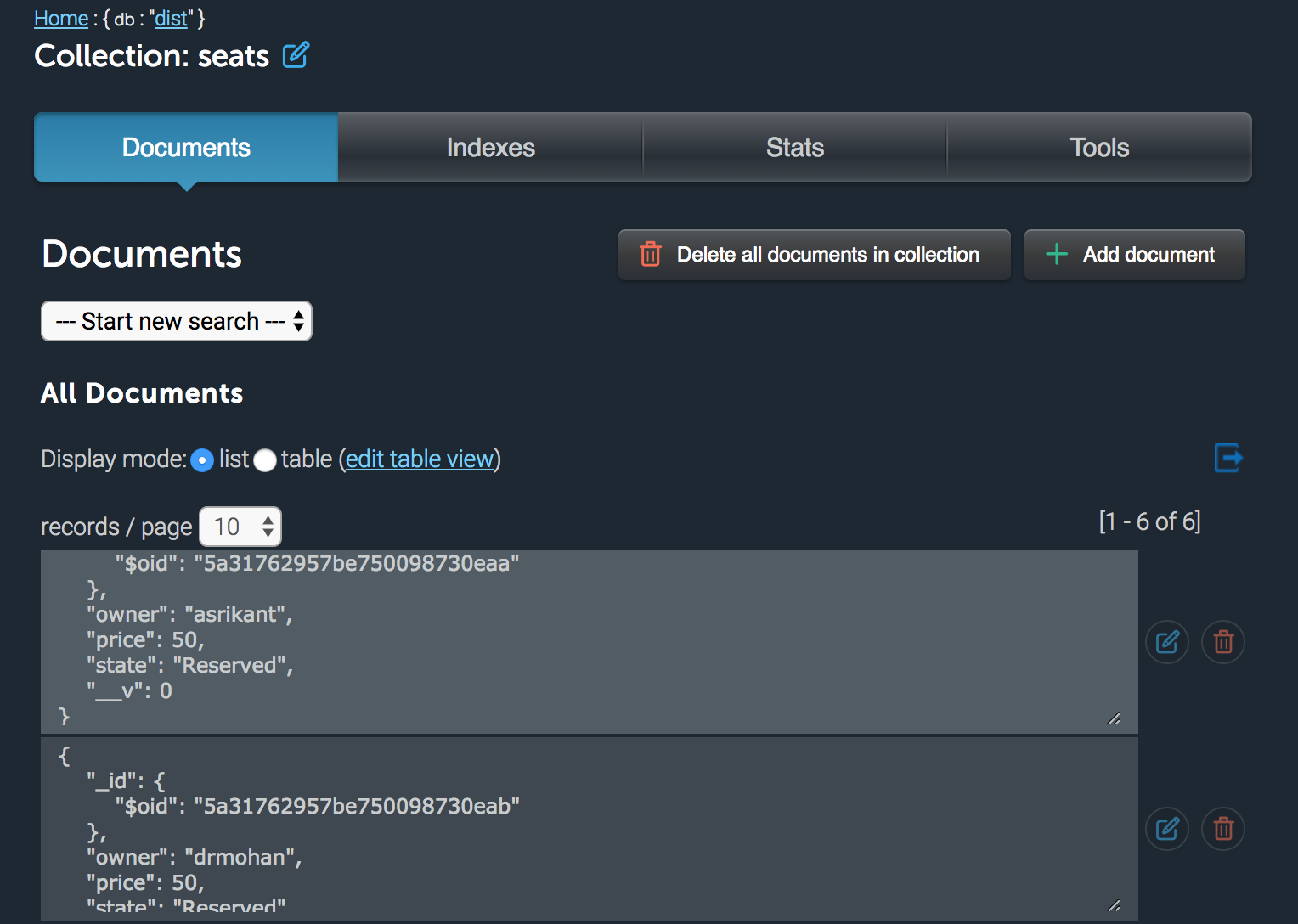
1. **Use cookies and/or sessions (or localStorage / sessionStorage) to manage web application state information.**

I use the PassportJS middleware for authentication, authorization, session creation and creation of cookies that store session information from page to page. This allows the user to access information pertaining to their account even when they move from page to page.

1. **Store some data persistently using a server-side database.**

Ticketing information and user account information is stored persistently in the database. Session information, temporary reservations etc are not.



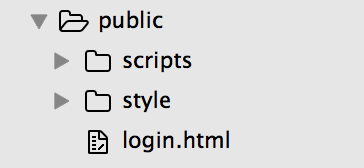
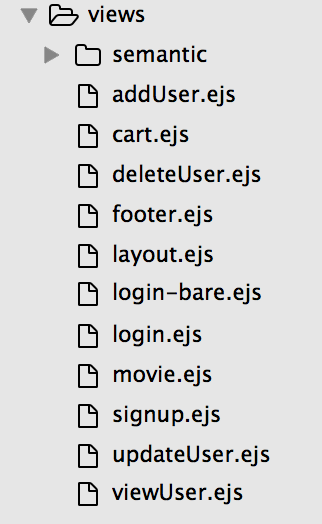


1. Adapt for access by desktop (function may be different than mobile)
2. Adapt for access by mobile (function may be different than desktop)

I used a framework and web queries to customize the front end to fit mobile and desktop sizes and adapt to what size of screen it is seeing.

1. Be deployed and working in the cloud via Zeit Now (make sure /\_src is available)
2. Demonstrate good separation of JavaScript, CSS, and HTML

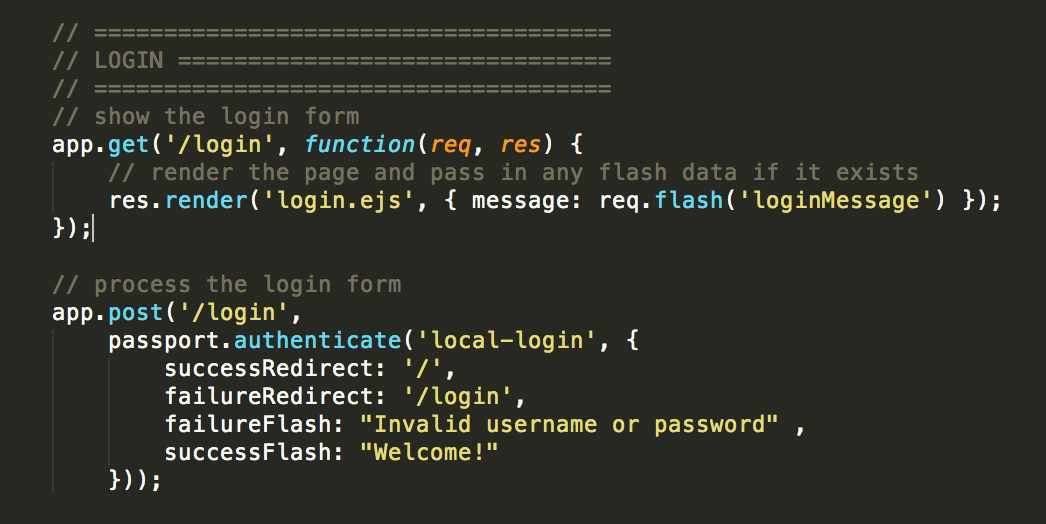
* Scripts – contains all of the javascript for the html files
* Style – contains all of the css for the html files
* Semantic – contains all of the CSS for the EJS files

1. Make appropriate choices of HTTP methods

All the relevant HTTP methods are made depending on the request:

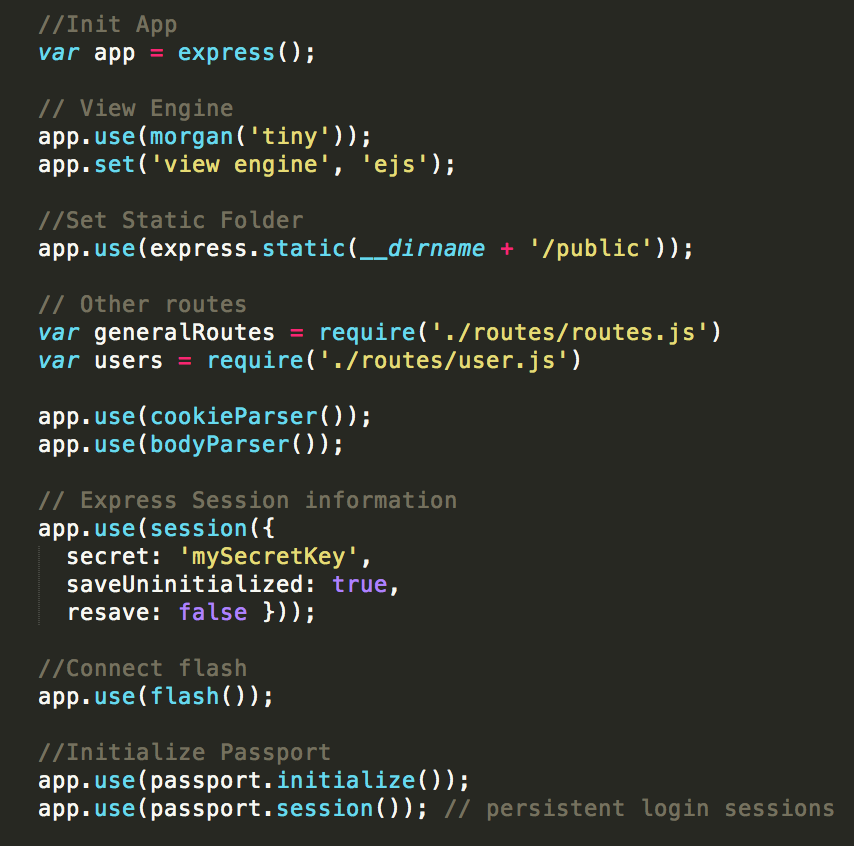
1. PUT method – this method is used to add users, add tickets to the system after a new reservation.
2. GET method – this method is used to retrieve account information, retrieve the user, retrieve ticket information and what seats those are from, etc.
3. POST method – this method is used to edit a particular user, change his or her information, edit a seat’s information to change it’s state (for example, if a user has booked a ticket for a particular seat, we want to update the seat’s information to make sure that it cannot be booked in the future)
4. DELETE method – this method is used to delete a temporarily reserved seat, delete a user, etc.



1. Demonstrate good coding and commenting style.

Below are excerpts of my code with good commenting. They make up the bulk of the logic of my application.

1. The first is my set up of the app, modules and passport.



1. This is setting up of passport’s functionality. It sets up a local strategy on how to handle people who sign up.



1. This is the functionality for the backend logic of handle socket connections, receiving signals between the client and server side and manipulating the database



1. Employ authentication and authorization. (This must also include the ability to *create* new logins.)

I used PassportJS to carry out authorization and authentication. We are able to create new accounts and login with created accounts.

1. Be published to a GitHub repository, including LICENSE and complete README.md files.

