Info 253 - Web Architecture: Technical Report

by Stephanie Snipes, Anubhav Gupta, Sindhuja Jeyabal and Carlos Miguel Lasa

---

I. Introduction

Memeoke is a web-based application which simulates the offline karaoke experience by allowing users to sing their favorite songs online while infusing it with a contemporary twist by serving up memes and animated GIFs in the background that are contextually related to the song being sung. Users can also share their experience through the generation of a shortened URL which will lead other users directly to the song page on the Memeoke website, without having to navigate through the menus to get there.

II. Problem Statement

The impetus for this project was the creation of a URL-shortening service that effectively emulated how existing URL shorteners such as Bit.ly currently work. Our team decided to go in a different direction though, as we felt that there could be so much more than URL-shortening in our application. We all agreed that URL-shortening is a great way to share creative content generated by users across the Internet, and one of the manifestations of this creativity has been the preponderance of memes and animated GIFs of trending topics in popular culture.

Following this train of thought, our team believed that creating a mash-up of these memes together with some form of audio, would be a great way to expand the entertainment factor of these animated

GIFs and increase their virality, which would in turn make sense for URL-shortening to step into the picture to make the mash-up easily shareable. The natural by-product of our brainstorming around this area led to the idea of an application that combined memes with popular music, and Memeoke was born.

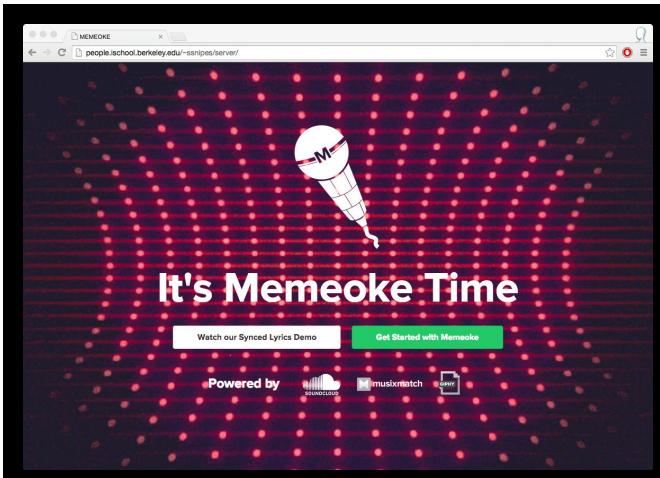
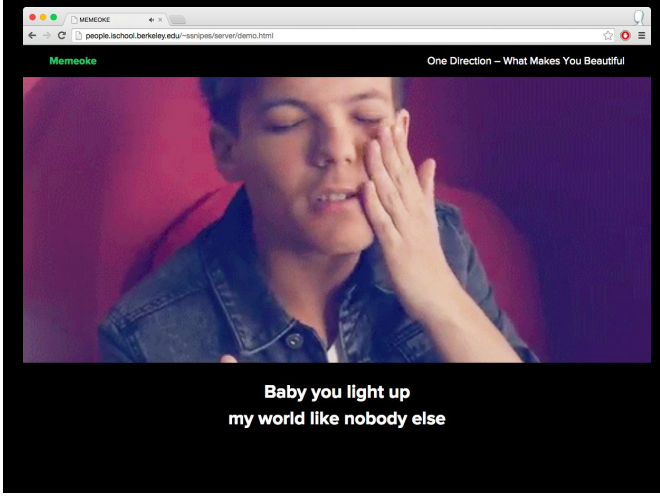
III. Application Features

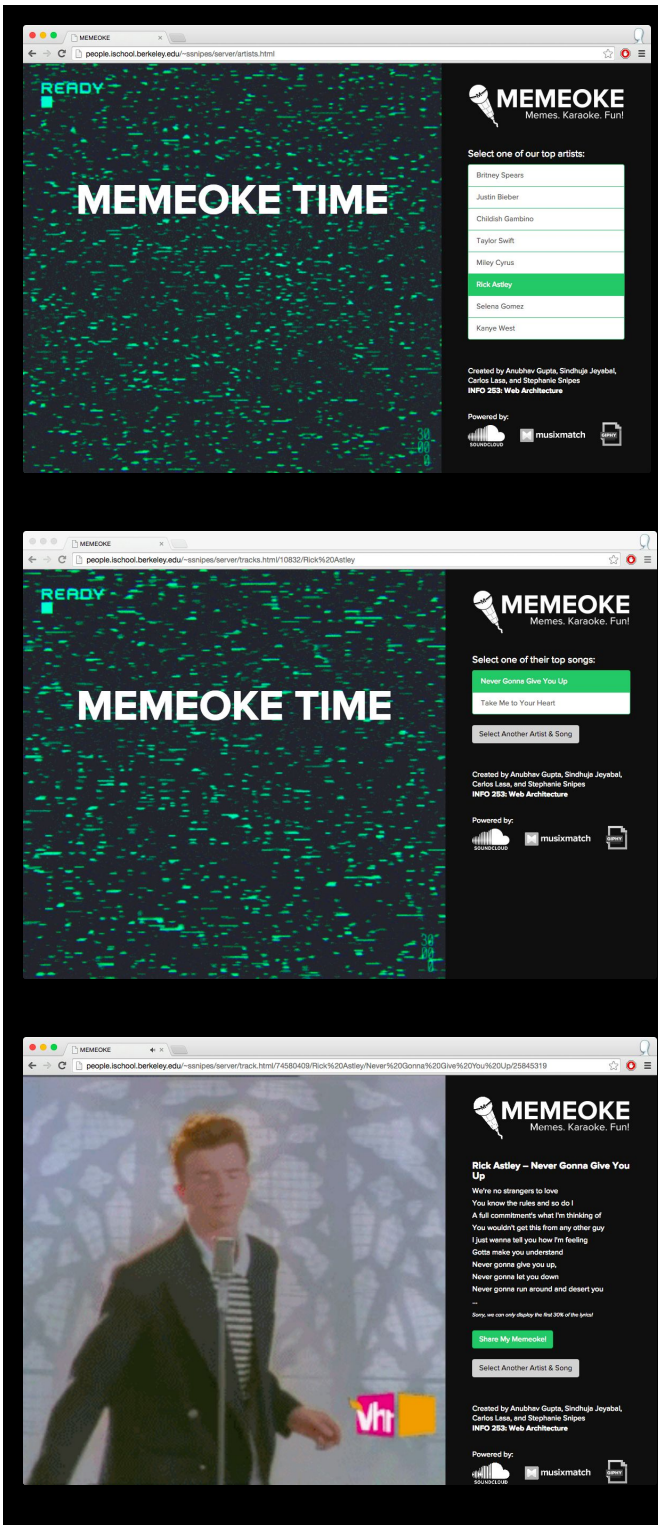
To implement the meme-karaoke mash-up, the following shows the breakdown of features that were initially proposed and what were eventually implemented in the application:

Proposed Feature	Implemented Feature
We plan to have a pre-defined, fixed collection of songs in our database, and we will integrate a streaming service for the song audio.	We offer the user a curated list of popular artists to choose from, after which we generate a list of recent songs for each artist (for which both audio and lyrics exist) that the user can select to sing.
In order to get the lyrics for a particular song we will either store its LRC file (a file format synchronizing a song with its lyrics) in our database or hit the Echonest API which will return the lyrics as a JSON/XML file.	Once a song is selected, lyrics are sourced from the Musixmatch API which returns them in JSON format. Not all lyrics are returned, and there are also no timings for the lyrics due to limitations imposed by the free Musixmatch service. A “demo” feature was created to illustrate how the service would have looked like with timed lyrics.
To make the Memes relevant, we will read the LRC file, generate search keywords or phrases, and pull Meme results from an animated gif database called Giphy through the site’s API.	Memes shown are based on the name of the artist chosen (as the Musixmatch API only returns a fraction of the lyrics) and are sourced from the Giphy API. Only the first 20 memes are taken by the application.
We will start off with changing memes on a timer (for example, every three seconds) and then try to improve the experience by changing them on sound pitch variance.	Memes are displayed 5 seconds at a time. Sound pitch variance was not implemented.

We would also like to expand the application by allowing users to record themselves karaoking to their chosen songs so that they can share their recorded “Memeoke” using a shortened URL. If we do not implement this recording feature, we will allow users to share their session data, including song choice and artist, through a more accessible shortened URL.	The recording feature was not implemented due to limitations of the APIs used. The URL-shortening service can still be used to share the meme-karaoke mash-up though.

Sample User Walkthrough

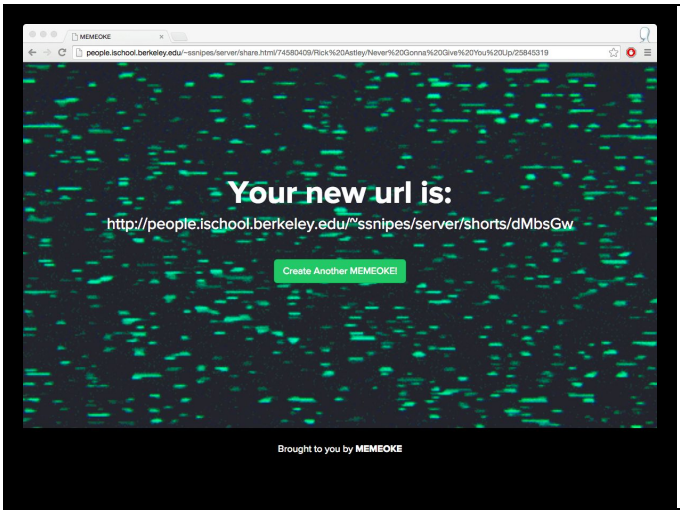
	This shows the landing page of Memeoke. The user has the option to try a demonstration of what Memeoke would ideally look like with synced lyrics or to proceed to its’ current incarnation.
	If the user selects the demonstration, One Direction’s “What Makes You Beautiful” starts playing with synced lyrics displayed on the bottom while One Direction memes and animated GIFs play in the background.



If the user enters the Memeoke website, they are brought to the homepage which shows the list of curated artists on the right (who have a substantial amount of memes on Giphy) which they can choose from. Here, the user is selecting “Rick Astley” from the list.

Selecting “Rick Astley” will load the songs for which he has both lyrics from Musixmatch and audio from Soundcloud. Here, the user is selecting “Never Gonna Give You Up”.

Once a song is selected, the memes and animated GIFs of the artist are loaded in the left container while the right container is updated with the limited free lyrics, as well as links to share the song via shortened URL or to return to the home menu.



If the user selects to share the song, a shortened URL is automatically generated and displayed on the screen for sharing.

IV. Technologies Used

Front-end

The front-end of Memeoke uses standard HTML, CSS and Javascript. The Bootstrap framework was used to create a responsive layout that works on smartphone, tablet and desktop platforms. JQuery and custom Javascript functions were employed to dynamically update the page content, while a Javascript SDK audio player provided by Soundcloud was used to stream audio in the background.

Back-end

The server was implemented using flask and hosted on the ischool server, harbinger. We used the jinja template to access the parameters passed by server to the front end. Clicking on various tabs on the frontend, sent GET/POST requests to the server that were handled by flask using @app.route. In order to store the short urls and corresponding long urls we used an SQLite database. A click on the Share Memeoke button will add a (short_url, long_url) pair to the database, if the long_url was not already present. In order to generate a short URL, we used the algorithm that encoded the current time on the system on the server into ASCII characters and appended it to people.iscool.berkeley.edu/~ssnipes/server to generate a short URL. Use of system time in second ensured a Unique short URL.

APIs

Instead of using static data we used APIs to pull real time dynamic content. We read the documentation of different providers and assessed the APIs that best fit our purposes. We registered

for Developer API keys with the shortlisted service providers. With the help of the registered keys, APIs are invoked across the application to source content:

1. **Musixmatch API** -

   `artist.search` -To get Artist IDs for our curated list of artists.

   `artist.albums.get` - Used Artist ID to get a list of album IDs for the artist.

   `album.tracks.get` - Used Album IDs to get Track IDs for the particular album. The JSON response returned contains the Soundcloud ID for the track which we use to play the song through the Soundcloud widget.

   `track.lyrics.get` - Used to get the lyrics of a track from the track ID obtained above.

2. **Giphy API** -

   `v1/gifs/search` - We used this Giphy API to pull all the relevant gifs by passing the Artist name as a search parameter.

3. **Sound Cloud -**

   We used Sound Cloud's Javascript SDK to integrate SoundCloud API into our Website to play the song.

   `SC.initialize(Client ID)` - Initialize our client using the Client ID we had generated.

   `SC.stream("/track/"+soundcloud_id)` - The streaming of the SDK is powered by the soundManager2 library which is loaded by the SDK automatically

Challenges & Trade-offs

There were multiple challenges encountered by the team in developing the application. The first one we encountered was trying to identify the appropriate services to use to generate the content on the website.

For audio streaming, we initially wanted to use Spotify to play songs, but Spotify required users to create or login to their accounts, an extra step that further complicates the user experience of the website. We then switched to Soundcloud, which provided a simpler user experience with their Javascript SDK but the trade-off there was that there was a high probability that the audio would be a cover of the song rather than the original version due to licensing issues.

For sourcing of lyrics, we originally intended to use Echonest as it had a special partnership with Lyricfind. We then found out that a special approval would be needed from the Lyricfind sales team to secure our API key, so we switched to Musixmatch instead. Musixmatch provided us with a robust artist, album, track and lyrics database, but the trade-off was that the free version of the service only allowed us to access a portion of the song lyrics (not 100%) and the song lyrics were no longer timed or synced to the audio. Similar paywalls were encountered for other lyric services across the Internet, so we stuck with the static lyrics of Musixmatch.

Once we had narrowed down APIs, we were presented with another challenge - ensuring that content from all services (Giphy, Musixmatch and Soundcloud) existed for the songs we wanted to include in the scope of our application. Thus, we had to limit the artists and songs in our database to a select few which had enough content to provide a good user experience.

Some of these selected artists also had mixed results. While some artists provided full-length songs with full SoundCloud track ID numbers, the artists also had restrictions on other songs which would not provide any audio, likely due to licensing. The songs without audio returned a SoundCloud track ID of “0.” So as not to confuse the user by displaying songs that could not actually be played, we used JavaScript and CSS to add a style of “display: none;” to all songs with a “0” SoundCloud track ID, thus hiding them from the menu of available choices.

Another issue we had was making AJAX calls. We were running into cross origin (CORS) request issues. We created a CORS request object and were still not successful in running the API. So we resorted to the flask framework and made api calls from the flask server.

In terms of front-end development, making the application responsive was a challenge in that its desktop size utilized a fixed sidebar on the right-hand side. We needed to make sure that the sidebar could adjust itself to move below the Giphy images so that the lyrics displaying in the sidebar would be readable. It was challenging to figure out how to adjust the “position” attributes in CSS and to use media queries to ensure that the layout changed successfully at certain device widths.

V. Future Possibilities

If we had unlimited resources (i.e. budget) to implement all the features we wanted, there are several key features we would like to implement. First would be to implement synced lyrics to fully emulate the karaoke experience. This would require a paid subscription to services such as Echonest or Lyricfind. Having this would enable another feature we would like to see, which is displaying contextually-aware memes or GIFs based on the lyrics that are currently being sung by the user. If we also had the budget, we would prefer to stream the original licensed audio of the song. With more robust APIs such as these, we would be able to expand the song selection, and perhaps be able to implement a search functionality to search for popular songs or artists. We would also like to expand the application by allowing users to record themselves singing to their chosen songs. These recordings would then be saved on our server and can be shared by the user with a shortened URL. With enough usage metrics on the application, we can then start to implement additional features such as identifying popular songs based on number of times sung, number of times shared and number of times accessed via the short URL.