11/12/2017

Anu Srikanth

Project Proposal

# Overview

For this project, I will attempt to create an application for event creation, space reservations and processing. I came up with the idea for this application after seeing that there was a dearth of application that customize space reservations to the organization and events.

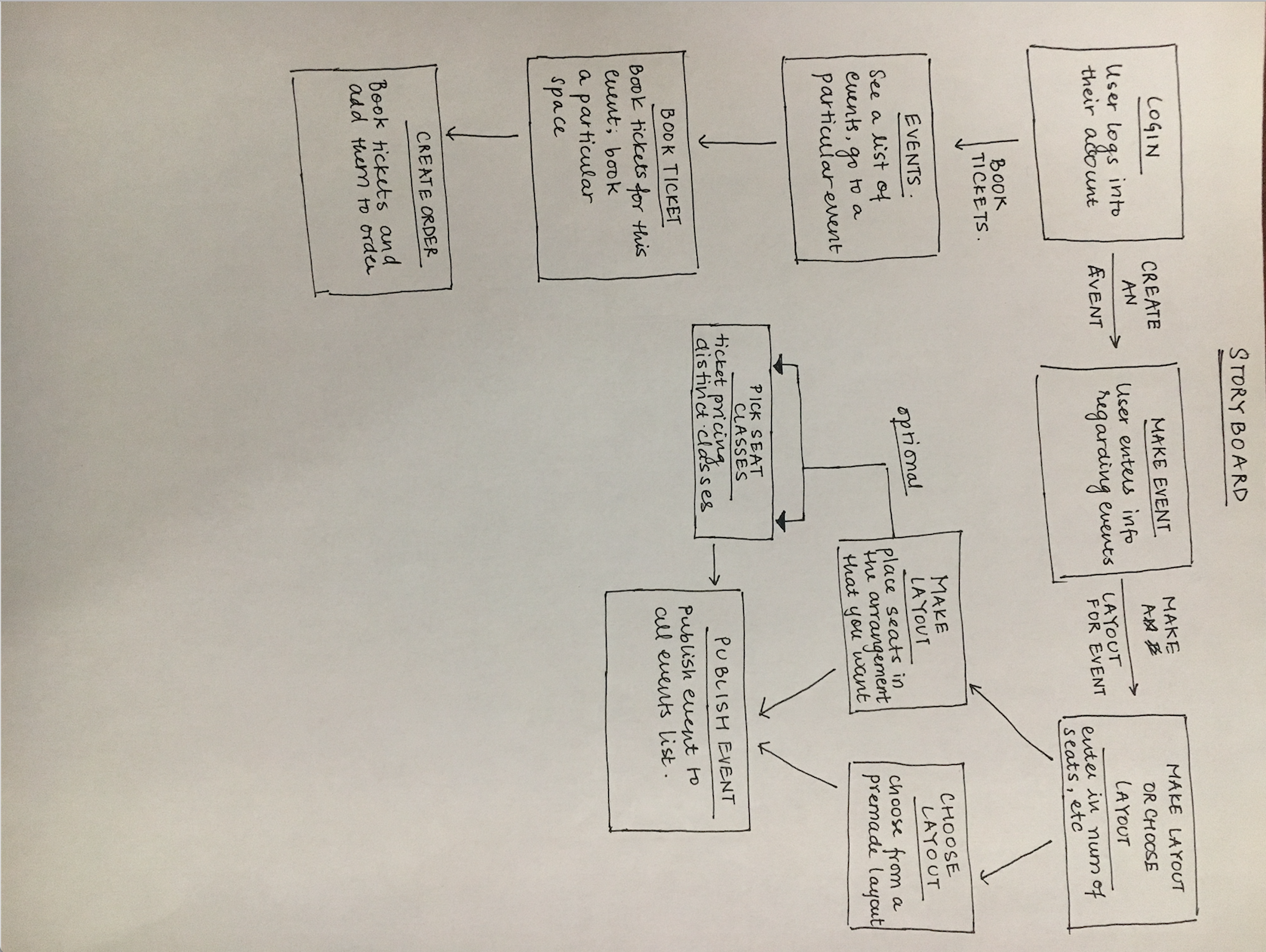
A user will be able to make an account. From that account, he can choose to either make a space reservation systems for an event or book seats at another event. If he chooses to make a space reservation system for his event, he can choose from a standard format or customize a layout to his location by making a custom layout along with custom ticket prices depending on seat location.

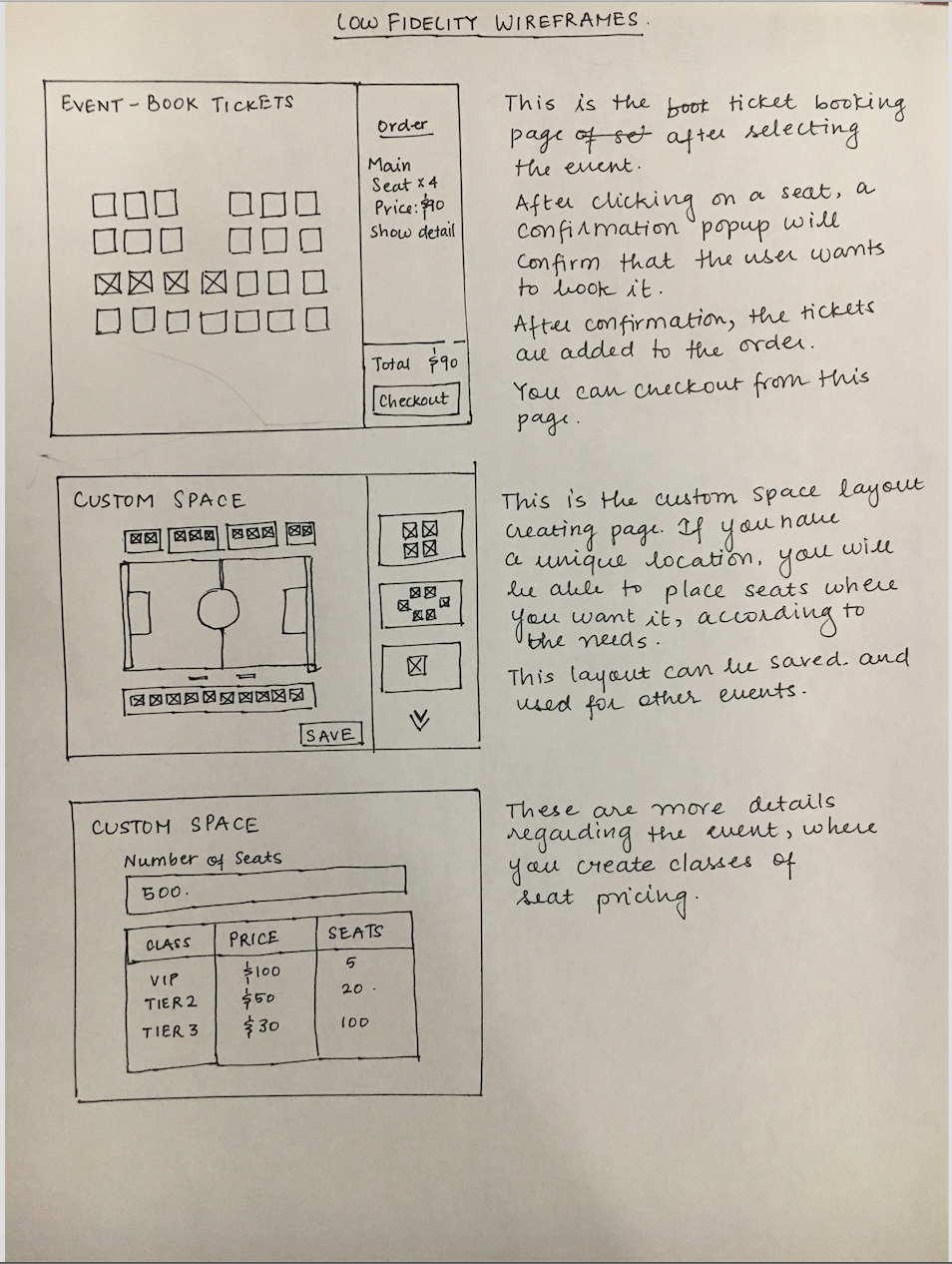
If he chooses to reserve seats at another event, this will require socket.io to update all of the other users real-time that the seats are being reserved.

The following proposal gives a detailed set of use cases that will be implemented for this project as well as a detailed description of the framework of the application. To expedite the process, I want to limit the number of database calls made, however, certain information such as account information will be stored in the database.

# StoryBoard

I am including here a storyboard of the various capabilities that this application has to offer as well as wireframes for those key features. The wireframes are hand drawn because they are very specific to this application and cannot be replicated using standard wireframing tools because they will not be able to convey the specific purpose and uses.





# External Web Services

I am using the following technologies and services in my application.

|  |  |  |
| --- | --- | --- |
| **Technologies** | | |
| **Technologies** | **Type** | **Description** |
| Socket.io | Framework | This framework will be used in the reservation system so that everyone can, in real-time, see what seats are being reserved. |
| Sails.js | MVC framework | This is an MVC Framework that includes Express framework that organizes web applications in a particular way. |
| Eventful API or similar local event detector | API | This is an API that can give a list of local events that can be booked using this application. |
| MongoDB | Database | This is the database that would be used to store the account and event information in. |

# Functional RequirementS

This section will describe the business logic of the application, the MVC components that are present in the application, the routes that the front and the backend of the application communicate, the socket sessions and interactions that are taking place, and the database of the application.

## URL Patterns for the Routes to handle

|  |  |
| --- | --- |
| Routes | Description |
| /account | * View your account details |
| /orders | * View all of your orders for account that created an event. (I don’t know if this is necessary) * View all of the orders you’ve places on different events |
| /orders/:id | * View details of a particular order and be able to edit it or delete from that page. This should update the event’s seat reservation page also |
| /events | * View list of events that are happening. * There will be a search box, powered by a same page JavaScript script that hides events that don’t apply. |
| /events/:id | * View the details of a particular event |
| /events/:id/reserve | * Reserve seats for this particular event. * Add these reserved seats to your order for this account |
| /space/new | * Create a space layout for the event or just to save for a later event * Popup to create a class of seats to distinguish one ticket from another. |

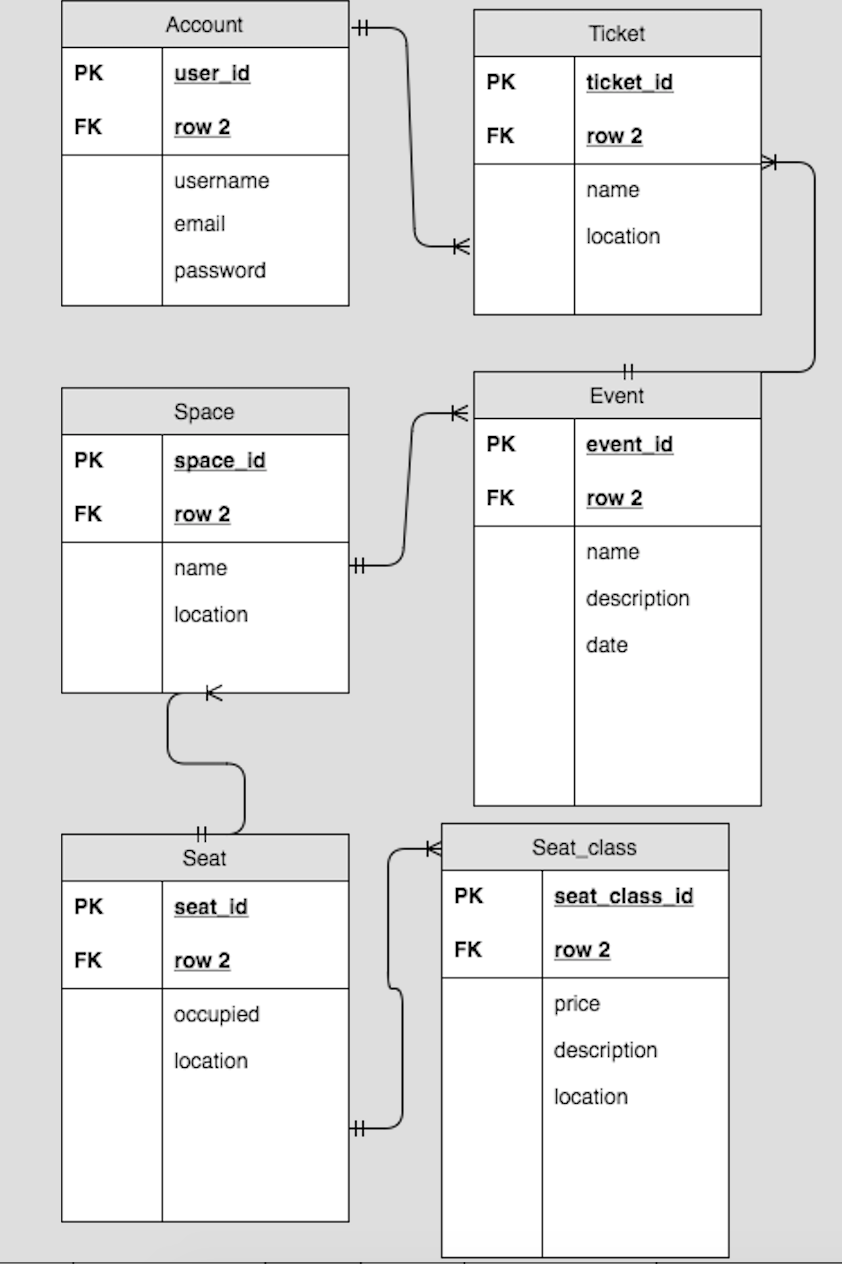
## Socket.io interactions

Socket.io interaction for booking a seat in an event is shown as follows. It shows the interactions between the client and the server.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Client or Server** | **Emit or detect** | **Event** | **Description** |  |
| Client | Emit | reserve | Client emit a reserve event when the user clicks on a seat. They will transmit the location of this seat |  |
| Server | Detect | reserve | Server picks up the reserve event when the user and the location of the seat. Check to see if the seat has already been filled. |  |
| Server | Broadcasts | Seat\_update | Server, if the previous check has passed, then the server broadcasts a seat\_update event to the client side |  |
| Client | Detects | Seat\_update | Updates the front-end display of the seats to disable the seat for everyone else but for you make it a reserved seat. |  |

## Business Logic – ERD Diagram

This ERD shows the layout of the backend of the application. The model names are the names of the tables and the rows within them are the attributes of these models.



## Business Logic - Methods

The following tables are the use cases we hope to accomplish by the end of the project. The A-level Use cases are the features that are part of the minimal viable product. The completion of these would provide all of the features described in our vision as well as our competitive advantage of making quick and easy real-time connections. The B-level use cases are features that will give us a significant competitive edge over the competition because these features are not present in any of our competition’s products. The C-level use cases are add-on features for specific client needs for smaller sections of our users.

|  |  |  |
| --- | --- | --- |
| **A-Level Use Cases** | | |
| **Use Case Name** | **Actor(s)** | **Description** |
| Create Account | User | The user can create an account with credentials |
| Login | User | The user should be able to login with previously created credentials |
| Create Event | User | The user should be able to create custom events with descriptions. |
| Choose a space layout for your event | User | Associate a space layout for an event so that it creates an instance of this. |
| Reserve seats in an event | User | User should be able to access the event space layout and reserve seats. This will be accomplished in a session. |
| Order tickets | User | User should be able to pick the seats that he wants and have this update globally. The order should be |

|  |  |  |
| --- | --- | --- |
| **B-Level Use Cases** | | |
| **Use Case Name** | **Actor(s)** | **Description** |
| Create seat classes | User | The user should be able to classes of seats and distinguish them by pricing and location in the space layout. |
| Fill in parameters of your group to get group ticketing and suggestions for seats | User | User should be able to fill in information to get group ticketing and suggestions for seats in the event. |
| Create a space layout | User | User should be able to create a custom space layout for their particular event. These consist of rows of seats arranged in a particular format. |

|  |  |  |
| --- | --- | --- |
| **C-Level Use Cases** | | |
| **Use Case Name** | **Actor(s)** | **Description** |
| Direct message through application | User | Be able to directly communicate with the creator of the event to ask any questions about event, tickets, etc. |
| Scheduling meetings through the application | User | Have an internal scheduling tool or connect to an existing calendar API |

## Database

The database will store information about the following models:

1. Account
2. Events
3. Space Layouts
4. Seats
5. Seat Prices and Classes
6. Tickets
7. Orders

The only information that will be stored in the session is an ongoing order and seat reservation information- for example, the seat that a person has chosen before placing the order will be maintained on the server as a variable and not in the database. The history of the orders and the events will be in a database. The database calls will be limited for intermediate steps in the seat reservation process so as to limit resource expensive database calls.