SP 17 Blue Chrome Plug-in Final Report

CS 4850-01

Fall 2024

Professor Sharon Perry

12/2/2024

Asa Longino, Eli Headly, Jonathan Hardy

Man Hours: 164

Lines of Code: 807

Number of Components/Tools: 9

Website: https://seniorproject17.github.io/cs4850-website/

Github: https://github.com/SeniorProject17/cs4850-senior-project

Table of Contents

1. Introduction

The purpose of this Spotify Chrome plug-in is to enhance the user experience for Spotify listeners. Our goal is to allow users to skip, like, and shuffle songs directly from their Chrome browser window without needing to open the Spotify app. Upon starting the extension, users will be presented with a pop-up window that prompts them to sign in with their Spotify username and password. After logging in, users will see the current song being played within a pop-up window. They will have the ability to play, pause, and navigate between next and previous songs in their queue. Outside of the pop-up window, this Spotify plug-in will allow users to highlight text in their browser, right-click on the highlighted text, and search for it in Spotify. Additionally, this plug-in will add a button under YouTube videos that allows users to easily find their favorite songs from YouTube on Spotify.

1.1. Project Goals

There are three main goals of the Spotify Plug-in that the team chose:

1. Provide some of Spotify's existing features in a smaller and condensed pop-up that doesn't limit the space taken on a webpage. Existing features include displaying the current song being played, the play button, next/previous song buttons, and the shuffle.
2. Provide a feature that allows users to open a song from YouTube on Spotify (if it exists). This will allow users to set up their playlists based on songs that they heard from one platform across to Spotify.
3. Provide a feature for allowing a user to search for a song in Spotify based on a text or set of song lyrics.

1.2. Intended Audience

The intended audience for the Spotify Chrome Plug-in is any user who actively uses Spotify to listen to music on their computer using Google Chrome.

1.3. Definitions and Acronyms

Use case – describes a goal-oriented interaction between the system and an actor. A use-case may define several variants called scenarios that result in different paths through the use case, and usually with different outcomes.

Scenario – one path through a use case

Shall – an adverb used to indicate importance; indicates that the requirement is mandatory.

"Must" and "will" are synonyms for "shall".

Should – adverb used to indicate importance; indicates the requirement is desired but not mandatory.

May – an adverb used to indicate an option. For example, "The system may be taken offline for up to one hour every evening for maintenance." Not used to express a requirement, but rather to specifically allow an option.

Control(s) – the individual elements of a user interface such as buttons and checkboxes.

Feature(s) – an intended activity that is part of the plug-in that a user is supposed to experience or use.

Track(s) – another word for the song that is in a playlist or album.

2. Design Constraints

The design constraints listed below are some of the restrictions that we either chose to follow in our implementation of the Spotify Chrome Plug-in or were forced to follow based on the architecture that Google Chrome uses for users to follow.

2.1. Environment

The Spotify Plug-in will be using the Spotify API to utilize some of the features such as playing a song, going to the next/previous song, displaying the current song, and shuffling the songs played. In addition, we will also utilize features such as searching for Spotify content to locate songs in Spotify from text or song lyrics as well as the ability to find a song from YouTube in Spotify.

2.2. User Characteristics

Most of the users who will be utilizing the Spotify Plug-in will have access to a Spotify account and be using their account through the plug-in. Since our plug-in will only be targeting users with Spotify accounts and who utilize Google Chrome, they will be our priority when developing features and making these features easy to use.

2.3. System

The Spotify Plug-in is intended to be on the Google Chrome internet browser as a Chrome extension. The targeted systems that should be able to run this plug-in would be Windows and Linux Workstations utilizing Google Chrome and the Chrome Store to add the plug-in.

3. Requirements

This section contains the defined requirements for the Spotify Plug-in that was created for users who have the Spotify application. The purpose of this section is to show users of the

Spotify Plug-in the requirements in a way that allows for easy readability and to show developers the design and implementation of the plug-in.

3.1. Functional Requirements

The following list outlines the functional requirements that must be implemented in the Spotify Plug-in. These requirements specify the actions that users should be able to perform through the Spotify Plug-in's user interface, organized into categories to clearly indicate where each requirement fits within the overall functionality of the Plug-in.

1. User Login
    a. Users must be able to login using their Spotify account using their email and password.
2. Music Playback
    a. Users must be able to play and pause songs, as well as skipping to the next or previous song.
    b. Users must be able to like songs and or shuffle through the currently playing songs or playlist.
    c. Users must be able to adjust volume on the currently playing track.
    d. The Plug-in must be able to support background playback while users navigate away from the Plug-in pop-up.
3. Additional Features
    a. The user should be able to search for a song on YouTube on Spotify through the use of the Plug-in.
    b. The user should be able to search for a song on Spotify using just the lyrics.

3.2. Non-Functional Requirements

The following list outlines the non-functional requirements that must be implemented in the Spotify Plug-in. These requirements specify the quality attributes and constraints, such as performance, design, and usability, that the Plug-in must adhere to, organized into categories to clearly indicate where each requirement applies within the overall functionality of the Plug-in.

1. Performance
    a. The Spotify Plug-in must be able to perform the calls to the Spotify API quickly to not cause more than a 3 second delay at most. In addition, any features beyond the Plug-in’s external interface must be able to perform quickly to not cause more than a 3 second delay for any features it utilizes.
2. Design
    a. The design of the Spotify Plug-in must be considered aesthetically pleasing to the user and the text elements on the Spotify Plug-in must be readable in addition to being aesthetically pleasing to look at.
3. Usability

a. The Spotify Plug-in must be able to access either the Spotify Desktop Application or the Spotify Browser application. The Plug-in must use one of these to make the calls for the extension to work.

3.3. External Interface Requirements

This Spotify plug-in must be easy to use and resemble the interface of Spotify's current application. The layout and themes used in the user interface should match those of the Spotify app. Additionally, the buttons provided in our extension should perform the same actions as the buttons in the Spotify app to avoid confusing users. Regarding the software interface, this Spotify Plug-in should utilize the API provided by Spotify to retrieve and update user data such as liked songs, Spotify playlists, and listening hours.

4. Analysis

As college students, we use Spotify daily, but managing Spotify and Chrome simultaneously can be cumbersome, especially when switching tabs just to change the current song. This project focuses on designing a Spotify Chrome plugin that students like us would genuinely find useful. Our primary goals were to enhance user experience by integrating essential Spotify features into a compact, space-efficient pop-up that minimizes webpage interference. The plug-in incorporates key Spotify functions, including displaying the current song, play/pause controls, next and previous song navigation, and shuffle options, all within a streamlined interface. Additionally, it enables users to open songs found on YouTube directly in Spotify, facilitating seamless playlist building across platforms. Finally, the plug-in provides the ability to search for a song on Spotify using text input or specific lyrics, making it easier to discover and enjoy music on Spotify.

5. Design

The purpose of this Spotify Chrome plug-in is to enhance the user experience for Spotify listeners. Our goal is to allow users to skip, like, and shuffle songs directly from their Chrome browser window without needing to open the Spotify app. Upon starting the extension, users will be presented with a pop-up window that prompts them to sign in with their Spotify username and password. After logging in, users will see the current song being played within the same pop-up window. They will have the option to search for songs at the top of the pop-up or navigate through songs in their queue. Outside of the pop-up window, this Spotify plug-in will allow users to highlight text in their browser, right-click on the highlighted text, and search for it in Spotify. Additionally, this plug-in will add a button under YouTube videos that allows users to easily find their favorite songs from YouTube on Spotify.

5.1 Design Considerations

We assume that users have the Chrome browser installed on their desktop or laptop computers with an up-to-date version of Chrome. Users can access Chrome on any operating system, whether it be Linux, Windows, or macOS. Our end users are not required to have any technical proficiency beyond the ability to open a browser and

click on the provided buttons. Regarding functionality, we aim to maintain the core features of our Spotify Chrome extension while introducing new features in the future based on customer feedback.

Firstly, our hardware constraints are defined by the hardware requirements for Google Chrome. The hardware specifications for each operating system and processor type can be found at this link: https://support.google.com/chrome/a/answer/7100626?hl=en. Additionally, the performance of our software is limited by the processing power available on the host device.Our software constraints are also influenced by those of Google Chrome. Chrome's Manifest V3 restricts the elements that our extension can access. Our extensions must obtain permissions from both the device and the webpage to edit elements and download content to the user's device. The user must enable our extension in their Chrome extension settings for the plug-in to function. Moreover, our Spotify Plug-in requires network connectivity to retrieve data from Spotify and edit YouTube pages.

For this project, our team will employ Scrum, an agile software methodology, to iteratively develop our software. We will achieve this by planning one-week sprints, during which a set of tasks will be designated to signify the completion of a major step or feature in our project. At the end of each sprint, we will hold a sprint retrospective and a sprint planning session, led by our team lead, who will act as the Scrum Master. Every few weeks, we will provide progress updates to the product owner, Sharron Perry.

5.2 Architectural Strategies

For this Spotify plug-in, we will implement our product in JavaScript, using HTML and CSS to create a user interface with a minimalist design paradigm. This means our pop-up and any UI functionality added with this extension will be designed to minimize the number of clicks required for user interaction, keeping it simple and intuitive. Additionally, we will design our extension to ensure proper security. This will be achieved by having users log in to Spotify via a proxy server. After logging in, users will receive an authentication token that allows them to securely access their Spotify information. In the future, we hope to extend the functionality of our plug-in to provide Spotify song recommendations based on cookies collected while surfing the web. This would involve using a machine learning model to predict song recommendations based on users' Chrome activity.

5.3 System Architecture

Our system is composed of three major parts: a user interface, a backend, and a Spotify API interface. The user interface will be written in HTML to define the buttons and elements in our pop-up, with CSS used to style each component. Our backend layer will provide the functionality for each button and feature offered by our Spotify plug-in, as well as handle the transfer of information from the Spotify interface to the UI. Finally, the Spotify interface will be responsible for making Spotify API calls and formatting the responses in a usable format for our application.

5.4 Detailed System Design

1. Classification
    a. The Spotify plug-in is classified as an application program. The plug-in involves the use of Google Chrome's extensions and the Spotify API to give the required services that the users may need.
2. Definition
    a. The purpose of the Spotify plug-in is to streamline the process of finding and playing a song based on specific criteria, reducing the time and effort needed to search for music on Spotify. It simplifies the experience for users who want to locate a song they discovered while viewing lyrics or who wish to find a YouTube song on Spotify. This service also makes it much easier to add songs to playlists or create new ones.
3. Constraints
    a. The constraints on the Spotify plug-in vary on a hardware level to a software level as well. From the hardware level, the hardware requirements of Google Chrome may limit the Spotify plug-in's performance by the amount of processing power that is available on the host device. This can result in reduced speeds and response time for our plug-in and can result in delays for the end users.
    b. From the software level, Google Chrome controls certain elements that our plug-in can utilize. The plug-in must require permission from the device and the webpage to edit elements and content on the user's device. This will also require the user to allow the plug-in in their Google Chrome extension settings to function as intended. In addition, the plug-in will also require good network connectivity to ensure it can retrieve data from Spotify and YouTube for some of its features.
4. Resources
    a. There are a wide array of resources that are available to implement and develop the Spotify plug-in. The Spotify API will give the necessary calls to Spotify and its features and will allow the plug-in to connect with the user's accounts on Spotify. Chrome for Developers will assist in utilizing Google Chrome's extensions and learning about any of the tools that Google Chrome's extensions can offer.
5. Interface/Exports
    a. Spotify Search API Endpoint: Allows users to search for tracks, artists, albums, or playlists. The response is returned as a JSON object containing a list of matching items.
    b. Spotify Playback API Endpoint: this enables users to control playback directly from the plug-in. These actions are executed via button clicks in the plug-in's UI.
    c. Spotify Playlist API Endpoint: this enables users to view and manage their playlists, and will be used to add songs from Youtube.

6. Development

This section details the development process followed for each component of our Spotify Plugin

6.1. Frontend Development

In our implementation each component of the popup layout is containerized within a tag and styled using a flex display, allowing elements to dynamically reorient based on varying song titles and album cover sizes. In the control panel, button icons were generated with the Font Awesome CSS library, while the song title and artist information were implemented with paragraph tags and styled in CSS for readability. For the display panel, the background showcases the album cover of the currently playing song, with a linear gradient applied in a separate container inside the display panel to create a smooth transition between the control panel and the display panel.

After designing the popup's initial structure, we used JavaScript objects to store and manage the state information for each song, including the title, artist, album cover, and associated HTML/CSS tags for the popup state. This state data is generated by our backend and updates dynamically whenever the next or back buttons are clicked or the popup is opened.

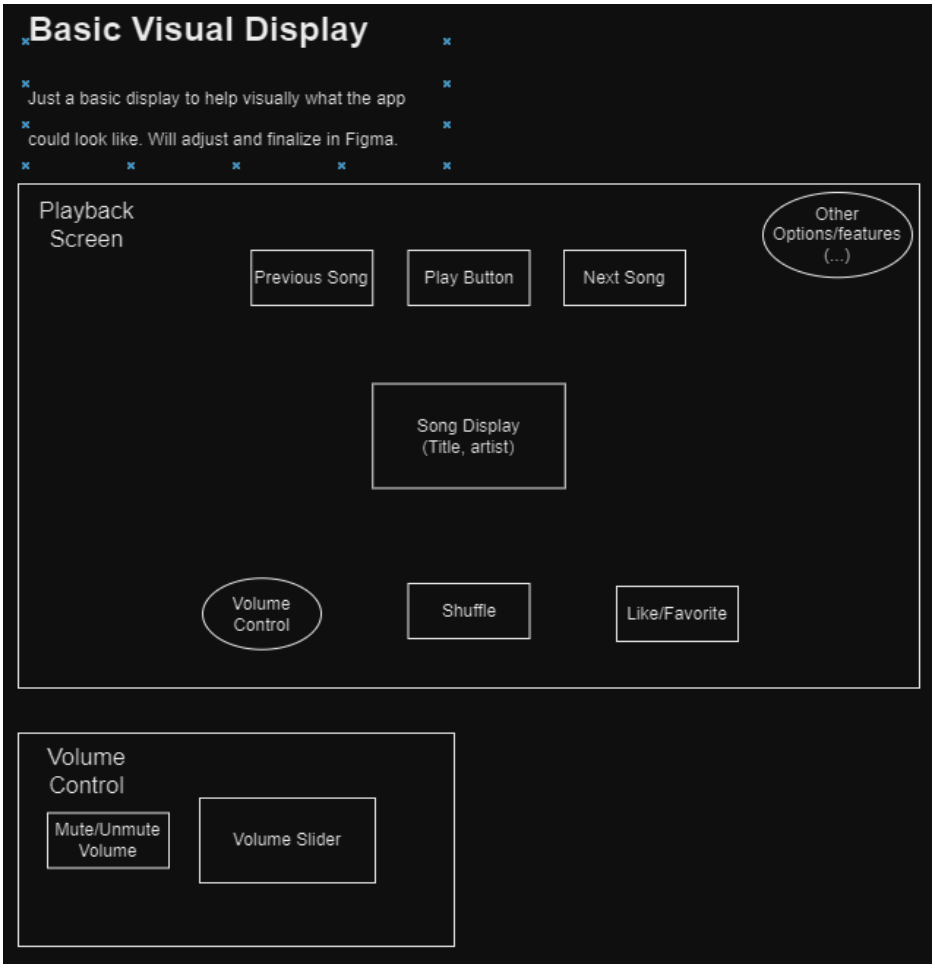


Fig 1. Picture of the first skeleton drawing of the layout of certain buttons and features on the Spotify Chrome Extension

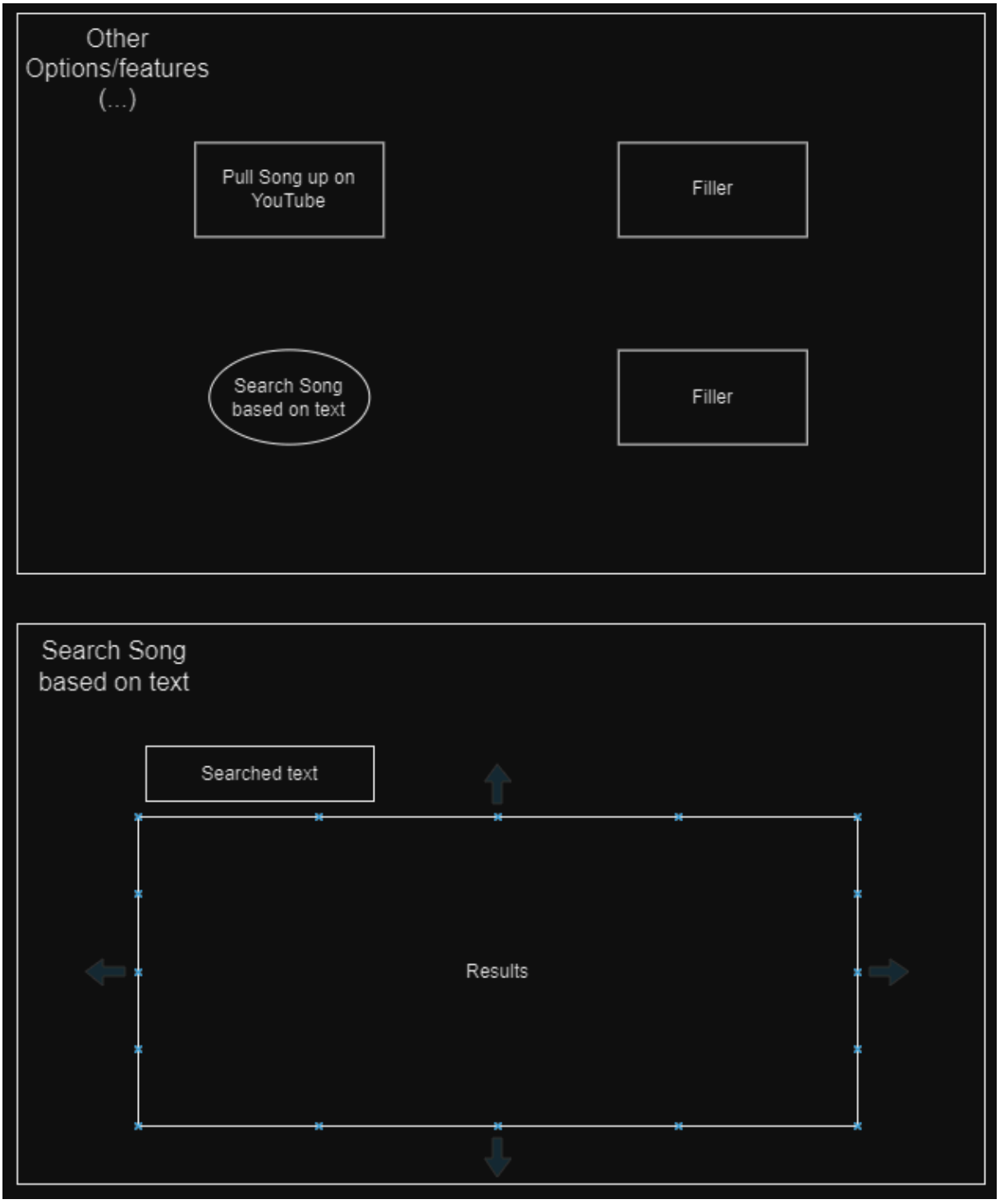


Fig 1 (Continued). Picture of the first skeleton drawing of the layout of certain buttons and features on the Spotify Chrome Extension

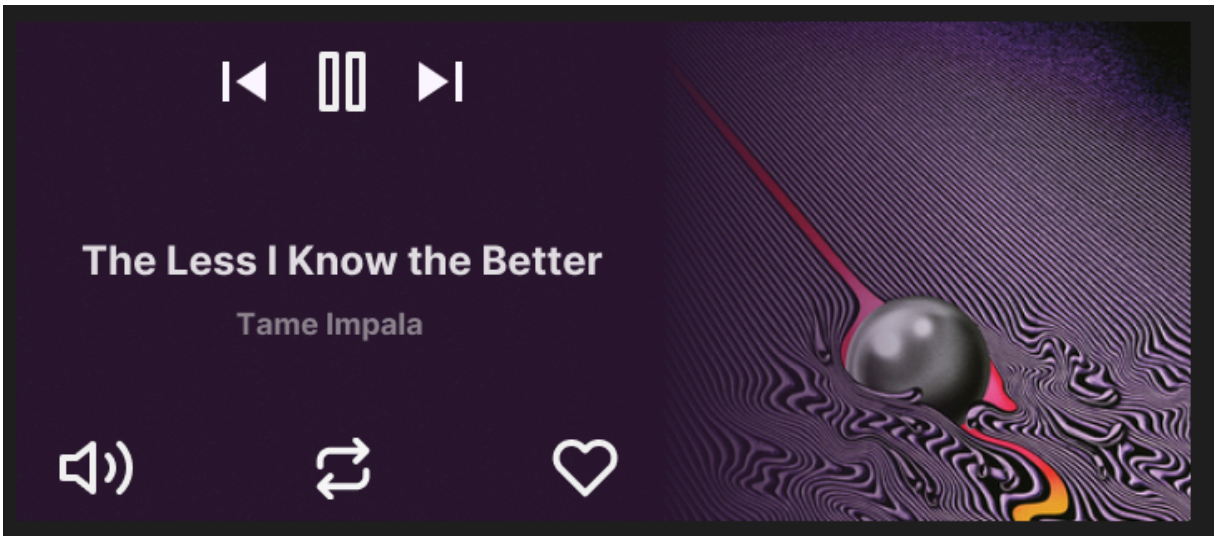


Fig 1. Picture of the first Figma drawing we had for the UI of the Spotify Chrome Extension

6.2. Backend Development

On program initialization, the backend establishes a connection with Spotify using a client ID and specified scopes, storing the access token in the user's local storage as a cookie to maintain authenticated access. During the “onInstalled” event, we add a listener to authenticate with Spotify and set up the context menus, including options for searching lyrics and methods for context menu actions.

To dynamically generate a cohesive color scheme, we use image color analysis on the album cover of the currently playing song. By looping through blocks of pixels, we initialize an array of pixel colors, which are then processed through a basic k-means clustering algorithm to group similar colors. The two largest clusters are selected to form the color palette. A luminance check is then applied to determine the brighter color, which is assigned to text and button elements, while the darker color serves as the background, ensuring optimal contrast and readability in the popup interface.

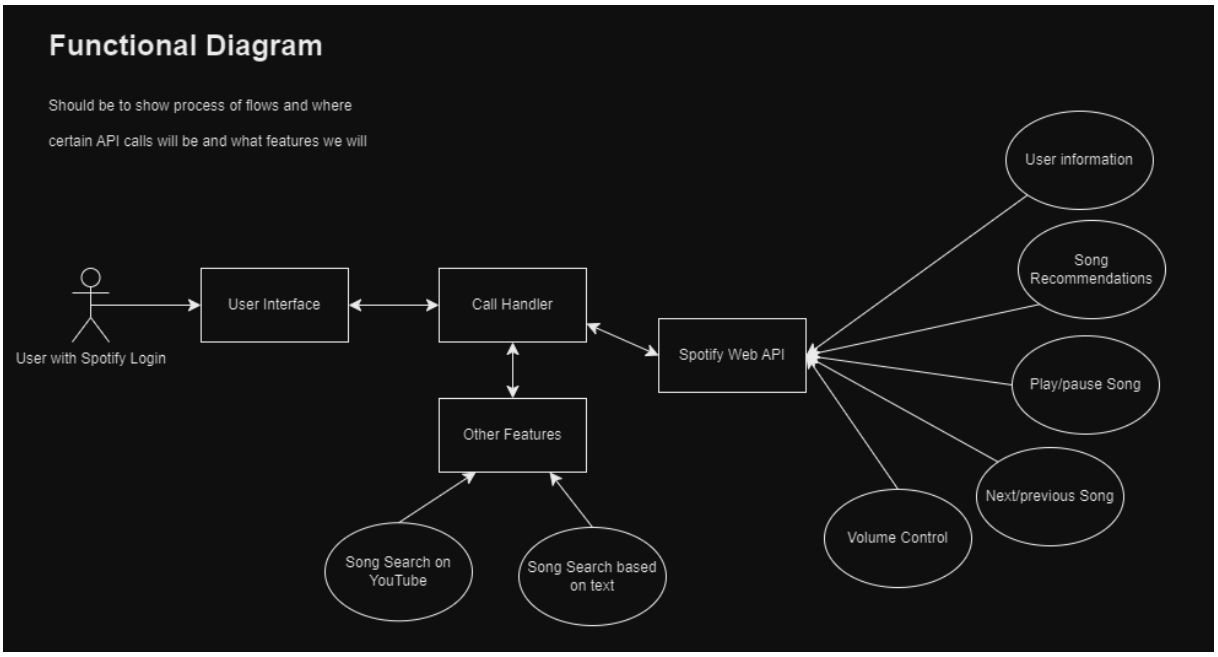


Fig 2 . Picture of the functional diagram of the interactions that would occur from the Spotify Chrome Extension

6.3. Additional Features

To enhance the functionality of our Spotify Chrome extension, we implemented several additional features through HTML injection and context menus. For the YouTube "Search on Spotify" feature, we used HTML injection to detect the YouTube video player on the webpage. When detected and fully loaded, our extension inserts a button onto the video page, allowing users to search the current song on Spotify. This logic also checks if other page elements are still loading to avoid inserting the button prematurely, ensuring it only appears on a fully loaded webpage.

For the song search with lyrics feature, we integrated our functionality into existing context menus. This allows users to highlight text (such as lyrics) on any webpage, right-click, and use our extension to search Spotify for the highlighted text. By leveraging the native context menu, we streamlined the process, enabling users to access Spotify song searches directly from highlighted text in a familiar way.

7. Testing

7.1. Test Plan

Our plugin succeeded at the majority of our most important test requirements. The few that it failed were mostly not severe. For certain album covers, notably very dark or light ones with low contrast, the primary and secondary colors are very similar. Another case that it fails is when the plugin fails to load. This is higher severity

because it can affect the end user's experience. Lastly, the search features open in the browser instead of in the Spotify app, which is low priority but still needs work.

7.2 Test Report

Requirement	Pass	Fail	Severity
Display the current song title	1		N/A
Display current song artist	1		N/A
Play the current song	1		N/A
Switch to the next song in the playlist (Next song button)	1		N/A
Switch to the previous song in the playlist (Previous song button)	1		N/A
Volume Control adjusts the volume	1		N/A
Enabling Shuffle changes the order of the playlist	1		N/A
Enabling the Like button adds the song to the Liked Playlist	1		N/A
The color palette is visually pleasing		1	Low
Search for a song on Spotify from YouTube	1		N/A
Search for a song on Spotify with just the lyrics	1		N/A
Spotify Browser opens from the Search Song from		1	Low

Requirement	Pass	Fail	Severity
YouTube feature			
Spotify Browser opens from the Search Song from lyrics feature		1	Low
Spotify Plugin loads without failure		1	Medium

Severity Key:

- High - Core implementation of the program or a hard error preventing user interaction with the Chrome Plugin or Spotify
- Medium - Hard Error or Soft Error that may or may not be preventing user interaction with the Chrome Plugin or Spotify
- Low - Soft Error, bug, or issue that occurs, but doesn't inhibit the user interaction in any way

8. Version Control

We implemented version control for this project using Git on our local machines and GitHub for team collaboration in the cloud. Our project structure included a main branch that served as the production environment and a dev branch for development and testing. Each feature was developed in a separate feature branch following the naming convention "feature/<feature-name>." Once a feature was completed, a team member created a pull request, which was reviewed and merged into the dev branch. After thorough testing in dev, we created another pull request to merge the project into our production environment which was then loaded by the Chrome browser.

9. Conclusion

This project successfully provides users with a seamless Spotify experience through a Chrome browser extension. Our team was able to fit in the necessary features of Spotify controls such as play, pause, next, previous, shuffle, like, and volume, along with new original features such as finding songs by lyrics or searching YouTube videos on Spotify, all of which adhered to project goals of increasing user convenience and functionality. This has been made possible by having a user-centered design and exhaustive testing to make an extension that is both usable and responsive. This project has given us immense insight into some of the technical constraints and requirements that the Chrome extension had to be molded within. These also informed some of the design decisions and the development process. In the future, this may involve the addition of new features or enhancements as seen through user feedback, which would continue to extend the quality of the plugin and enhance the users' listening experience.

10. Appendices

10.1. Appendix A - Project Plan

10.1.1. Project Overview

We all love getting into the zone with our favorite Spotify Playlist, but nothing kills your focus more than constantly having to switch between tabs to skip songs. Fret no more, our Chrome extension allows you to stay in the zone while being able to select your favorite songs.

Our Spotify Chrome extension will display the current song that's playing as well as the song playlist queue. A few features that we would like to offer is the ability to easily open a song from YouTube in Spotify, as well as directly click any text on a webpage, such as song lyrics, and search for it on Spotify.

10.1.2. Project Website

https://seniorproject17.github.io/cs4850-website/

10.1.3. Deliverables

- Team/Project Selection document
- Weekly Activity Reports
- Team Status Report
- Peer Reviews
- Project Plan
- SRS, SDD, STP & Dev Do
- Present Prototype for Peer Review
- Final Report Package
- Final Report
- Source Code
- Website
- Video Demo
- C-Day Application/Submission

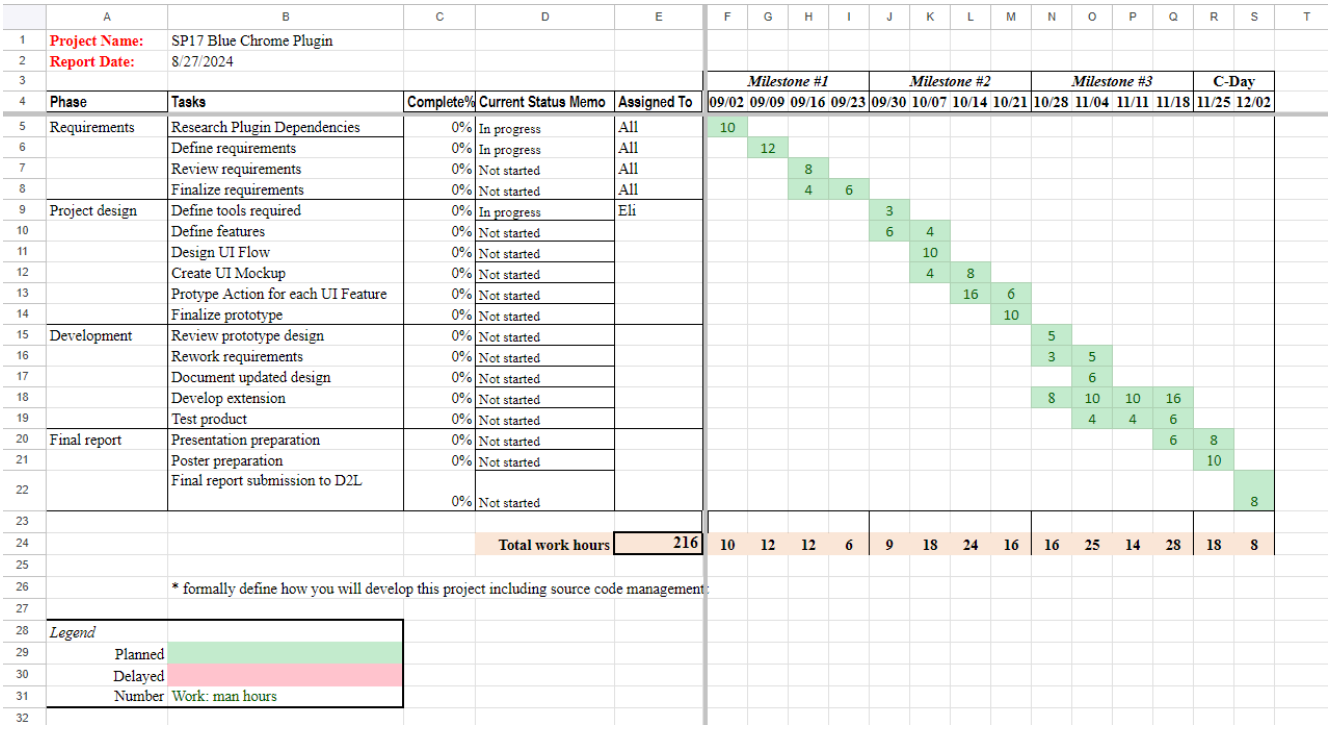
10.1.4. Milestone Events

Solidify Plugin Requirements - By 09/23/2024

Beta Version Complete - By 10/21/2024

Publish Plugin to Chrome Store - By 11/18/2024

10.1.5. Project Schedule and Task Planning



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	**Project Name:**	SP17 Blue Chrome Plugin																		
2	**Report Date:**	8/27/2024																		
3						*Milestone #1*				*Milestone #2*				*Milestone #3*				C-Day		
4	**Phase**	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	09/02	09/09	09/16	09/23	09/30	10/07	10/14	10/21	10/28	11/04	11/11	11/18	11/25	12/02	
5	Requirements	Research Plugin Dependencies	0%	In progress	All	10														
6		Define requirements	0%	In progress	All		12													
7		Review requirements	0%	Not started	All			8												
8		Finalize requirements	0%	Not started	All			4	6											
9	Project design	Define tools required	0%	In progress	Eli					3										
10		Define features	0%	Not started						6	4									
11		Design UI Flow	0%	Not started							10									
12		Create UI Mockup	0%	Not started							4	8								
13		Protype Action for each UI Feature	0%	Not started								16	6							
14		Finalize prototype	0%	Not started									10							
15	Development	Review prototype design	0%	Not started										5						
16		Rework requirements	0%	Not started										3	5					
17		Document updated design	0%	Not started											6					
18		Develop extension	0%	Not started										8	10	10	16			
19		Test product	0%	Not started											4	4	6			
20	Final report	Presentation preparation	0%	Not started													6	8		
21		Poster preparation	0%	Not started														10		
22		Final report submission to D2L	0%	Not started															8	
23																				
24				**Total work hours**	216	10	12	12	6	9	18	24	16	16	25	14	28	18	8	
25																				
26		* formally define how you will develop this project including source code management																		
27																				
28	*Legend*																			
29	Planned																			
30	Delayed																			
31	Number	Work: man hours																		
32																				

10.1.6. Collaboration and Communication Plan

3-4 virtual meetings on Discord to discuss development progress and documentation.

11. Bibliography

"Extensions / Get Started : Chrome for Developers." *Chrome for Developers*, developer.chrome.com/docs/extensions/get-started. Accessed 1 Sept. 2024.

"Web Api." *Web API | Spotify for Developers*, developer.spotify.com/documentation/web-api. Accessed 1 Sept. 2024.