Spiremedia ShowFro

CSM Field Session 2013

Naomi Plasterer and Brandon Bosso

Table of Contents

Introduction

Client Description

Founded in 1998, Spiremedia is a privately owned and self-funded consulting firm based in Denver, Colorado that architects, designs, and develops web and mobile solutions for the world's top companies. Spiremedia offers strategy and UX design, artistic design, and development for web and mobile. Their mission is to make technology more meaningful, more useful, and more successful. In the past, Spiremedia has worked with such clients as Dish Network, Volkswagen, Dell, and Toys R' Us.

The company consists of a small team of professionals that are very good at what they do. Their user experience, or UX, department has several sections which involve wireframes, user stories, and strategy. Many clients come to Spiremedia with a simple idea that the UX department turns into a functional set of documents. The documents from the UX department then get handed to the art department. The art department consists of four designers who can write fluid css and appealing interfaces. The next step is development. Spiremedia only has a few full time developers and most of the developing is contracted to local self-employed developers. This is the stage where the product takes form. Development is set up in an agile form with scrum meetings every morning. The setup is often composed of six, two week sprints with the last of these reserved for quality assurance. The quality assurance is usually done by college students Spiremedia contracts on for work. From start to finish, a project at Spiremedia can take anywhere from six months to over a year.

Product Vision

ShowFro was the brainchild of Spiremedia's CEO, Mike Gellman. His idea was to create a website where users could search for any event in the United States and view data predicting how crowded an event would be. A number between 1 and 10 would be used to let the user know when the event was expected to sell out and when they should purchase tickets. He wanted ShowFro to be the ultimate online prep tool for any music, sports, or theater event one may be planning to attend. It would provide predictive data on when to buy tickets, when to show up, when to leave, and what to do before and after an event. It would automatically generate a page for every music, sports, or theater event. Users would be able to find these pages on search engines when looking up an event. ShowFro would also serve as an objective pricing guide, a la Kayak, for after-market tickets. Through the use of Google AdSense and affiliate programs, the site would be monetized. The key was to provide users with comprehensive, easy to understand information making ShowFro a useful part of the event going process.

Requirements

Functional

The website must allow users to search for events by venue, artist, or city. After a search has been executed, the website must then display the results and allow the user to view a specific page for each event.

- Users can search by venue, artist, or city
- The website takes a user's search terms and queries Pollstar.com's API for any matching events
- These events must be displayed on a results page with the following information
    - A link to the event specific page
    - A list of performers who will be at the event
    - The event's venue
    - The event's date and time
- A page must be created for each result
- These pages must be stored in a MySQL database allowing for future access

When a user selects an event to view, the website must:

- Display detailed information about the event including:
    - The event name, date, time, venue, and performers
- Determine a fair ticket price the user should expect
    - If not enough information is available to calculate this, a message stating this must be displayed to the user
- Calculate a number predicting how crowded an event will be based off a number of social media metrics about the event's performers
    - This number must be normalized on a scale of 1 to 10
    - This number will be used to generate a statement explaining when the event is expected to sell out
    - This number and its description must be displayed to the user
    - If the needed social media data is unavailable and the number cannot be calculated, a message stating this must be displayed to the user
- Google AdSense ads will be added to each page allowing for the monetization of the site
- Affiliate programs will be used to provide users with the resources needed to purchase tickets and to generate revenue
- Google Analytics will be used to track the site's traffic
- Search engine optimization will be implemented to ensure that users can find a link to the site when searching for an event on Google or similar search engine

While no formal performance requirements were laid out, a few were implied

- The site must load pages at a reasonable rate
- The site must perform regular database maintenance

Non-Functional

- The Drupal content management system will be used to construct this website
- RESTful API queries will be made with PHP
- HTML and CSS will be used in conjunction to produce each page
- A MySQL database must be used
- Event details will be returned from Pollstar.com
- Fair ticket prices will be determined using information returned by the Seatgeek.com API
- Social media data used to generate the ShowQuo number will be obtained from the Nextbigsound.com API
- Data returned by each API will be in a JSON format
- An Amazon EC2 Cloud Server will be used to host the site
- Bitbucket will be used to share code and push to the client server.
- The website must be left open to modification and addition

Risks

- We may encounter legal restrictions when accessing data from various APIs
- Slow interface because of too many API calls and database queries.
- Data points may be corrupted, missing, or hard to manipulate.
- Events existing in the Pollstar database may not be found on Seatgeek

System Architecture

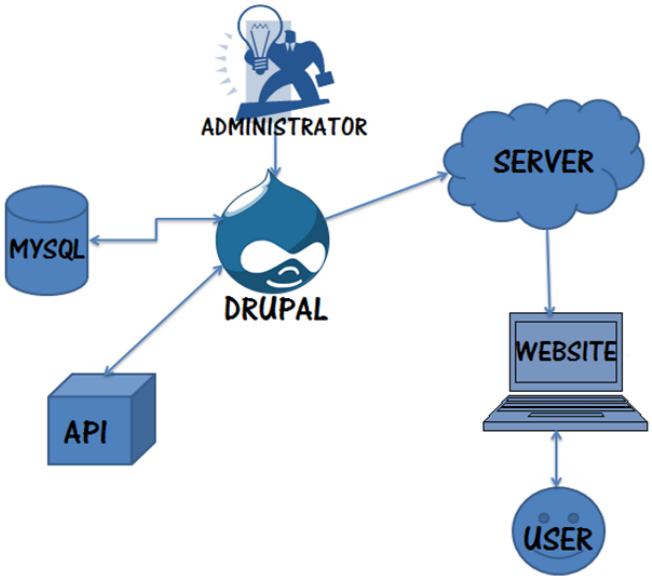

Figure 1: ShowFro Architecture

User

The user is the main actor in our system (*Figure 1*). The user inputs the artist, venue, or city into a search box on the user interface. They are the key component in let the website know when to query the server for data.

User Interface/Website

The user interface acts as a buffer between the user and the server. When a user searches for events, the user interface records this action and sends a request to the server. The information that the server returns is then displayed to the user.

Server

The Amazon EC2 Cloud server acts as a remote access point for the information on our site. It allows any machine to request data. When this data is requested, specific functions stored in Drupal modules are invoked producing the information to be returned to a user's machine. It is here where our database is stored.

Drupal

Drupal acts as our content management system. When a user runs a search and the server requests the matching information, Drupal finds any matching information in its database. Drupal modules then send a RESTful call to the Pollstar API to retrieve and store new events in its database. Existing events are updated as needed. A results

page is crafted with the matching information and returned to the server. When an event page is accessed, another database query is executed. All of the details regarding the event are requested and placed into a page template. Additional calls to the Seatgeek and Nextbigsound API's are executed in order to generate the ShowQuo and ShopFro numbers. Drupal is also used to perform database maintenance and acts as an interface for administrators. Once a day, Drupal runs a function to scan through all events stored in its database and delete any that have already occurred. This prevents the database from growing too large and increases the site's performance.

API's

Pollstar, Nextbigsound, and Seatgeek all provide public application programming interfaces. These provide various functions necessary for Drupal to retrieve information. When a query is received, these API's return a JSON file containing the requested data. Each API returns a very specific set of data. It is from this data that an event's details, ShowQuo, and ShopFro are obtained.

Pollstar is used to create a list of events. An event's name, date, venue, address, and list of performers can be obtained from this information. Seatgeek provides information about ticket prices for each event. This data provides our ShopFro number. The Nextbigsound API provides the social media data used to produce the ShowQuo measurement.

MySQL Database

The MySQL database is where all of the website's information is stored. Each event page has its details placed into a series of tables. When a page is requested, the database returns any necessary information needed to display the page. This includes theme options, URL aliases, and event details. As new event pages are created, they are stored here. A wide range of Drupal is also stored here.

Administrator

The administrator has direct access to a number of settings for the Drupal content management system. Administrators set which theme the site uses, patterns that page urls follow when they are created, caching options, and what information is displayed in each portion of the page. Drupal provides administrators an easy to user interface for editing the website content without writing any php or html.

Technical Design

Database Schema

By making the choice to use the Drupal content management system to build our site, our input on the database format was very limited. Drupal uses content types to serve as a template for each type of page it has to display. When a content type is created, fields are added that serve as placeholders for the page's specific data. In our case, we created an event page content type, which had a field for each detail about an event. When an event page is created, these fields are populated and the page information is stored in the database. Rather than having one table that contains every field for each page, a node table is created storing the page id. This id is then used to refer to separate tables for each field where the respective data is stored (*Figure 2*). By doing this, the database remains easy for developers to read and manage. It does, however, result in a high amount of database queries.

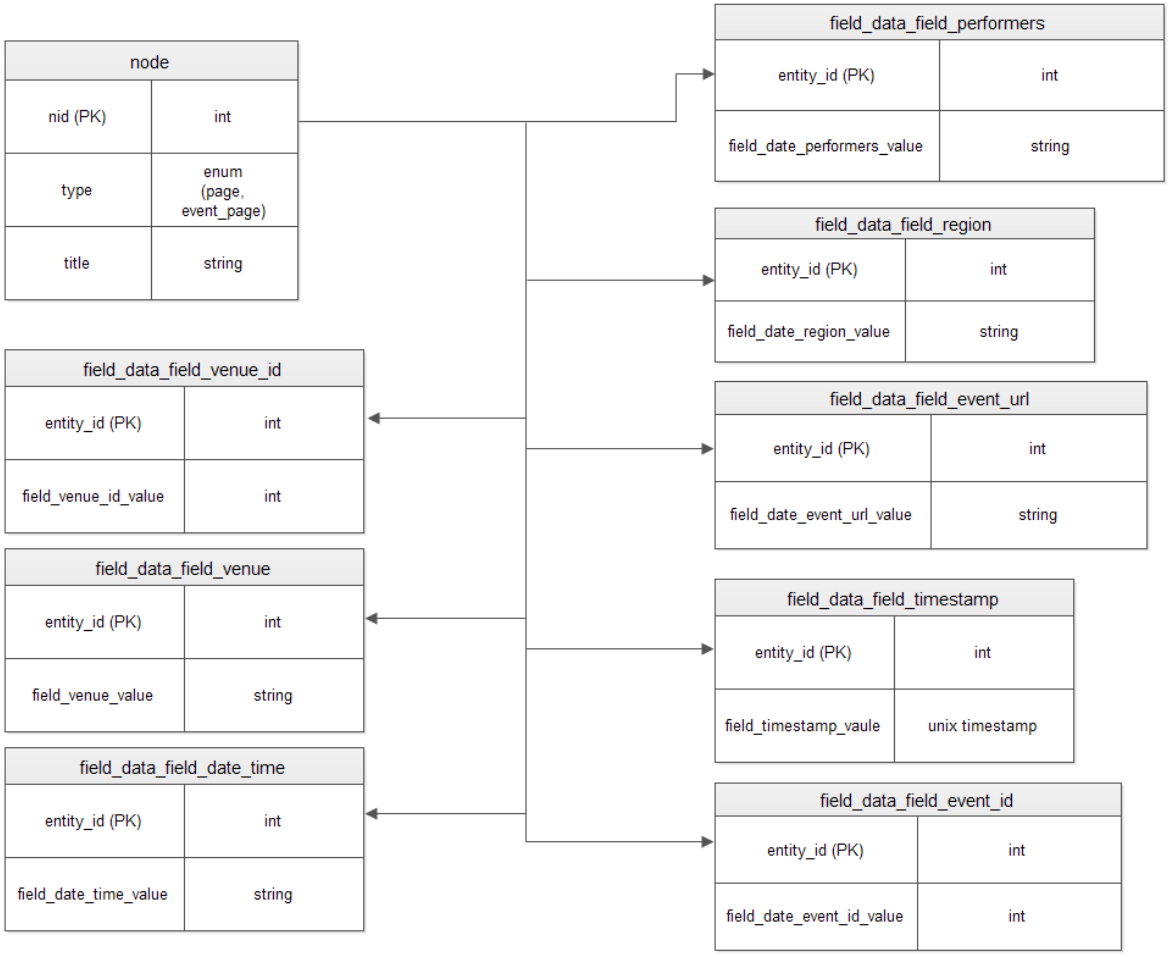

Figure 2: ShowFro Database Scheme

Search

One of the main features of our website was to allow the user to search for events. We decided that users would likely expect to be able to search for these events by artist, venue, or city (*Figure 3*). In order to accomplish this, we had to override Drupal's main search functionality. By default, the Drupal search takes a user's input and scans all existing content for matching keywords. These results are then displayed to the user. To implement our own search function, we had to write a custom module that would interact with the Drupal interface.

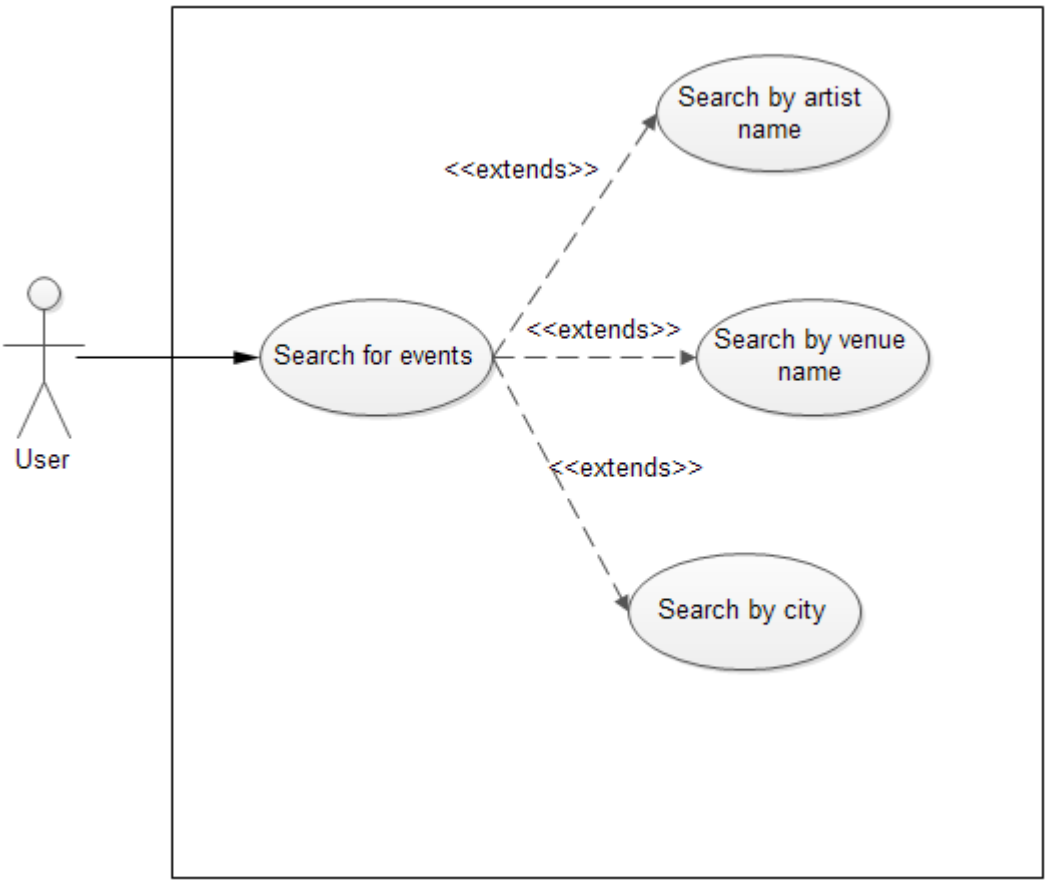

Figure 3: ShowFro Search User Case

Per Drupal requirements, PHP was used to create this module. Drupal relies on specialized functions known as hook to accomplish most things. In our custom search module, we had to implement many of these hooks in order to override the search box. Instead of searching only existing content when a user searched for events, our module had to request data from the Pollstar.com API. Since we had no previous knowledge in language analysis, we were limited with how much we could analyze the user's input. Rather than scanning the text and determining whether it was a venue, city or artist, Pollstar is queried for any entities that match the input.

When a search is executed, any events existing in the database will be retrieved first. Pollstar is then queried and returns a series of entity id's with which we could search for events. For each city, venue, and artist that is returned, another query is executed for any upcoming events. Pollstar returns the requested data in a JSON format. This data is then translated and placed into a series of event objects. These objects store the event name, date, venue, city, and artists. A link back to the event's page on Pollstar.com is also stored as required by the Pollstar API's terms of use. Any of these event objects which were previously stored in the database are updated as necessary and new ones are added. These results are then displayed to the user with a link to an event specific page.

Since our knowledge base about the subject was limited, we encountered some performance issues with our search. We had planned to include a pager so that only 10 results were displayed per page and the user could switch pages as desired. This posed to be more difficult than planned. We made the choice to place all results on the same page and focus on the rest of the site's functionality. For most searches, this was fine. However, when a search returned a large amount of matching events, it took too long to load them all and frequently timed out. As a temporary fix, we decided to only have each Pollstar query return 20 matching results. While this caused our search to be less comprehensive, there was a noticeable increase in the performance. Because of time constraints, we were unable to go back and implement a pager and thus had to keep this limited search function.

ShowQuo

An important part of our website was the ShowQuo value. When designing our website we had made several initial decisions on how we wanted it to display and what we wanted each value to mean. The ShowQuo value is supposed to predict the chances of an event selling out. For example, an event that had a ShowQuo number of ten meant that the event was going to sell out within minutes of going on sale. We defined all ten of the values manually in a switch statement and got the text approved by our client.

The ShowQuo number utilizes Nextbigsound.com's API. Nextbigsound is a website which predicts who will have the next top of the chart song. They have data for almost every artist and provide a simple list of the current top 50 artists. The data points provided in the API included the artists plays, fans, and likes across several social media networks; including Facebook, Twitter, Instagram, YouTube, LastFm, MySpace, and more. We decided to use certain information from only a few social media sites for our original algorithm. We pulled plays from MySpace, YouTube, and LastFM. Similarly, we pulled fans from Facebook, Twitter, and MySpace. This information was totaled and averaged for use in our algorithm.

Our algorithm consisted of several core components. First it calculates an artist's average rating and normalizes it on a scale from zero to one. We referred to the

average rating of an artist as p. To find p, we divided the average number of plays an artist has over the average number of artist's fans. This number was equal to x. X is then normalized over a scale from zero to one and that number is used in our final algorithm as p (*Figure 4*).

$$X = \frac{AVG\ PLAYS}{AVG\ FANS} \qquad P = \frac{1+(X-1)*(1-0)}{(1000-1)}$$

Figure 4: ShowQuo Calculate P

Similarly, our algorithm used the summation of the number of fans. This number was referred to as n and was just a sum of all the fans from MySpace, Facebook, and Twitter. The final number we used in our algorithm was just a simple constant z. Z was equal to 1.645 so that it would have a confidence interval of ninety percent. The final algorithm (*Figure 5*) then took all of these numbers into account and calculated a number between one and a million. This number represented the popularity of the artist. So for example, Justin Bieber would be a million while a garage band might be a one.

$$POPULARITY = \frac{P + \frac{Z^2}{2N} - Z * \sqrt{\frac{P(1-P) + \frac{Z^2}{4N}}{N}}}{1 + \frac{Z^2}{N}}$$

Figure 5: ShowQuo Popularity Equation

The popularity number is then normalized down on to a scale from 1 to 10. This number is called the ShowQuo and is then fed into the switch statements; allowing for the page to apply the appropriate text to the number.

Design and Implementation Decisions

Decisions

Languages

When writing code for our custom Drupal modules, we were required to use PHP. This included the .info and .module files for each module. When we needed to get data from one of the API's, we used php's curl functions and then wrote custom code to parse the JSON results. The template files were written using a mix of PHP and HTML5. To style our site we used very basic CSS. These languages worked well in conjunction with each other. PHP can be used to generate HTML5 so it worked perfectly for what we required.

JSON data was much easier to work with than the XML we had initially planned on. PHP contains preset functions that can convert JSON files into a series of objects. When we requested information from one of the API's, these functions were used to create a series of event objects. Each of these objects then contained the information we needed to display an event to the user in a simple, easy to access format.

Framework and Database

We chose to build the website using a Drupal framework for multiple reasons. First, it is widely used in Spiremedia, so there were many developers willing to help and ready to answer any questions. We also chose Drupal because it is open source, which allows for custom modules that are easily modifiable and very easy to implement. This saved us time by allowing us to just import and implement modules that had already been created. Drupal also allows for automatically generated pages from a template. This was important to us because we generate multiple pages just from data pulled from the API. Because we chose to use Drupal we were required to use a MySQL database, which houses configuration, information, and other important files. The MySQL database helps with caching information which came in handy for allowing pages to load quickly and not make too many calls to the API.

Data Sources

We decided to use Pollstar.com for our main source of event data. It provides a wide variety of genres and all of the information about each event that we were looking for. To obtain our ShowQuo metric, we used Nextbigsound.com since it provided data about the artist's popularity. We then pulled certain data points from Nextbigsound and formulated a complex algorithm for calculating an artist's popularity using social media sites. Seatgeek.com is an open API so we were not required to apply for an API key. Seatgeek offered minimum, maximum, and average prices for tickets. This is where we pulled our ShopFro metric from.

Wireframes

We went through several iterations of wireframes with the client. Since our project started from just an idea we had to use our user stories to pick what we wanted on each page. We did this through a process of identifying what gave our website value. This allowed us to narrow down our content and keep our layout clean and concise.

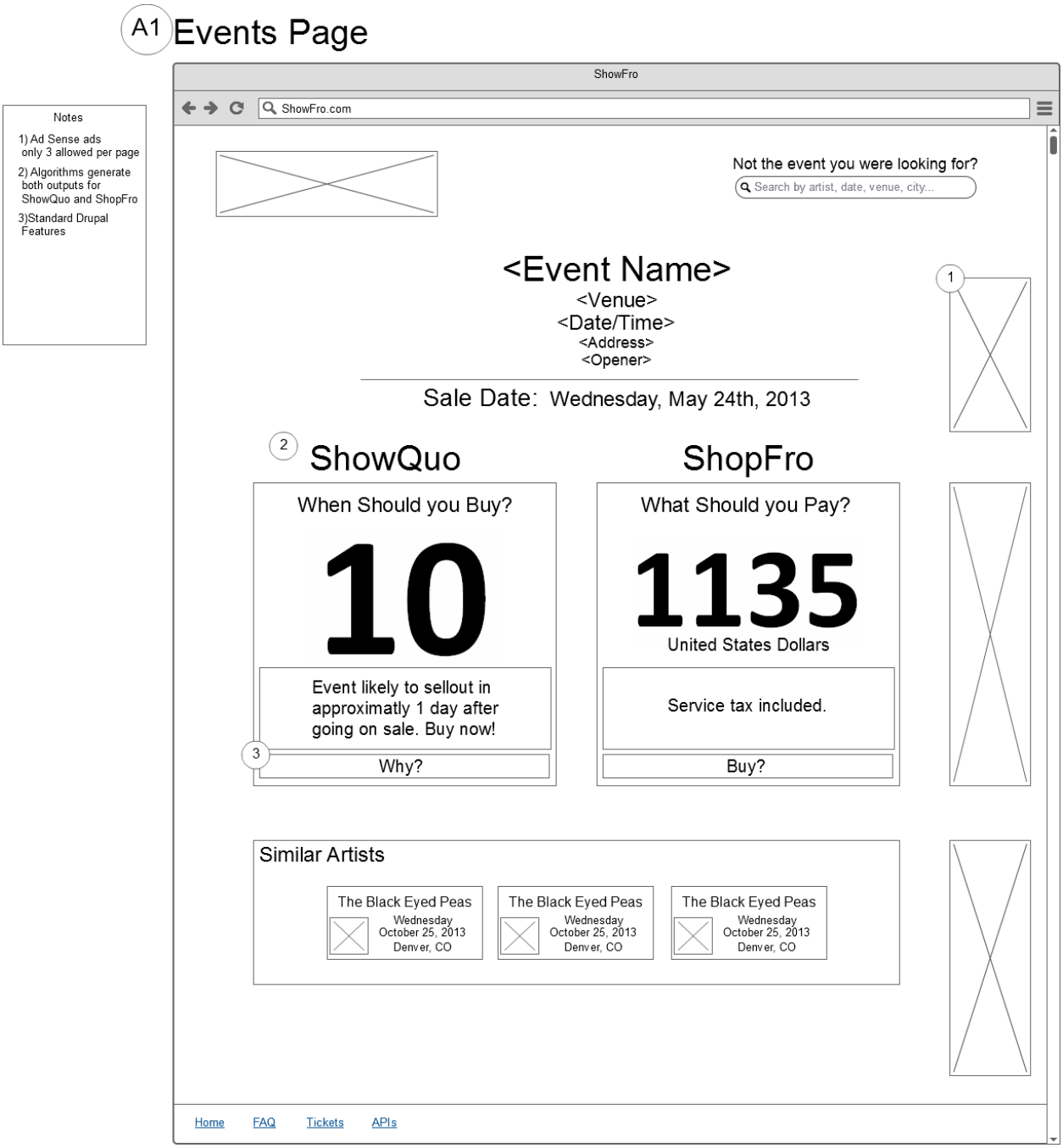

Figure 6: Events Page Wireframe Version 4

We decided on our final design (*Figure 6*) because it presented our two most important pieces of information clearly and obviously on the page. The ShowQuo and the ShopFro are the two pieces of information that make our site valuable and unique from other event sites. We chose to make the numbers large on the screen so a user in a hurry could easily identify the two pieces of information they were looking for.

Lessons Learned

While working on this project, we encountered some major time constraints. We had initially planned on implementing a wide array of features including a trending artists page, alerts when tickets for an event went on sale, and sharing of events through various social media outlets. We quickly realized that since neither of us had any prior experience with the technology we used that many of these features would take too long to implement and were thus out of scope. If we had initially planned to not include these features, we would have had more time to fine tune our project. In the future, we now know to manage our time better and to keep the first iteration of our program minimal.

The Drupal content management system has a very steep learning curve. Since we were required to do extensive research in order to implement even the basic aspects of our project, we had to learn how to teach ourselves to use this system. Through doing this, we gained valuable skills regarding how to utilize existing, open-source code and what types of resources are useful when learning new programming skills.

For the first week that we were working on this project, our client was unavailable due to personal issues. Because of this, we worked with other employees at the company in order to come up with a set of requirements, user stories, and wireframes. We began to implement these before having an official meeting with our client. When we did however, we realized that he had a different vision than we had anticipated. This required us to really embrace the agile programming method. We had to learn how to adapt to changing requirements and thus adjust our development schedule.

Results

The goal of this project was to create a website where users could search for concerts and other events. The site would pull data from an API (application programming interface) and then print and translate that information to tell the user whether an event was going to sell out. Specifically, a page would be created containing details about a given event. These included the event name, date, time, venue, address, and a list of performers. Each page would also display a ShowQuo and a ShopFro measurement. The ShowQuo was a number between one and ten that gave users insight into how crowded an event would be, if it would sell out, and when to purchase tickets. This worked very closely with the ShopFro, a measurement of fair ticket price for each event. Each event page would also contain Google AdSense, search engine optimization, and analytics tracking.

Our final delivery met many of these requirements. Users are able to search for events based on city, artist, or venue. When a search is executed, matching results are passed to the user with a link to each event's specific page. These pages display all of the required details about the event. The ShopFro number was implemented and gives the user an idea of what they should pay for tickets. The ShowQuo, however, differed a bit from what was initially requested. Rather than being calculated based on an event's popularity, it was based off of an artist's popularity. This pulled various data points about the performing artist's social media outlets. We did this because of an unavailability of information that we were able to access. None of our chosen API's provided data about how many tickets had already been sold for an event and the rate at which they were selling.

Due to time restraints, priorities had to be set and a few of these features had to be left out of our final delivery. Event pages do not display Google AdSense. They also do not implement any sort of analytics to monitor traffic and inform our client of the most used features of the site. Very basic search engine optimization was implemented using an open source module for Drupal. However, this does not provide the level of SEO that our client asked for. We were also unable to enter into any ticketing outlets' affiliate programs preventing our site from generating the revenue as requested.

We tested our final project in several browsers including IE10, Chrome, Firefox, and Safari. It was very important to clear caches often because bugs often hid themselves under cached pages. We identified, logged, and fixed any bugs we could find across browsers, and concluded that each browser had one or two small differences. We also tested our data points to ensure what we thought we were pulling from the API was actually being returned. We did this by searching for events from our page and our API to make sure they matched. Similarly, we tested our algorithm for calculating our ShowQuo. We hand calculated our ShowQuo number for several artists and then called these artists through our website. We compared to make sure the

ShowQuo numbers matched and printed extra data to make sure previous steps were being calculated correctly as well. Our testing helped to clear out many bugs and issues and increase the efficiency of our site.

Appendix – Server Installation

We used Bitbucket to manage our Git repository. In order to install our product on the server we had to set up the server to pull the code from our Bitbucket repository. First, we gained access to an amazon cloud EC2 instance where we housed our website. From here we SSHed into our server and pulled our code down from Bitbucket using a simple git pull command. After pulling the code down to the server you need to do a server restart from the command line to update the website. We then installed Drupal directly on to the server. The next step was to create a copy of our local database from which we then imported directly through Drupal on to the server. At this point the website should be up to date with the code on Bitbucket. By navigating to this link, http://ec2-54-234-92-135.compute-1.amazonaws.com/, the website can be seen live.