NORTHERN MICHIGAN UNIVERSITY

# SENIOR PROJECT REVIEW

B.S. Mobile and Web App Development

**Author: Alizée Wickenheiser (nwickenh@nmu.edu)**
**Adviser: Dr. Michael R. Kowalczyk (mkowalcz@nmu.edu)**
**11/28/2016**

End of term review for software project to develop an implementation of realtime subtitles for music with extra features.

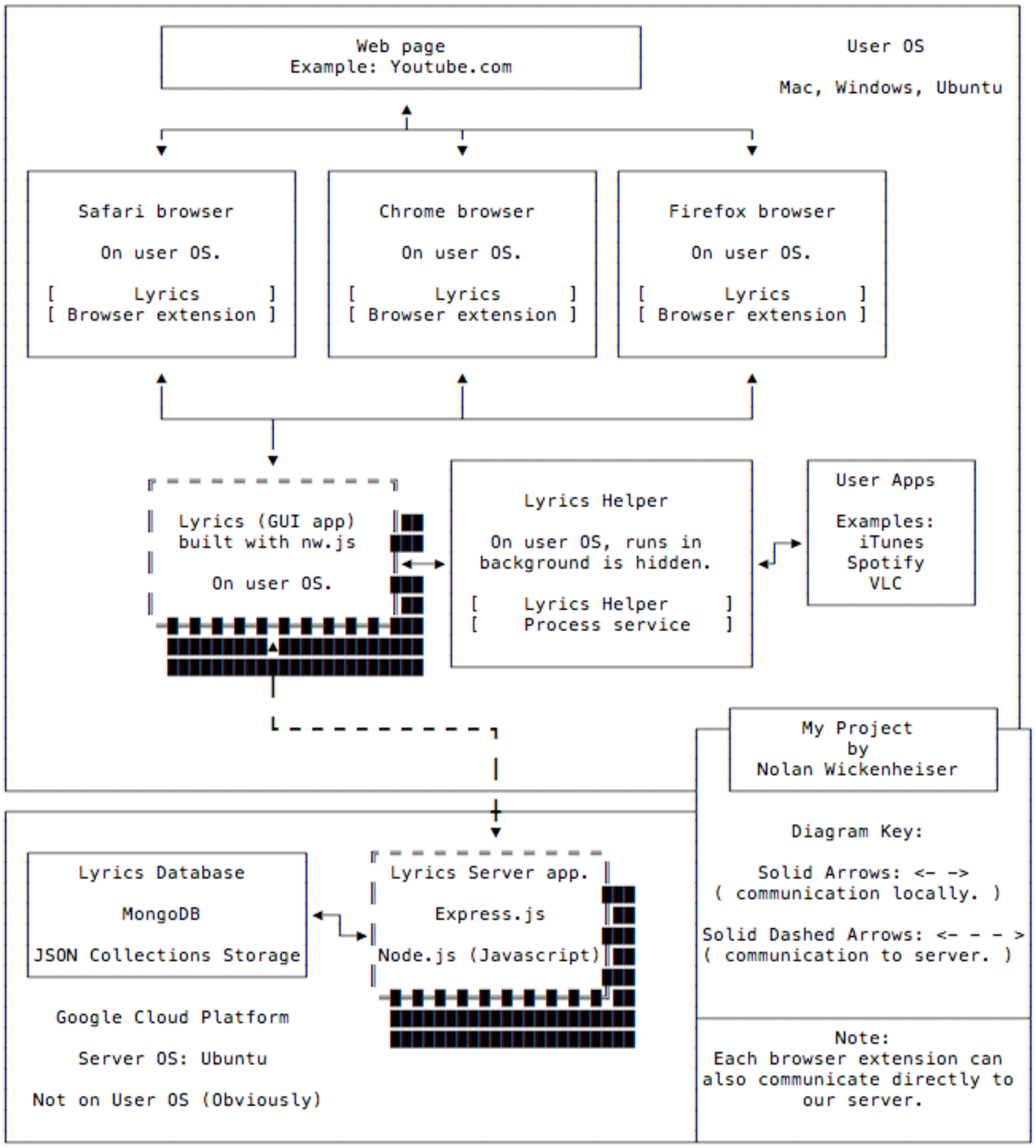
I approached my senior project planning in advanced and taking the time to research what technologies are cool. My definition of cool for technologies staggered around reusability. Will I be able to reuse the structure of how I'm going to approach future problems with what i'm building? Hopefully, yes because going by a process that can be reused with adaptions ultimately continues forward progression and I have a lot of awesome project ideas that will require functionality shared in this project.

In advanced, before building anything with my time, I first think about the reason of building and what the expected outcome will resemble. I listen to music a lot because I'm a software engineer/person, being able to listen to music while coding is one of the great gifts bestowed upon 'most' programmers. Typically I enjoy classical music set to low volume but my taste changes depending on the difficulty level of coding which always fluctuates. In my life I've become attached to listening to foreign music. Reasons unknown to me I adore good melodies and vocals that share an interesting passage. Listening to music in other languages than my own can be a challenge but the process helps with learning foreign words and it's enjoyable as a bonus. So how can I improve this listening experience? I believe by adding eyes to the communication pipeline of data to the brain.

Adding subtitles to media content isn't a new thing but it hasn't in my mind been executed properly. If a listener of music or a viewer of media wants to have a subtitle experience of content, today there is no real good way to get it done. Translating content is costly, difficult and hard to measure in quality-wise with the end result always being reliant on trust or there would be no need for a translation. A solution I should aim for solves handling cost of translation and decrease the difficulty of getting something translated with making the experience somehow better which results in fun features to implement as a developer.

Displaying the media subtitles along with the translation of the user's native language is the ultimate goal. Yet, the system of translating should stay separate from the actual content as best it can. I believe that translations should be able to hook onto media files and translations should not be stored in the media files. Reasoning is that baggage is bad, users have storage limits and the cloud may solve storage but users need perfect 24/7 online connection before the cloud truly solves the problem in an elegant way. Also navigating humans to do something like translating content isn't easy, I imagine the ultimate solution being all automated and that is futuristically possible. My solution is not fully automated, relies on a 'bounty system' I decided on building, which I believe gives motivation for humans to navigate reasonably on a path where music is translated and is more feasible to develop with time constraints. I also like the bounty system because it can be used for not just music and I kept it simple so it's very pluggable code.

The toolbox used to accomplish my tasks ended up being mostly Javascript, some Objective-c, Swift, NodeJs, MongoDB, Nw.js and open source libraries which can be found in the codebase. I also have C++ code written for music info retrieval on Windows but I decided to focus my development for macOS. Yet, the codebase will run on Windows or even a popular Linux distribution such as Ubuntu with some small build deployment adjustments. There is a likely chance that I'll eventually finish development for all operating systems. Nw.js is a framework sponsored by Intel, I ended up using for the GUI development which supports all major operating systems. Nw.js takes the open source code from Chromium built by Google and strips a lot away so the end result is simple GUI development provided by html, css and javascript. Yes, the app I ship is pretty much a chrome browser in disguise but has processes written in more native languages that run on the user's OS in the background. The processes are written in robust languages like Swift, Objective-c, C++ and can run code for OS specific API tasks or send commands to the GUI app. The process launched on the OS from the GUI app has end to end communication with the GUI app; done by a local socket on an open port. Data in packets being sent typically contain JSON that can contain music information currently on the user's computer or a 'custom user task' was done message. The receivers on both ends of the communication will execute events or modify properties. There is even a web-socket from the GUI app to my server and same goes for web browser extensions I built for Chrome, Firefox and Safari. The browser extensions can talk to the GUI app as well. So I'm able to get music information from youtube played in the browser and send it to my GUI app which then figures out what is being played. Same goes for local processes finding out information from local media player apps, such as iTunes and sends it to the GUI app. I've illustrated the communication layer architecture between the main components of my code below in a diagram representation.



As the illustration shows it’s actually very simple to have the code for access to information and all development advantages that comes with it separate by components. I’m able to have my development environment for each component as it’s own project to keep the code from getting cluttered. So I have multiple IDE windows open for each project on my development machine which is a MacBook Pro Retina, I recommend for the maximum ‘screen real-estate’ resolution. The IDE I mainly used for development in this project is WebStorm by JetBrains and I highly recommend it with their whole suite ‘flavors’ of IDEs. Such as CLion for C++, AppCode for Swift/Objective-c, and WebStorm for Javascript/Node.js are all the IDEs I used in making this project. The IDEs generally all look, function and behave the same but configurations are different depending on the development environment. My provider for hosting my online services that communicate with the GUI app ended up being Google. Generally their Cloud Platform was simple for configuring a Ubuntu distribution and once SSH was setup between my development machine; the deployment process between development to production execution is a few commands.

Once I had a straightforward idea of the design complexity needed for data communication between components. I ran into issues with retrieval of music data and how best to approach the task in general. iTunes music information can be retrieved using Scripting Bridge which is provided by Apple; you provide an output path for the Objective-c header file and path to iTunes with a slick command to your Terminal. Example: `sdef /Applications/iTunes.app | sdp -fh --basename iTunes`

Scripting Bridge can be used to access basic tasks of typical OS X applications but it tends to be luck based if you actually get useful event call methods for the application. When I first started using Scripting Bridge there was no Swift programming language and I eventually converted over to using Swift instead of Objective-c because of a memory leak issue with Scripting Bridge that still exists when using Swift but isn't as bad. The problem is that the process running in the background using a Scripting Bridge header file to keep checking music information from iTunes will continue to grow in memory by a memory leak. After spending days stripping out code to pin point the issue by using Xcode's "Instruments" for memory analysis by running my program for an hour in between tests. I determined there was no possible solution to fix but I could monitor the memory of the process and relaunch the process if it ever gets to large which isn't perfect but helps me sleep at night. It takes a couple days of running non stop where a restart may be needed and the user likely will not notice anything even if using my app. Another issue was that Scripting Bridge was not returning any details if the user was using Apple Music a new music streaming service introduced recently similar to Spotify. A solution was to write some code that hooks onto Apple's OS notification events for iTunes which gives me JSON data as seen below but doesn't give me current time in song which Scripting Bridge method does provide.

```
{
   "Name": "A coeur fendre”,
    "Total Time": “187360",
    "Artist": “Alizée",
    "Album": "Une enfant du siecle”,
    "Player State": “Paused"
}
```

The current time in a song is crucial for my app and without it I will not be able to know when to show or add subtitles. While integrating the notification code with Scripting Bridge code, I realized that I shouldn't have to constantly poll for every bit of iTunes information every second, if I create a timer that mirrors the song time by only needing to check the 'play state' event of the song. I ended up saving cpu cycles with solving how to get all the necessary data by just polling the song 'play state' and modify a timer which becomes the song time.

In conclusion I'm happy with all the knowledge I've gathered with building this project. I encountered interesting issues and dug deep into the javascript sandbox with producing a pretty sandcastle in the end. I'm proud of the work, in reality it was a big project for one person and had many different aspects that could have been approached differently but I'm happy with what I chose in the end. I plan to continue building features for this project, it's one of those projects that may never be done where code stays untouched and I'm happy with it being like that.