Webtech Report

Imna Malik n James Sewart

29-5-16

Contents

Abstract

The Smple (short for sample) web application is a visual geospatial music discovery service. Users can discover new artists playing in their area and sample their music using data gathered from the Internet.

1 Client

1.1 Style

1.1.1 General Layout

The Smple web app consists of a single page which is filled with a map from the Google Maps API. There are two panels: the search on the left, and the information (info) on the right.

The search panel consists of an input to type a query and two date pickers to choose the date range for events to be displayed. When a search query is made,

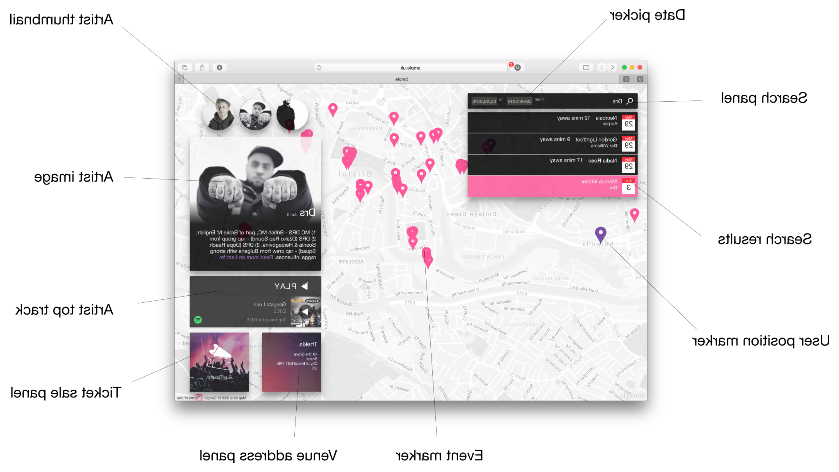

Figure 1: Basic layout description.

results are shown underneath with the date of the event, the band names that matched, and the time to walk there.

When an event is selected by clicking a marker or search result, the info panel slides into view from the right. At the top, the artist thumbnails are displayed. They are dynamically created depending on how many artists are playing in the selected event. Below, is the artist name, image, and biography for the selected artist. An embedded Spotify player is revealed afterwards if the play button underneath is clicked. Lastly is the events venue address and the link to the ticket-sale website which are contained in their individual panels.

There are two markers present on the map: pink and purple. The pink markers represent events and the purple is used to indicate the current user location which is determined by the browser.

If the map is moved or zoomed, the info panel will slide out of view. Additionally, when the info panel's content overflows the bottom of the page, it becomes scrollable.

1.1.2 Base Classes

The design of our layout is inspired by Material Design with it's iconic card styled boxes. Each panel is given a shadowbox class which gives the element a border radius, shadow, slight transparency, and a dark background colour. The search panel and all panels in the info bar are given this class.

1.1.3 Resizing

We use CSS media queries to resize the info panel stuff like the and basically and the seearch panel thats everything. The font resize thingy we use percentages and ems we basically use the media query thing we change the font size and have everything inside percentages so everythings dynamic

We use CSS media queries to change large scale layouts based on the screen size. There are three of these: mobile-sized, large, and extra large. These changes involve setting the widths of the main panels such that they look good on the screen size. For the mobile screen this means the panels are full width and stack on top of each other. For the large size the info bar takes up a third of the screen, and the search 40% of the screen.

For a more dynamic resize control we subscribe to the onresize event of the window and scale the whole documents font size based on the screen width. Most elements are defined using ems and so they all scale accordingly.

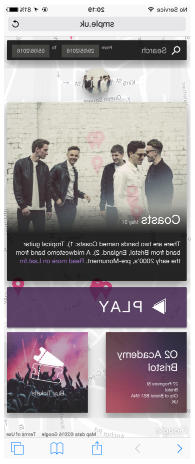
There are two bands named Coasts: 1). Tropical guitar band from Bristol, England. 2). A midwestemo band from the early 2000's, pre-Monument. Read more on Last.fm

Figure 2: Example layout when rendered using a mobile browser.

1.1.4 Info Panel

The info panel involves many different styling and scripting elements. The panel is animated in using the css transition property and setting the margin-right to 0 from -33%. By default moving this panel to the right would bring up a scrolling bar however this is disallowed by setting overflow to hidden for the body.

In order to get the desired scrolling effect we set the overflow-y to scroll which allows the contents to scroll within this div. The height is set to 100% of the screen height so the content will never increase the full page size.

The artist thumbnails are contained within an element with text-align set to centre. Each image is an inline-block which means they overflow and stay centred no matter how many artists are inserted. As the images they contain could be of any aspect ratio and we want the thumbnail to be square, we use the object-fit css property to keep the container box filled with the image inside. The containing box is given a border radius of 50% to make it circular, and the overflow is set to hidden to crop the image inside. The thumbnail is animated to zoom on hover using a css selector that transforms the scale of the container.

The missing artist picture was created by importing an image found online into inkscape with the image rendering mode set to smooth. Then an avatar silhouette was overlayed on top of it.

The second box down includes an element for the artist title, along with a paragraph tag for the biography. The title is overlayed onto the the artist image and has a gradient behind in order to make sure the text is visible. The title is pinned to the bottom of the image. The image width is set to the parents width so that only the height changes depending on the aspect ratio of the image. this height will push down the biography. Padding is set on the biography for visual aesthetic.

The next box contains a self made svg that when clicked, triggers a set of animations. The two triangles will turn 120 degrees with slightly different durations using SMIL animations, the background colour of the box fades from purple to grey, and the size of the box increases to reveal an iframe with a spotify player inside. These animations are reversed on second click to hide the player. when updating the info panel, the url for the iframe is updated with the current artists uri, this triggers that url to be loaded inside, causing the correct player to be displayed.

This svg was created using the inkscape polygon tool to make a filled triangle. Then by duplicating the triangle and enlarging it then excluding we get a stencil triangle. Then simply by making a same sized triangle and offsetting its psition we get the desired effect.

The final two boxes display information about the venue. The first displays the venue title and its address. Clicking this will open a new tab with the songkick venue page. The second box holds another self made svg of tickets. Clicking this will take you to the songkick ticket purchasing site.

Starting with a rectangle we rounded the corners. Making two squares for each end with rounded corners, we used the difference tool to cut out a semi-circle. By duplicating a rectangle with rounded corners 5 times and combining them using the combine tool for simplicity. Using this combined object we exclude the shape from the ticket. Then we tried duplicating the svg and using the outset tool to make one bigger, then aligning the two and using the exclusion tool we get the stencil ticket. This didn't quite have the desired shape so instead we scaled the second ticket and extruded these two. This gives near uniform line thickness. Then by placing the stencil over a copy of the original and using the difference tool we cut a shadow. Now by simply placing the stencil in the gap we produced the desired svg.

The background image for the ticket button was taken online and edited using gimp. First the image was cropped to be square. Using the colour balance tool we adjusted the shadows, midtones, and highlight to make the pink and blue stand out. Next using the hue tool we modify the pink to match the websites colour scheme. Other colours were reduced using this tool. Next using the curve tool we adjust the rgb channels so that the image is more aesthetically pleasing. Next we created a new pink layer and layer mask for that layer. Using the gradient tool set to black to white we add a gradient to the mask which adds a pink gradient to the image. This is done twice, once to add a pink gradient that is prominent at the top, and one with a black gradient that is prominent at the bottom to produce a vignette effect. Next using the fuzzy tool with feathered edges to select the people in the image and using the hue/saturation tool, the blues were adjusted to be more grey using the overlap setting to consider overlapping hue values. Next we added the songkick watermark as an overlapping layer, positioning it in the bottom right layer and resizing as desired.

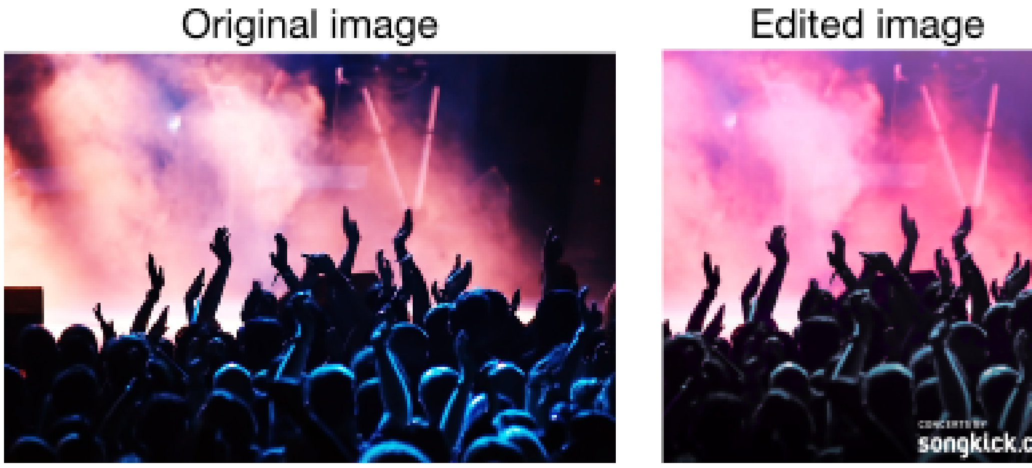

Figure 3: Before and after gimpifying.

When a user hovers their mouse over the lower two boxes, an animation is triggered to dim the image. This is achieved using an anchor html tag that is given the class boxClickUrl that expands the anchor to cover the entire parent. The dim effect is achieved by setting the opacity of the image to a smaller value, revealing the black shadowbox behind.

1.1.5 Search Panel

The search panel consists of a few permanent items. From left to right we have an icon that is from google material icons custom font. Then we have an input box with placeholder text to notify the user that it is a search box. Then we have a from and to date picker using the pikaday library.

The items are positioned using the float css attribute to get them next to each other. Padding is set in various places to get nice spacing and non breaking space are used to separate the from and to text from their date boxes. For the labels and date clickers a pointer is set to indicate to the user that they are clickable.

For the result list an empty list container is added for the results of a search to populate. A border-bottom is added to each list item to act as a separator indication. Using the css selector last-child we remove the last bottom border so that it only exists inside the list.

A custom date icon was created using two divs, one inside each other with classes red and white. The javascript that generates these creates the divs and populates them with a date and month.

The date filter uses a library for visualising calendars. Initially this didn't work in strict xhtml mode due to a bug in the library that we have fixed and files a pull request for. We initialise two of these calendars and subscribe to their click events, when a user clicks a date from the from calendar, we disable choosing date before that date on the to calendar, and vise versa.

1.1.6 Map

The map is from the google maps api with a custom theme set to match the theme of the site. We use the google map markers for our event visualiser but add our own custom svg in place of googles. A basic svg marker was found, and using inkscapes break apart tool, was edited to add a center hole. The transparency was also modified. The marker is animated using googles marker settings.

1.2 Logic

There exists one script for all the logic of the webapp. We subscribe to the window onload event as well as the google map callback. In the window load event we setup a few things, including setting up the callbacks for the svg button, initialising the websocket to the server, and initialising the date filter boxes.

When the map has loaded, we request the browsers location via the google maps api and setup the markers on the map. After both the map has loaded and the websocket has loaded we send the user location to the server via the websocket. This triggers the server to find nearby events to be send back to the client.

When an array of events arrive we populate our internal events array and initialise the marker for each event. A query is made to find the walking time to each of the events as well. Once this has happened the user can click on the markers to open up the information for the event, as well as being able to search for events using the search bar.

When a user clicks a marker, a class is swapped out on the sidebar so that it has a new margin-right property. As the infopanel has a transition property it will animate in. The same thing is done with the play button and spotify player but with css properties background-color and padding-bottom to change the background colour of the play button and to show the spotify player.

When filling the info panel with event information we need to create a thumbnail for each artist. There can be any number of artists in an event so we create a div with class bandimage for each artist and append each one to the bandpics div, organising them properly. The spotify player url is also updated to load the correct song.

For the search bar, the keyup event is subscribed to for the input box so we can get a string representing the current search each time a new key is pressed. This event is used to update the search results. First the current query string is used to do a sublime style fuzzy matching on each of the first and second artist names in every event. The fuzziness refers to lack of need for sequential characters to match and instead there can be skips before matching. To visualise the matches the result of the search wraps each matched character with a bold html tag so that when the search results are displayed it can be clearly seen which of the artist names characters are matching.

The results from the search include data such as the event date, first and second artist title, and walking time from the current location. For each result a list element is created that contains a visualisation of the date, ordering of artist names, and walking time.

These list items are appended to a html list element that exists in the base html.

2 Server

The server uses the provided script with modifications. The index is served as a static file and dynamic content is achieved using Websocket connections between the client and server. This design choice simplifies the file resource serving logic, reducing the dynamic parts to message passing.

Once the client has initialised the webpage, it sets up a secure Websocket connection to the server. A secure socket is preferred as non secure sockets are disallowed when viewing the page over https. The connection is abstracted into websocket.js such that server.js can simply pass a callback that is called when a new client query is made, with a parameter that can be called with the result to send back. server.js doesnt need to worry about managing connections.

The client will query the browser for its current geographical location and when given, will send this to the server along with a desired date range. The server will reply at some point in the future with an array of events.

The server will receive this data and use it to query the database using a geoNear query on a 2dsphere index, and a date range query on the event date. The results are sent to the client.

Websockets provide a stream of messages, where a message is either some binary data or a string. We serialize JSON for sending data between the client and server. When the client sends its position and desired date range we serialize a data structure that looks like the following.

```
{
    pos: {
        lat: Number,
        lng: Number
    },
    dateRange: {
        from: Date,
        to: Date
    }
}
```

Dates aren't parsed into JavaScript date objects when using a JSON.parse so these are handled manually for the database queries.

As well as giving the client the desired data we have, we also check whether the database needs updating. The data is collected using a variety of APIs. First we ask songkick for the area that the user is in, this allows us to check the database to see if we have checked this area at all/recently. If we haven't got an up to date set of events for this area, we start downloading the events for the area as well as querying last.fm for the artists biography and picture, spotify for a reference to their spotify page, and googles reverse geocode api to get the full address of the venue.

For each area we limit the speed of requests in order to avoid rate limiting. Any returned errors by the APIs are handled graciously. Once complete data about an event is retrieved, it is stored in the database using an upsert on the event title, to either update an already queried event, or insert as a new event.

A possible optimization of the client-server interaction process would be to determine what events have already been sent to the client in order to avoid sending them again if they only scroll the map slightly. This would allow for significantly less data to be transferred between the two and would avoid reprocessing every event on the client side on every query change.

3 Deployment

The server is written using some ES6 features and so requires a newer version of nodejs to run. By default the server connects to a local mongojs server with no special configuration added. This implementation favours root accessible vps' such as digital ocean or EC2.

The development workflow consisted of a digital ocean instance that was setup with a bare git repo. Git hooks were written so that when a push to the repo is made, a clone of the repo is updated and a call to npm install is made to update and install any libraries. On the server the server script is run using nodemon so that when the repo is updated, the server will restart.

This allows very fast staging tests, however this isn't suitable for a production run of the server. The site should be live at https://smple.uk.

Due to security issues some browsers wont connect a secure websocket with an invalid certificate making local development hard. The site is tested working on chrome locally for both https and http. When run under the correct domain everything works as it should.

4 API Usage

As we are using a few public APIs, we needed to take into consideration the terms for each of these.

The songkick API requires attribution when using their data in the form of a graphic they provide. We accommodated for this by adding their graphic into our ticket purchase button.

Spotify have no requirement for the use of their embedded player as it already advertising where the player comes from.

Last.fm require a link to their site on any information that is retrieved from them, we get a biography from them in which they include a link to their website so we are covered here.