“Gallery R”

Interactive Video Content Portal
Web Application

Sarah Federman
IGME 330
Final Project
11/30/14

Sarah Federman
Final Project Documentation

Github link:
https://github.com/sarahbethfederman/Gallery-R

Project link:
file:///Users/Sarah/Documents/Portfolio/3.0/Portfolio/Portfolio/vid-reel/index.html

Project Definition
The Gallery R exhibit is a visual showing of the history of the New Media Design major, through the use of projection screens leading through the exhibit and interactive installations to keep viewers engaged. My portion of the project involves creating an interactive installation to be displayed on projector screens and iMac displays.

Goals
Create an web-based interactive installation that:
- Keeps viewers/gallery visitors engaged
- Functions with or without user interaction (loops without prompts)
- Works on all screen sizes
- Incorporates the exhibit titling
- Gives viewers a way to navigate through the directory of interviews

High-Level Architecture
Javascript architecture:
- *main.js* - controller object, hooks up DOM, starts intro sequence, runs init function on DOMContentLoaded
- *introLoader.js* - intro controller (singleton), starts the intro titling sequence and when finishes starts the main loop
- *dataLoader.js* - loads the metadata and urls for the videos from *Firebase*
- *mainLoop.js* - controls the main video loop, inits when the data from Firebase finishes loading
- *slide.js* - creates a prototype for a slide module that wraps the video and metadata with some helpful functions and variables
- *slideNav.js* - creates and controls the navigation overlay boxes
- *preloader.js* - module for loading the next video ahead of time to reduce buffering
- *video.js* - module contains common functions to be used on HTML5 video events
- *buttons.js* - module contains helper functions for hooking up buttons

CSS architecture:
- _base.scss - base document styles
- _normalize.scss - browser style inconsistencies normalizer
- _buttons.scss - handles button styling
- _headers.scss - styling for the metadata display
- _animations.scss - holds keyframes and animation classes to be used with javascript
- _slides.scss - handles the slide nav styling
- _states.scss - handles state classes like show/hide/selected, etc. to be used with javascript
- _layout.scss - base layout styling
- _video.scss - style for video and progress bars
- style.scss imports all the partials and compiles to style.css

Design Decisions
One challenge I ran into was that some people don't have interviews associated with them because they've passed or were unavailable to interview.

I decided to show a filler video for some time so users can see their bio and picture. At first I tried to just show a poster image for a small duration, but it turns out this was not possible for a multitude of reasons, some of which include: breaking the video to show the poster image involves forcibly stripping the src tag, and because a picture isn't a video and the duration property of HTML5 videos is a read-only property, I couldn't set it to show for a certain amount of time. Then I had to make sure that the content overlay doesn't fade out when it's a filler video.

Technical Decisions
A technical challenge I ran into was loading the firebase. Firebase runs via web sockets and keeps an open connection to the Firebase server, and the way to access the data from the API is an asynchronous event callback. So when the event fires that says the data is loaded, it runs a callback function. The problem was that the video loop was being initialized synchronously before the data it needed was loaded. The solution involved passing around a callback function to the data loader that inits the loop once the data has finished loading.

Another technical issue was since the videos are large files they cause long loading and buffering. I tried to start the video only on the 'canplaythrough' event, which would not play the video until its loaded through, but Chrome apparently has a documented issue and is not firing this event correctly as in the HTML5 spec, and its firing at the same time as the 'canplay' event (which fires when the first few video frames have finished loading).

So instead, I thought I would show a loading animation before this event is called, and whenever the video starts to buffer. The animation is an svg that is added and removed from the DOM on relevant events. But apparently chrome isn't following spec on this either and doesn't fire any event when buffering, and firefox and safari fire "waiting" and "stalling" events with different behaviors.

So there was no win-win way I could think of to write preloader. Joy! Time to go back and compress the video files even more.

My solution involved writing a preloader that is a sort of "ghost element" and loads the next video in the queue before it has to play ,by creating an DOM node in the background and loading the video. This works all right, but it doesn't solve the whole issue, and can't help in situations where the user actually navigates to a specific (not yet loaded) video.

I chose to use flexbox for the slide navigation bar because it makes horizontal scroll easy, and the best part, in order to make the "view-all" function work, all I had to do was apply a class with 'flex-wrap' set to wrap and move it to the top of the page, and suddenly it's not horizontal, it's a full screen grid. I love flexbox.

HTML5 Technology
I did a lot of work with the HTML5 video browser API and its associated quirks. Hooking up a loader animation with the video's event turned out to be a larger-than-anticipated amount of work & learning.

Firebase's service is like an externally hosted JSON database (kind of like Mongo) and comes with its own associated API.

Credits

- *Browserify* is used for concatenation and module dependency management. This is a node js tool used on the command line. Last project I used requireJS for the same purpose, and this time I wanted to learn something different. The main difference is that Browserify concatenates files to one large file on the server (or in my case, locally) instead of loading script files on page load like requireJS. Browserify also uses nodeJS syntax (similar to 'require'-ing npm modules and defining exports for new modules), which I thought might help me learn node in part II of this course. http://browserify.org/

- For my workflow, I used the node module Watchify to bundle my browserify modules on every save. https://github.com/substack/watchify

- Normalize.css (in my partial as scss) by Nicholas Gallagher: http://necolas.github.com/normalize.css/

- Loader SVG is adapted from this pen: http://codepen.io/aurer/pen/jEGbA