Alex Hamstra
Jared Roesch
Kyle Jorgensen
Ben McCurdy
Brittany Berin

Vision Statement

1/22/2013

QwikStubs is a web-based ticketing system that efficiently relays information regarding the selling and purchasing of tickets in real time to its end-users.

Description of Features

QwikStubs, through the use of cloud hosting, will provide its users with a slick user interface unlike any other online ticketing service available today. It begins by simplifying the interaction between the two parties involved with ticket sales – the venue managers and the fans.
QwikStubs initially provides the venue manager with the following services:

- Entering an upcoming show
- Setting the ticket prices
- Setting the ticket sales start date/time
- Publishing the show for fans to see

Once the venue manager sets the above information for a specific show, the fans are able to do the following:

- Browse for a show using the artist name, venue, or date
- See the seating plan for the venue with the varying ticket price levels
- See a live countdown to the start of the ticket sales and as tickets are purchased

This all takes place through an internet browser on a personal laptop/computer or a mobile device. Once the countdown finishes and ticket sales begin, the fans can begin choosing seats to place on reserve. Where this service differs from other existing ticketing agents is that it allows fans to visually see the inventory on the venue's seating map in real time. As soon as tickets are sold to others, this will be indicated on the venue map and the fan will have to go on to select other seats that are still available. Once the fan selects and reserves their desired seats for the venue, they will have five minutes to make the purchase before the seats become available to the public again.

Once the tickets are successfully purchased, QwikStubs relieves the hassle of getting into the venue by sending a ticket directly to the fan's email. This ticket will have a secure barcode that the fan uses by either printing out the ticket or displaying the barcode on their mobile phone upon entry to be scanned by venue employees. QwikStubs will store the name and age of the fan associated with each ticket and will take further security measures to prevent tickets from being reused. The manager of the venue, in turn, will be able use QwikStubs to visually track how the venue fills at the time of the event.

Implementation

Our focus is to make a service that can visualize these actions in real time. In order to achieve this goal, we are going to have an aggressive prototyping schedule while unifying our development process and tools from the prototyping stage up to the final release. To satisfy our real time

requirement, we are going to use several technologies to make both the UI and backend as fast as possible.

Backend

The backend will be a two piece design, using Rails as the core of our service to handle the data exchange, and authentication. In conjunction we will use a daemon built around EventMachine to handle the receipt and dispatch of asynchronous events. Along with WebSockets this allows us to provide asynchronous updates to the UI, as well as push notifications. We can then push most of the UI generation and update into the browser using modern client-side MVC techniques.

For our database we want to utilize a fast non-relational data store in order to meet our real time requirements. MongoDB seems like the most promising candidate, of course we leave room as per the Agile approach to adapt as we build. MongoDB has a much faster read/write time then a traditional database allowing us to do things like keep accurate ticket counters, and carts in real time. MongoDB also being schema-less allows extremely rapid prototyping as it doesn't require us to write and apply migrations. We are also leaving the option to move some or all of our data into a SQL database such as PostgreSQL or MySQL, if we feel that the relational nature would be helpful.

Frontend

The frontend will of course be utilizing Javascript heavily. Additionally, we will use Coffeescript to ease the burden of practicing 'good' Javascript development practices, as it is easy to do 'bad' things in Javascript without realizing it, and after speaking with our mentor, we feel that Coffeescript's constructs encourage better practices. Our approach with a client model view controller allows us to do things that previously required a page refresh in the past, which broke the experience of continuity. Our number one goal is for ticket purchasers to *know* if they have tickets, and to be able to inform them when the situation changes without needing to refresh and repeatedly seek the information. This MVC approach pushes all the computation of rendering to the client's computer, allowing us to do less on the server, and reducing the latency of pushing updates across a wire while synchronously re-rendering the interface. Instead, we make an asynchronous request for data. When we get the updates or when we are pushed an update, we re-render. This approach provides a *much* faster UI experience, and makes it much more enjoyable to use. The frontend approach follows our number one goal of be fast, and be accurate. The best way to implement this seems to be Backbone.js, as well as some familiarity on the team, it allows us to maintain data persistence, and asynchronous updates back to the server in the background(also allowing lazy fetching of data) with very little work.

We are going to adapt an aggressive Agile development process, accompanied by an adapted test driven development methodology, where we test before, during, and after each piece of development. There are a number of promising testing frameworks in Ruby, we will mostly like use the testing built in with Rails.