qwikstubs

Software Requirements Specification

Version 1.1

March 7, 2013

Prepared by

Group Name: The Constructors

Alex Hamstra	4506291	**alexhamstra@gmail.com**
Jared Roesch	4826574	**roeschinc@gmail.com**
Kyle Jorgensen	4165916	**kylewjorgensen@gmail.com**
Ben McCurdy	4336822	**bpmccurdy@gmail.com**
Brittany Berin	4298345	**brittany.berin@gmail.com**

Instructor:	Chandra Krintz
Course:	CS 189A
Lab Section:	*Friday, 12 PM*
Teaching Assistant:	Stratos Dimopoulos
Date:	February 12th, 2013
Mentor:	Colin Kelley

Table of Contents

Revision History

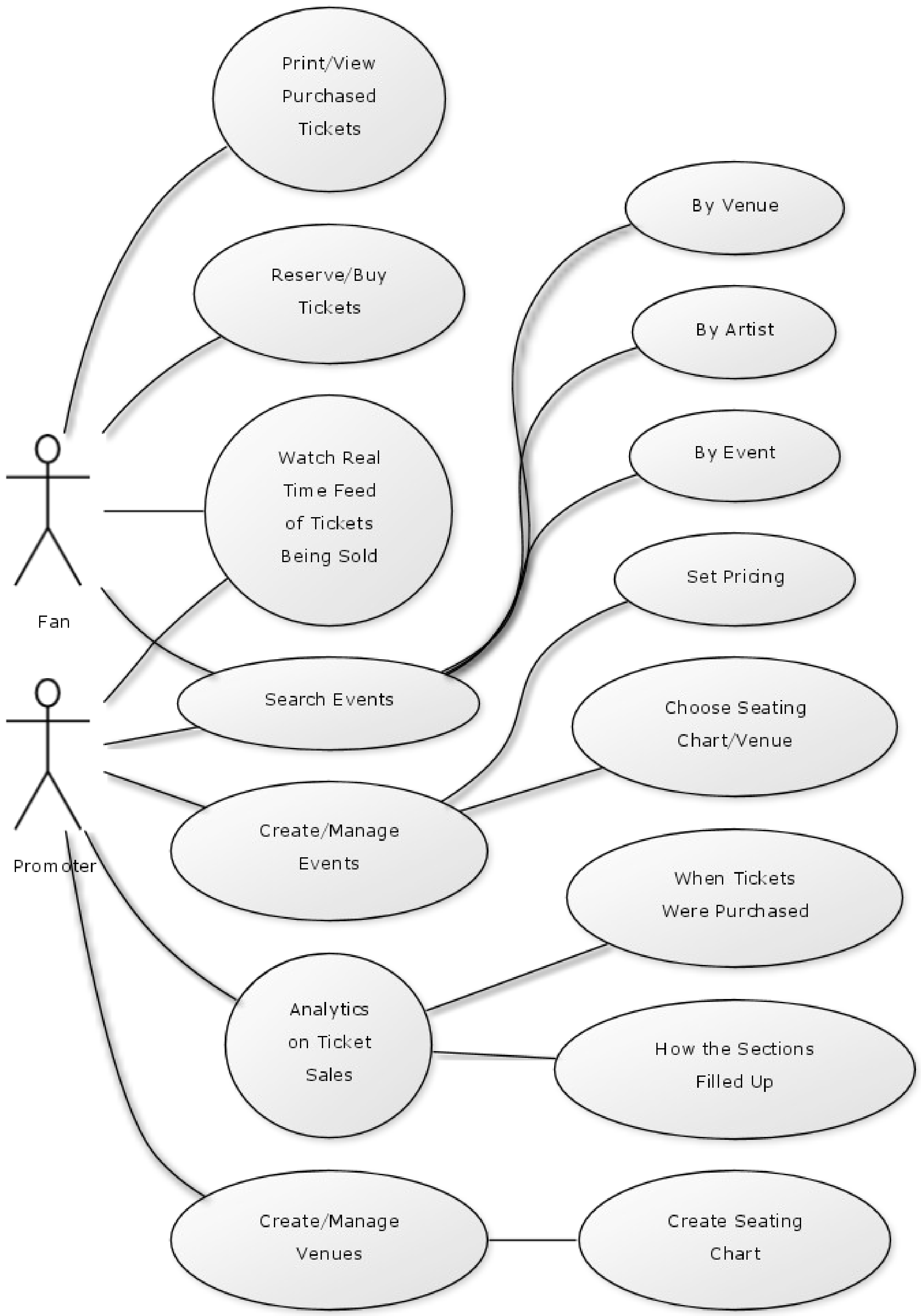
Version	Primary Author(s)	Description of Version	Date Completed
1.0	The Constructors	Initial version of our SRS, with more of a focus on our goals for our first prototype.	2/12/13
1.1	The Constructors	Updates based on feedback from Chandra. Some updated diagrams, more description about the scope of the project.	3/7/13

1 Overall Description

1.1 Product Perspective

Today's online ticketing systems are inefficient and often create technical difficulties that make it nearly impossible for customers to buy the tickets they need. Seating arrangements can be unclear, their websites strain under high traffic, and tickets in the cart can be lost between page loads. QwikStubs addresses these weaknesses, and serves as an open-source alternative to other universally accepted ticketing systems. What sets our product apart from its competitors is its sleek, hassle-free interface that delivers information to its customers in real-time.

In Figure 1, You can see the basic interaction between users and our system from a customers perspective.



Print/View Purchased Tickets
Reserve/Buy Tickets
Watch Real Time Feed of Tickets Being Sold
Search Events
Create/Manage Events
Analytics on Ticket Sales
Create/Manage Venues
By Venue
By Artist
By Event
Set Pricing
Choose Seating Chart/Venue
When Tickets Were Purchased
How the Sections Filled Up
Create Seating Chart
Fan
Promoter

Figure 1. User Perspectives

1.2 Product Functionality

QwikStubs provides the following services for its two types of targeted users:

Both a **fan** and a **promoter** can:

- Log into QwikStubs using their Facebook or Google account using OAuth, or if they prefer simply by personal email address.
- Enter the billing information to be used for future purchases from the QwikStubs website.
- Edit their account information as changes arise.
- Browse for a show using the artist name, venue, or date

A **promoter** can:

- Enter an upcoming event (Venue, Name, Artist Involved, etc)
- Set the ticket prices
- Set the ticket sales start date and time
- Set the seating chart for the event
- Publish the show for fans to see
- Create a new venue

A **fan** can:

- See the seating plan for the show with the varying ticket price levels
- See a live countdown to the start of ticket sales

Once a ticket sale begins, a **fan** can:

- Choose which seats to place on reserve
- Keep reserved seats in their cart for a maximum of five minutes (after which the seats will become available for another fan to purchase)
- Receive an electronic version of the tickets that were successfully purchased
- View which seats are taken and open in real-time as they are purchased by other fans.

Once a ticket sale begins, a **promoter** can:

- View which seats are taken and open in real-time as they are purchased by fans utilizing QwikStubs.
- See visualizations of the ticket transaction data.

After the sales period is completed and the event has begun, a **fan** can:

- Use a QR code or some other form of bar code on a mobile device to scan in a ticket at the Venue entrance.

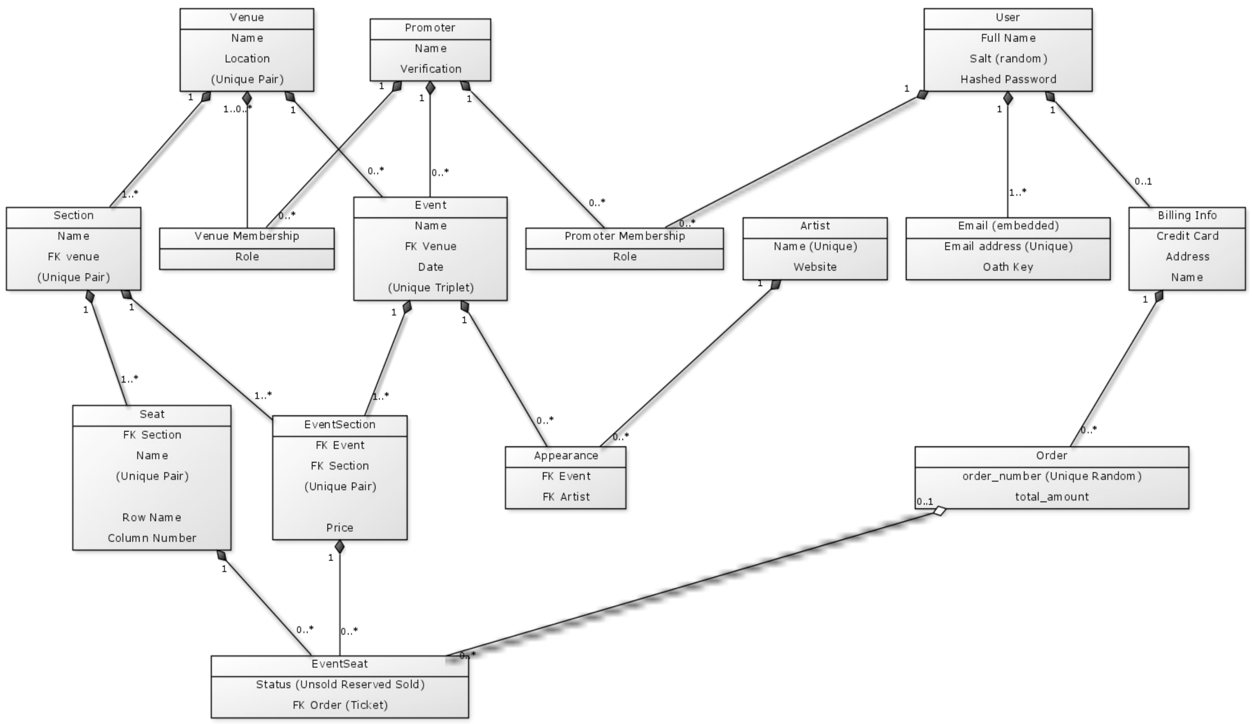


Figure 2. QwikStubs Class Diagram

1.3 Users and Characteristics

QwikStubs is targeted at three basic types of users -- the casual fan, the die-hard fan, and the venue promoter.

Casual Fan - a user who occasionally likes to buy tickets to events or browse events at their leisure. These types of users should be easy to satisfy, and the service should be easy for them to use.

Die-Hard Fan - a user who will purchase tickets as soon as they go on sale. They will constantly monitor the site to view ticket sale statuses and will most likely be the user who begins purchases as soon as an event's countdown hits zero. The real-time aspects of the QwikStubs interface will be necessary to satisfy these users.

Venue Promoter - a user who represents a specific venue and/or vendor who wants to sell tickets. This user utilizes QwikStubs to more efficiently sell tickets to their events. Ease of use will be an important factor to satisfy these users.

1.4 Operating Environment

We want to leave some room in adapting our environment as we go so that we have an optimal production environment. Right now our operating environment will be:

- Amazon AWS EC2

- Running Ubuntu 12.10/OS X 10.8.2
- Ruby 1.9+
- Rails 3.2+
- Ngnix 1.2+
- MongoDB 2.2+

Refer to the blue text in Figure 5 below.

1.5 Design and Implementation Constraints

Running a deployment of our app should be relatively straightforward, we can horizontally scale our hardware as needed thanks to AWS, and therefore, hardware is not really a concern. Our inter-process interfaces will all be over HTTP, and use JSON to communicate. We will use Rails, EventMachine on top of MongoDB to implement the backend, and will use Javascript in the frontend to render and manipulate the UI. We have a daemon running in parallel to our web interface, which allows us to express asynchronous parallel operations, that can occur while a request is being made. Since we have already formalized our use of Rails, almost all code will be written in Ruby, allowing us to express concise code. All of the components will talk exclusively over HTTP, using JSON as our serialization and data exchange format.

We will rely on Rails' built in security features to guard against web vulnerabilities such XSS(Cross Site Scripting). Thankfully because of MongoDB we are immune to certain classes of web exploits like SQL injection. For the security features we do have control over, such as password storage, we use password hashing with randomized salts, one of the most secure ways to store passwords. We use BCrypt as our salting and hashing library. It is an expensive hash function, making dictionary attacks much more difficult, as computing the hash takes much more time than something like SHA1. Our design is focused around strongly modeling our interfaces, and keeping the concerns separated, so that we have a very modular architecture, that allows modifying the design as our experiences dictate. Our coding standards are going to follow typical Ruby and Rails conventions, and enforce a style guide to keep the code base consistent and easy to read. As an open source project, we hope to host a demo version, but leave the software open and free to host, so that anyone can use it to run ticket sales.

2 Specific Requirements

2.1 External Interface Requirements

2.1.1 User Interfaces

The following user interfaces will be present in the QwikStubs system:

All of the resources in our program, such as users, promoters, venues, and events will have typical Create Read Update Delete (CRUD) interfaces. The following are all of the pages with which a user might interact:

- *User Login*
  The User Login page will allow QwikStubs users to login to the system using their unique email and password combinations, their facebook account, or google account. This is shown in Figure 3.

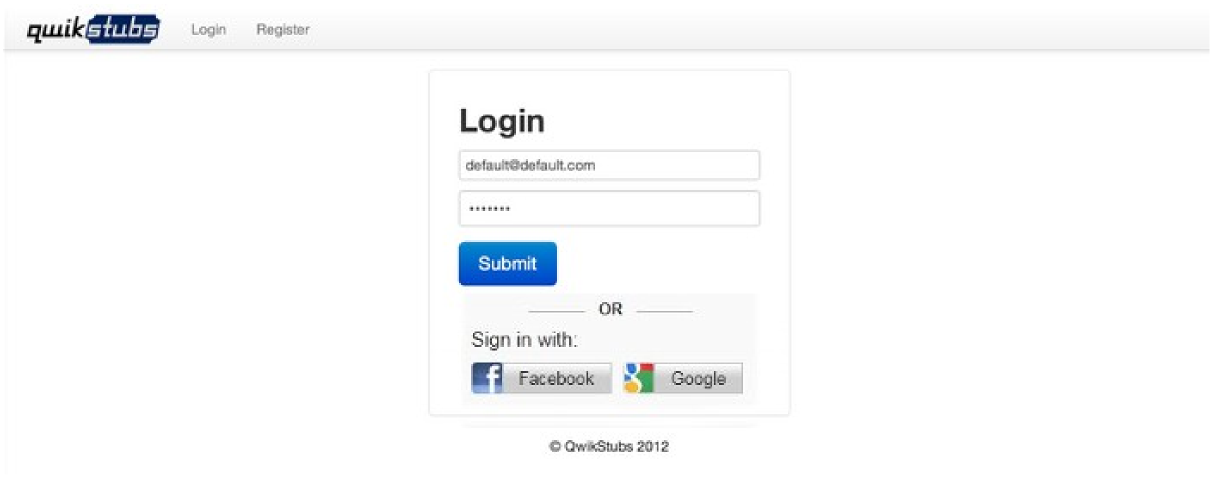


Figure 3. QwikStubs Tentative Login Page

- *User Registration*
  The User Registration Page allows users to create an account with QwikStubs by either logging in with their facebook or google accounts, or providing the site with a name, email address, password, and password confirmation. This is shown in Figure 4.

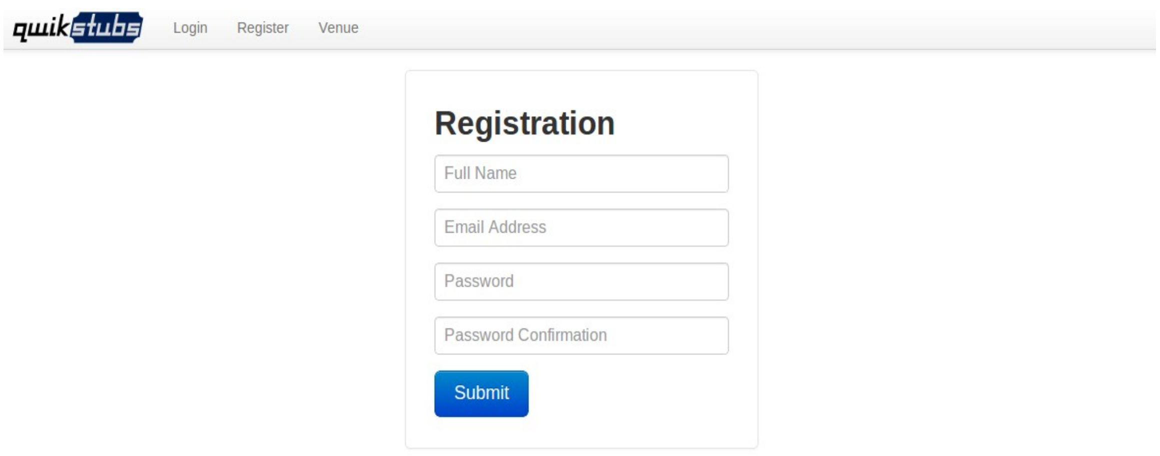


Figure 4. QwikStubs Tentative Registration Page

- *User Account Information*
  The User Account Information Page will be where users can view and edit their personal account information and will also be the area of the application where the user can enter in their billing information for future ticket purchases.
- *Event Browsing or Searching*
  This page will be where users can see upcoming events in an organized layout.
- *Event Creation*
  The Event Creation Page will be utilized by verified promoter users and will be where they enter in the details of an event.
- *Venue Creation*
  Venue creation will be done by promoters.

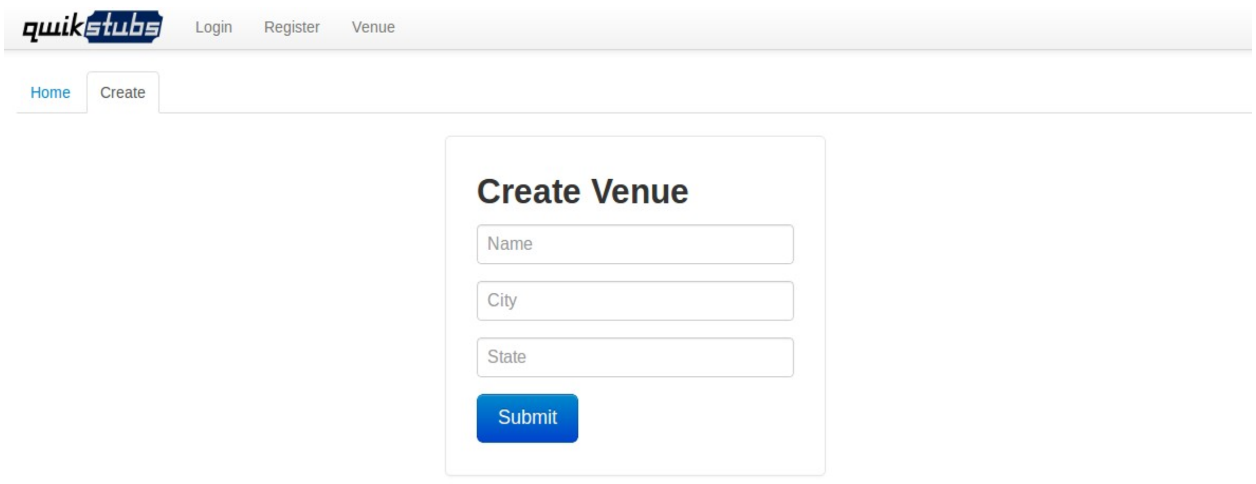


Figure 5. Creating a new Venue

- *Ticket Selection/Ticket Purchase/Venue Seat Monitoring (Updated in real-time)*
  This page will present all the information regarding the seats for an event. From the fan's perspective, this will be a view of the specific sections and seats within a venue, and colors will distinguish which seats are are available. Once the purchasing period is over,

the event manager will be able to see which seats fill up when fans arrive and enter the venue.

2.1.2 Hardware Interfaces

We have almost no hardware interfaces, since it will all be handled in a standard web browser. The only hardware interface will be for the scanning of QR codes to accept tickets at a venue. All that is necessary is a device that has a camera to scan the QR code on the screen of another mobile device. Any smartphone or tablet will most likely satisfy this requirement.

2.1.3 Software Interfaces

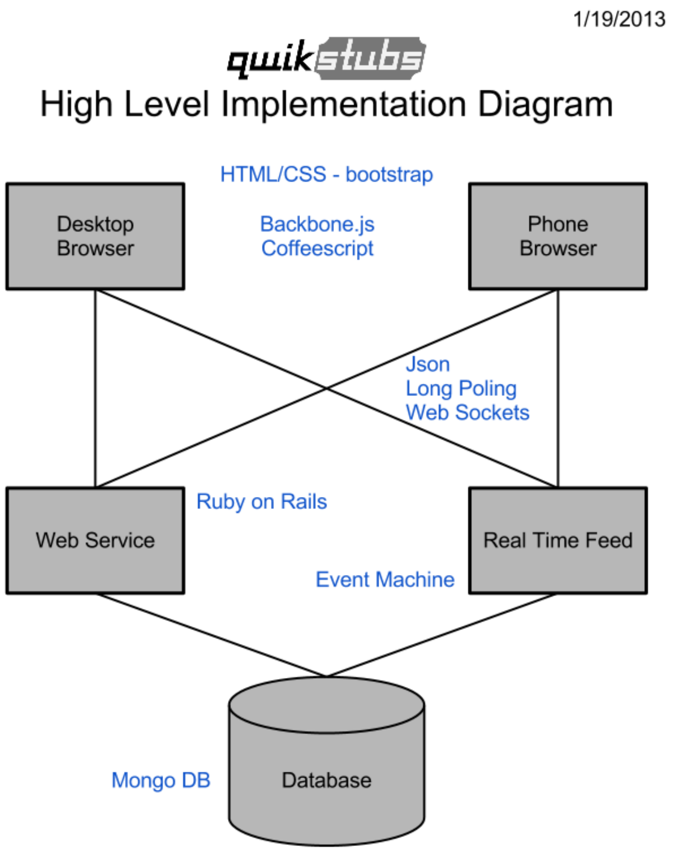


Figure 6. QwikStubs Implementation Diagram

- Web browser UI(HTML, CSS, JavaScript, CoffeeScript)
- Rails (Model, View, Controller)
- EventMachine(an interface from events -> actions)
- HTTP routes
- JSON data transfer
- AJAX (passing requests over AJAX)
- MongoDB (through queries)
- rQRCode - a Ruby library for generating QR codes

Our software interfaces are all loosely coupled and the components mostly use our communication interface to interact. Our Rails web service provides its interface as HTTP routes. Our EventMachine daemon waits for events and triggers the appropriate handler. MongoDB provides an interface in terms of queries. The browsers communicate over HTTP, and retrieve all content with it, and makes back channel requests over websockets and AJAX.

2.1.4 Communications Interfaces

Our communication interfaces are incredibly simple, since we are taking the approach used by most modern companies on the web. We will use HTTP, and HTTPS for serving all content to users in the browser. We will use HTTPS for interactions with sensitive information, like purchasing, and logging in. All of our services will either talk programmatically or by sending JSON messages over HTTP. There is little to no overhead in using HTTP instead of a raw TCP socket, as well as HTTP enabling the easy definition of interfaces. Our communication will be relatively limited, it will occur between the Rails app, and EventMachine. Both the application and daemon will also talk back and forth with the browser using JSON over websockets.

3 User Stories

Stories for the initial prototype
Implementation Difficulty

[Easy]	- can be done in 1 day
[Medium]	- can be done in 1 or 2 days
[Hard]	- can be done in 3 to 5 days
[Hardx2]	- can be done in a week or more

~~crossed out~~ = stories completed by version 1.1

[Hard]
~~As a user I can login and logout.~~

[Hard]
~~As a user I can register.~~

[Medium]
As a promoter I can browse my event listings.

[Medium]
As a fan I can select seats.

[Hard]
As a fan I can reserve selected seats.

[Medium]
As a promoter I can create and edit events.

[Medium] **As a user I can add and remove users to/from promoters I manage.**	[Medium] **As a fan I can browse events.**
[Hard] **As a fan I can buy reserved seats.**	[Hard] **As a fan I can search for events**
[Hard] **As a fan I can print tickets**	[Hard] **As a fan I can add billing information**
[Hardx2] **As a promoter I can create and edit seating for venues.**	[Hardx2] **As a fan I can see event seating availability in real time.**
[Medium] **As a promoter I can create and edit venues.**	[Easy] **As a promoter I can attach a venue to an event.**
[Medium] **As a user I can change or add email addresses.**	[Medium] **As a user I can create and edit promoters.**
[Easy] **~~A user can be a fan.~~**	[Medium] **Create seed data for a testing environment (partially complete)**
[Hardx2] (new) **A fan can use a QR code, which will be scanned at**	[Hardx2] (new) **A Promoter can see visualizations of ticket**

a Venue, to be accepted as a ticket	**sales data**