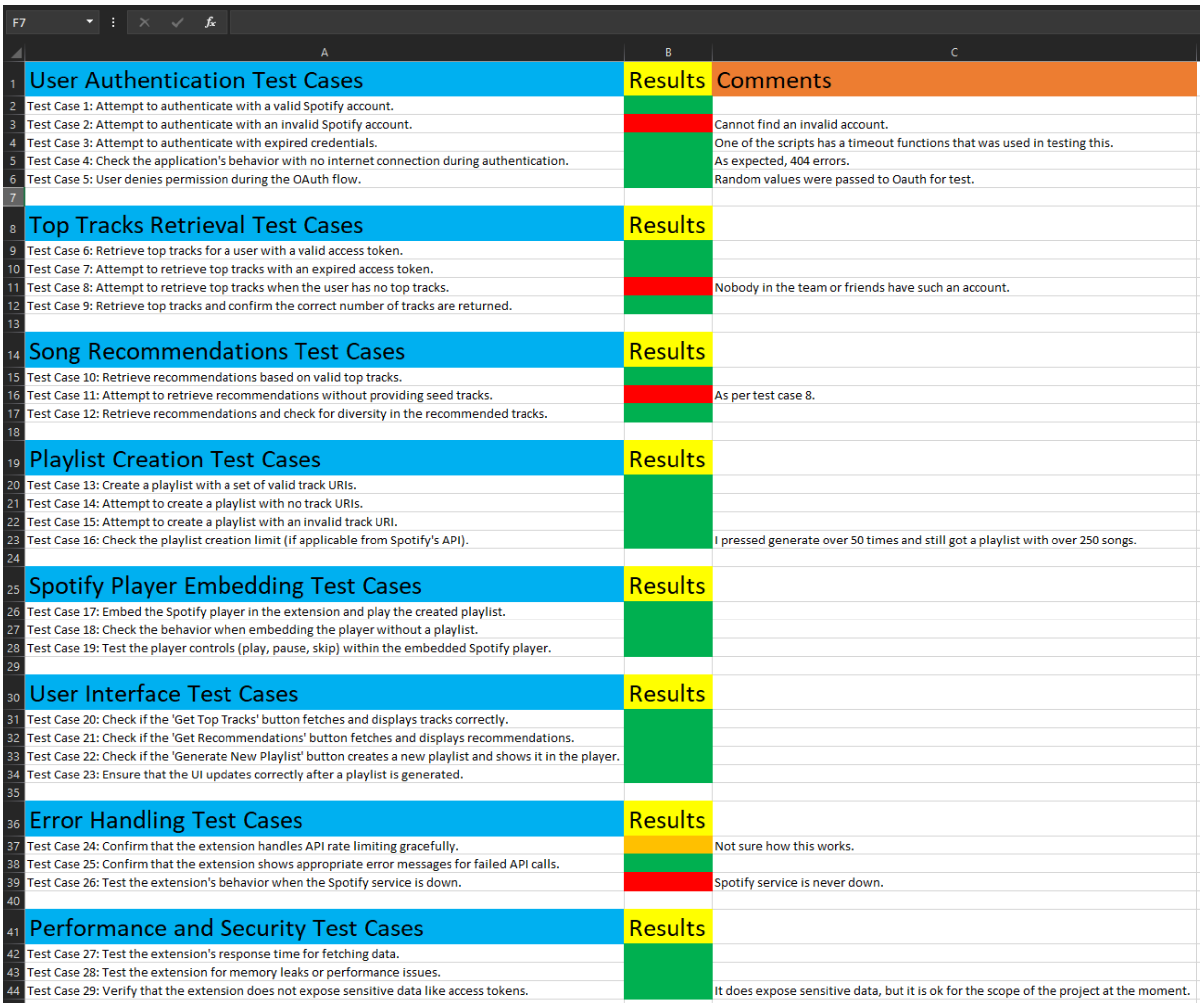
CS322 Music Programming Project
Testing | Results | User Evaluation

The primary objective of testing the Spotify Playlist Enhancer Chrome extension was to validate its functionality across all implemented features and to ensure a seamless user experience. The extension was designed to integrate with the Spotify API, enabling users to authenticate their Spotify account, fetch and display their top tracks, receive personalized song recommendations, and generate new playlists based on these tracks. A key aspect of testing was to confirm that the extension could also successfully embed Spotify's Web Player within the user interface, allowing users to play their new playlists directly from the extension right there and now. It was essential that all the buttons work as expected, no UI or text overlaps, and no crashes or errors.

Test Cases

During one of our team meetings, our team brainstormed the potential test cases that we can all gather and put into one file. We decided that Excel sheet is perfect for this task, so we populated it with all the test cases that we could come up with. We designed the Excel sheet in a way that you have a description of a test case, followed by a result that should be marked as GREEN/RED/ORANGE (Green for pass, Red for fail or untested, Orange for missing tools or other issues), and a comments section. Below is Figure 1 which is the screenshot of the Excel sheet after the development and all testing was completed and all the results were documented. The Excel sheet is also available in the GitHub repository and will be submitted as part of testing documentation.



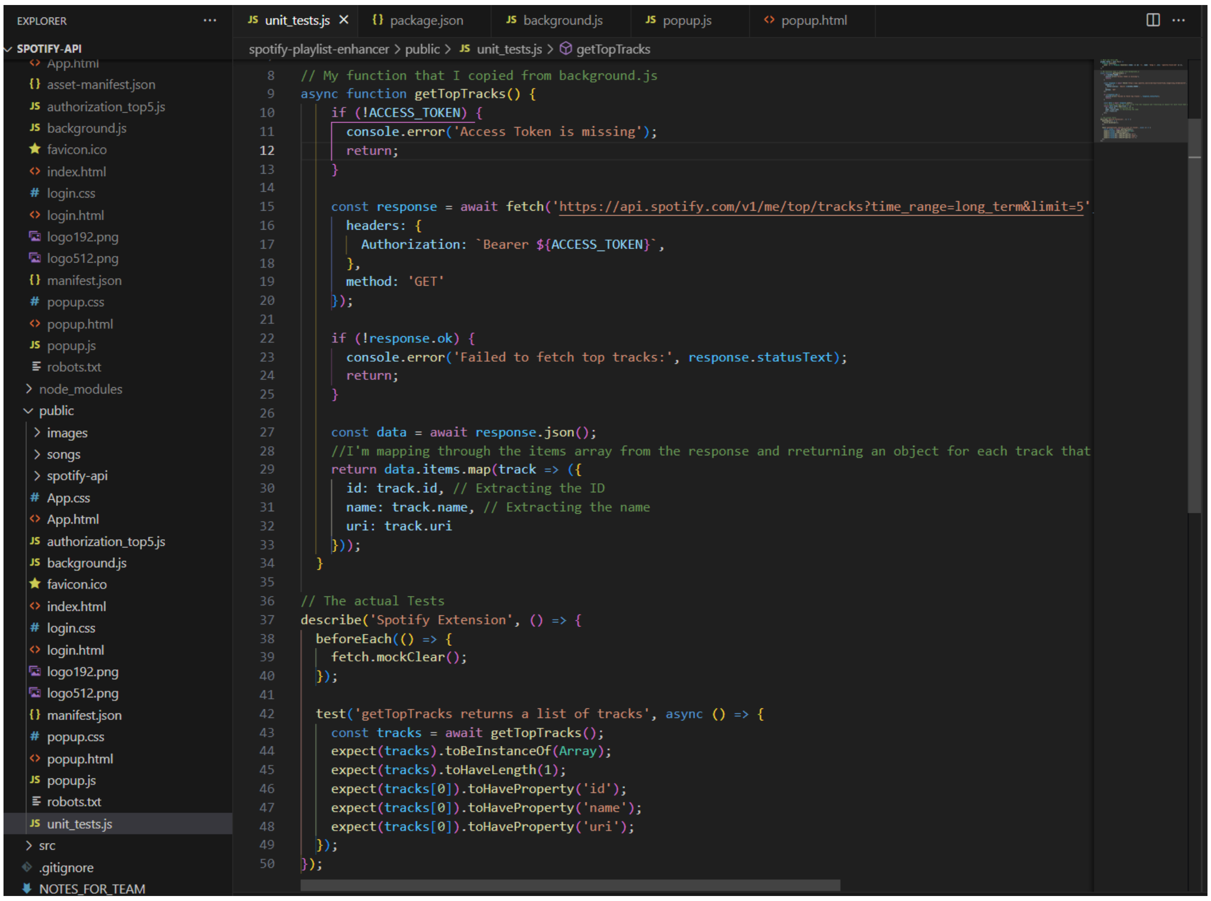
F7

	A	B	C
1	User Authentication Test Cases	Results	Comments
2	Test Case 1: Attempt to authenticate with a valid Spotify account.	🟩	
3	Test Case 2: Attempt to authenticate with an invalid Spotify account.	🟥	Cannot find an invalid account.
4	Test Case 3: Attempt to authenticate with expired credentials.	🟩	One of the scripts has a timeout functions that was used in testing this.
5	Test Case 4: Check the application's behavior with no internet connection during authentication.	🟩	As expected, 404 errors.
6	Test Case 5: User denies permission during the OAuth flow.	🟩	Random values were passed to Oauth for test.
7			
8	Top Tracks Retrieval Test Cases	Results	
9	Test Case 6: Retrieve top tracks for a user with a valid access token.	🟩	
10	Test Case 7: Attempt to retrieve top tracks with an expired access token.	🟩	
11	Test Case 8: Attempt to retrieve top tracks when the user has no top tracks.	🟥	Nobody in the team or friends have such an account.
12	Test Case 9: Retrieve top tracks and confirm the correct number of tracks are returned.	🟩	
13			
14	Song Recommendations Test Cases	Results	
15	Test Case 10: Retrieve recommendations based on valid top tracks.	🟩	
16	Test Case 11: Attempt to retrieve recommendations without providing seed tracks.	🟥	As per test case 8.
17	Test Case 12: Retrieve recommendations and check for diversity in the recommended tracks.	🟩	
18			
19	Playlist Creation Test Cases	Results	
20	Test Case 13: Create a playlist with a set of valid track URIs.	🟩	
21	Test Case 14: Attempt to create a playlist with no track URIs.	🟩	
22	Test Case 15: Attempt to create a playlist with an invalid track URI.	🟩	
23	Test Case 16: Check the playlist creation limit (if applicable from Spotify's API).	🟩	I pressed generate over 50 times and still got a playlist with over 250 songs.
24			
25	Spotify Player Embedding Test Cases	Results	
26	Test Case 17: Embed the Spotify player in the extension and play the created playlist.	🟩	
27	Test Case 18: Check the behavior when embedding the player without a playlist.	🟩	
28	Test Case 19: Test the player controls (play, pause, skip) within the embedded Spotify player.	🟩	
29			
30	User Interface Test Cases	Results	
31	Test Case 20: Check if the 'Get Top Tracks' button fetches and displays tracks correctly.	🟩	
32	Test Case 21: Check if the 'Get Recommendations' button fetches and displays recommendations.	🟩	
33	Test Case 22: Check if the 'Generate New Playlist' button creates a new playlist and shows it in the player.	🟩	
34	Test Case 23: Ensure that the UI updates correctly after a playlist is generated.	🟩	
35			
36	Error Handling Test Cases	Results	
37	Test Case 24: Confirm that the extension handles API rate limiting gracefully.	🟧	Not sure how this works.
38	Test Case 25: Confirm that the extension shows appropriate error messages for failed API calls.	🟩	
39	Test Case 26: Test the extension's behavior when the Spotify service is down.	🟥	Spotify service is never down.
40			
41	Performance and Security Test Cases	Results	
42	Test Case 27: Test the extension's response time for fetching data.	🟩	
43	Test Case 28: Test the extension for memory leaks or performance issues.	🟩	
44	Test Case 29: Verify that the extension does not expose sensitive data like access tokens.	🟩	It does expose sensitive data, but it is ok for the scope of the project at the moment.

Fig 1. Excel sheet for all test cases and their results and comments

Methodology

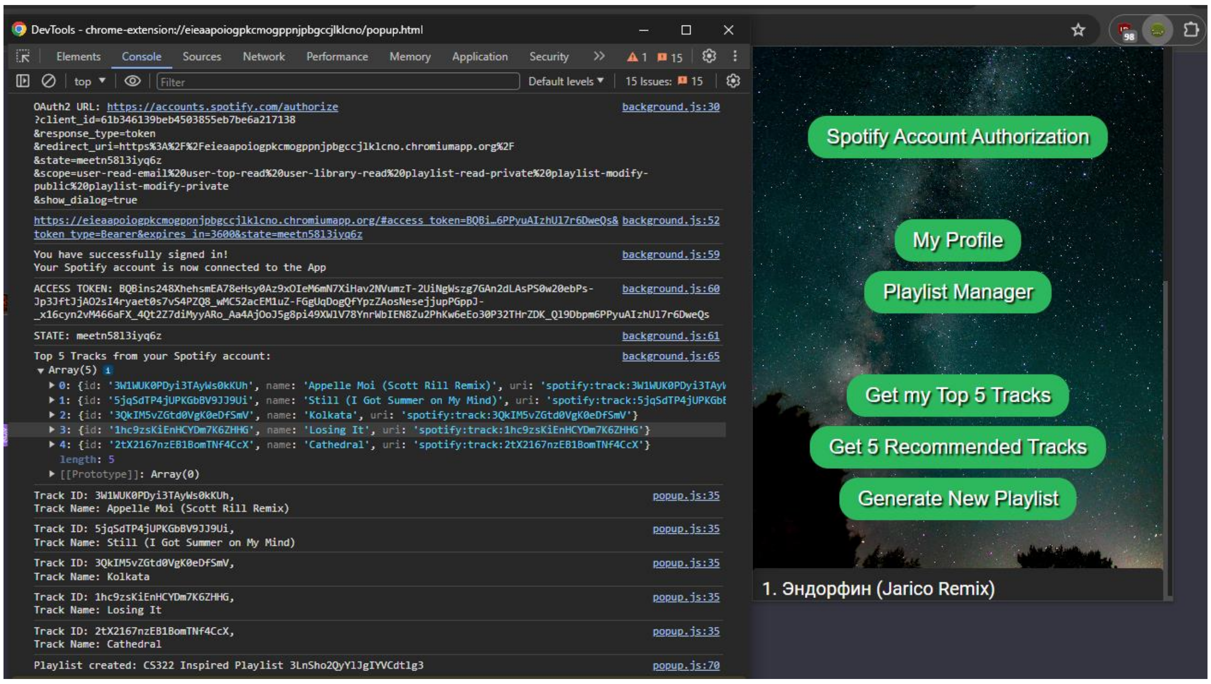
One of the first tests performed were unit tests for a function inside "background.js" back-end file, using a very popular testing framework called Jest. The unit test was performed on "getTopTracks()" function, with the goal to retrieve the 5 tracks in an array, with track name and ID present. The test checked the structure and length of the returned list, ensuring that it matched the expected format and contained the correct number of entries. Below is the screenshot of the test file used with the included function:



```
// My function that I copied from background.js
async function getTopTracks() {
  if (!ACCESS_TOKEN) {
   console.error('Access Token is missing');
   return;
  }

  const response = await fetch('https://api.spotify.com/v1/me/top/tracks?time_range=long_term&limit=5'
   headers: {
    Authorization: `Bearer ${ACCESS_TOKEN}`,
   },
   method: 'GET'
  });

  if (!response.ok) {
   console.error('Failed to fetch top tracks:', response.statusText);
   return;
  }

  const data = await response.json();
  //I'm mapping through the items array from the response and rreturning an object for each track that
  return data.items.map(track => ({
   id: track.id, // Extracting the ID
   name: track.name, // Extracting the name
   uri: track.uri
  }));
 }

// The actual Tests
describe('Spotify Extension', () => {
 beforeEach(() => {
  fetch.mockClear();
 });

 test('getTopTracks returns a list of tracks', async () => {
  const tracks = await getTopTracks();
  expect(tracks).toBeInstanceOf(Array);
  expect(tracks).toHaveLength(1);
  expect(tracks[0]).toHaveProperty('id');
  expect(tracks[0]).toHaveProperty('name');
  expect(tracks[0]).toHaveProperty('uri');
 });
});
```

Fig 2. Unit test file for a function that extracts top 5 songs from user's account

However, upon review, it became evident that due to the relatively small scale and straightforward nature of these functions, the usual unit testing was not necessary. Most of the testing was done manually after this, using print statements in correct places proved to be more efficient and equally effective. Since the Chrome Extension had buttons that already trigger the mentioned functions, and the Chrome browser provides a brilliant console for debugging, our team could swiftly confirm the correct behaviour of the back-end functions by inspecting elements inside the developer console as shown in Fig 3 below.



```
OAuth2 URL: https://accounts.spotify.com/authorize                          background.js:30
?client_id=61b346139beb4503855eb7be6a217138
&response_type=token
&redirect_uri=https%3A%2F%2Feieaapoiogpkcmogppnjpbgccjlklcno.chromiumapp.org%2F
&state=meetn5813iyq6z
&scope=user-read-email%20user-top-read%20user-library-read%20playlist-read-private%20playlist-modify-
public%20playlist-modify-private
&show_dialog=true

https://eieaapoiogpkcmogppnjpbgccjlklcno.chromiumapp.org/#access_token=BQBi…6PPyuAIzhUl7r6DweQs& background.js:52
token_type=Bearer&expires_in=3600&state=meetn5813iyq6z

You have successfully signed in!                                            background.js:59
Your Spotify account is now connected to the App

ACCESS TOKEN: BQBins248XhehsmEA78eHsy0Az9xOIeM6mN7XiHav2NVumzT-2UiNgWszg7GAn2dLAsPS0w20ebPs-     background.js:60
Jp3JftJjAO2sI4ryaet0s7vS4PZQ8_wMC52acEM1uZ-FGgUqDogQfYpzZAosNesejjupPGppJ-
_x16cyn2vM466aFX_4Qt2Z7diMyyARo_Aa4AjOoJ5g8pi49XWlV78YnrWbIEN8Zu2PhKw6eEo30P32THrZDK_Q19Dbpm6PPyuAIzhUl7r6DweQs

STATE: meetn5813iyq6z                                                       background.js:61

Top 5 Tracks from your Spotify account:                                     background.js:65
▼ Array(5) i
  ▶ 0: {id: '3W1WUK0PDyi3TAyWs0kKUh', name: 'Appelle Moi (Scott Rill Remix)', uri: 'spotify:track:3W1WUK0PDyi3TAyW
  ▶ 1: {id: '5jqSdTP4jUPKGbBV9JJ9Ui', name: 'Still (I Got Summer on My Mind)', uri: 'spotify:track:5jqSdTP4jUPKGbE
  ▶ 2: {id: '3QkIM5vZGtd0VgK0eDfSmV', name: 'Kolkata', uri: 'spotify:track:3QkIM5vZGtd0VgK0eDfSmV'}
  ▶ 3: {id: '1hc9zsKiEnHCYDm7K6ZHHG', name: 'Losing It', uri: 'spotify:track:1hc9zsKiEnHCYDm7K6ZHHG'}
  ▶ 4: {id: '2tX2167nzEB1BomTNf4CcX', name: 'Cathedral', uri: 'spotify:track:2tX2167nzEB1BomTNf4CcX'}
    length: 5
  ▶ [[Prototype]]: Array(0)

Track ID: 3W1WUK0PDyi3TAyWs0kKUh,                                           popup.js:35
Track Name: Appelle Moi (Scott Rill Remix)

Track ID: 5jqSdTP4jUPKGbBV9JJ9Ui,                                           popup.js:35
Track Name: Still (I Got Summer on My Mind)

Track ID: 3QkIM5vZGtd0VgK0eDfSmV,                                           popup.js:35
Track Name: Kolkata

Track ID: 1hc9zsKiEnHCYDm7K6ZHHG,                                           popup.js:35
Track Name: Losing It

Track ID: 2tX2167nzEB1BomTNf4CcX,                                           popup.js:35
Track Name: Cathedral

Playlist created: CS322 Inspired Playlist 3LnSho2QyYlJgIYVCdtlg3            popup.js:70
```

Fig 3. Google Chrome extension opened side by side with Chrome's developer console (DevTools).

Figure 3 above shows multiple tests performed during the normal use of the web application. For example, the maximized array in the DevTools window shows that the applications successfully extracted top 5 songs, and printed their ID's, names and URI's. There are many more useful print statements and console tools that helped our team with testing of the application.

Integration testing was also discussed within our team, and we came to the conclusion that it was not necessary again, just because Chrome's DevTools have so many options and different uses. Jevgenij created a messaging system that communicated events between the back-end and the front-end UI of the extension, and integration testing seemed like a perfect test to ensure the messaging works as expected, but Chrome's DevTools proved to be much quicker and efficient yet again.

Results

The testing phase of the Spotify Playlist Enhancer Chrome extension yielded positive outcomes, confirming the functionality and reliability of its features. All API calls were executed as expected, with endpoints responding correctly and data being retrieved and processed without any errors. Functions such as "getTopTracks()" and "getRecommendations()" performed with precision, accurately reflecting the users' music preferences in the application.

The createPlaylist() feature stood out, functioning seamlessly to generate new playlists that were instantly populated with selected tracks. This feature's robustness was particularly noteworthy, as it managed to handle various scenarios without any hiccups, for example having more than 100 songs generated into a playlist at once. The user interface elements, including the buttons for initiating actions within the extension, worked flawlessly. Each button triggered the correct sequence of events, and there was no overlap or interference between the different UI components, ensuring a smooth user experience.

Furthermore, the extension's layout and design maintained its integrity across different screen sizes and conditions, showcasing adaptive and responsive behaviour. The extension had no visible issues when it was opened on large 4K and 1440p monitors, or laptop screens of different sizes. Reducing the Google Chrome's browser window to small sizes also did not affect the performance of the app and all UI elements were still visible as shown in Figure 4 below.

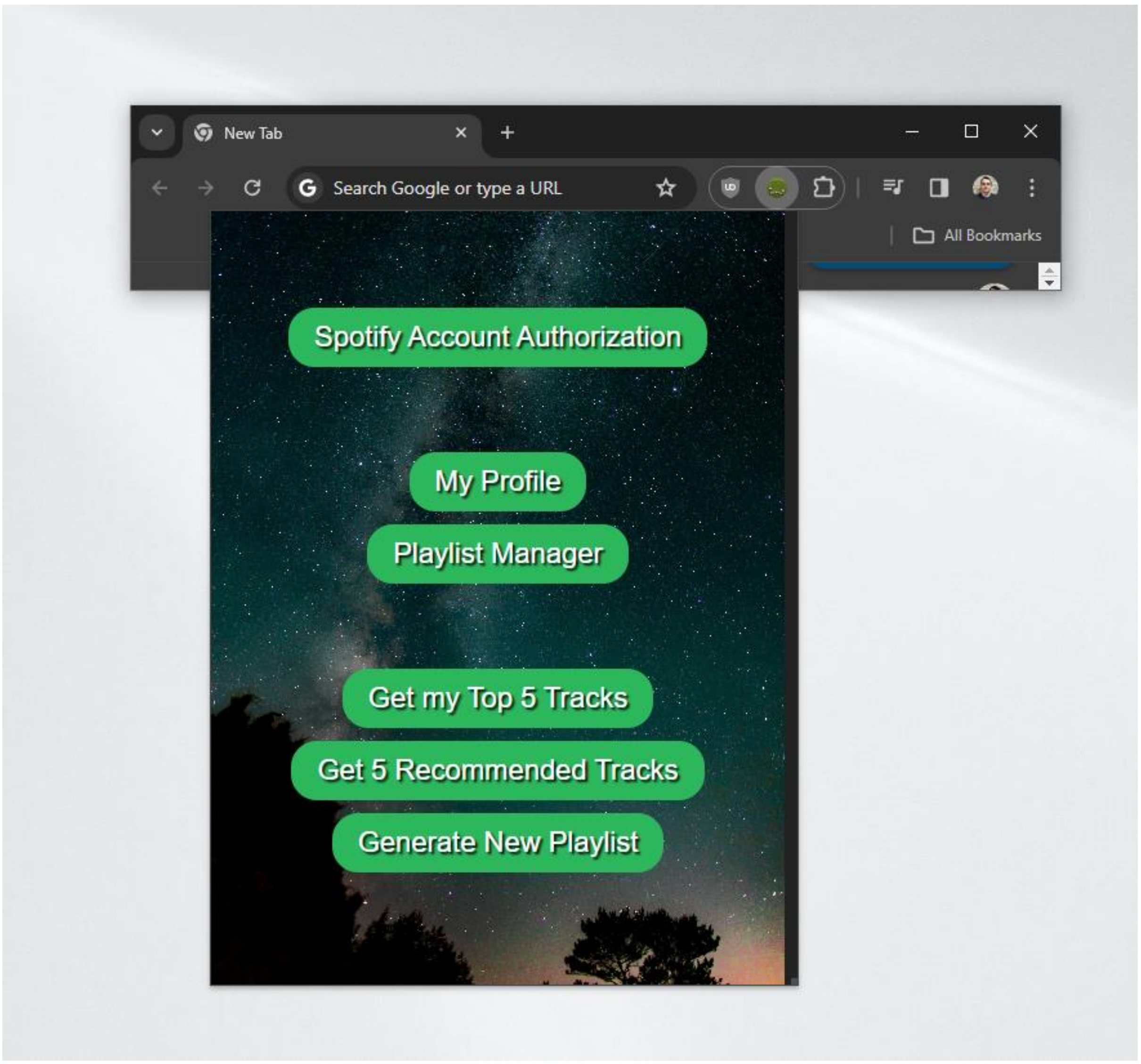


Fig 4. Google Chrome browser window in its minimum size

In essence, the extension delivered on its intended capabilities, with each component from authentication to playlist creation working in tandem to provide a cohesive experience. The results of this testing phase affirm the project's readiness for a broader development and scalability, with confidence that the end users will find the app both practical and fun to use.

User Evaluation

Our team decided to approach some of the students that are in the same course as us, and ask them to install our application and try to break and crash it somehow or find errors. A lot of students denied us as they were busy with their own projects or were just not interested. We then offered to buy a Monster energy drink to anyone who can crash our application, and we found two students that saw this as a challenge, and took up on the offer. Below are two feedbacks provided by our fellow 4th year students, Stillian Kolev and Bartek Kowal.

Stillian Kolev, 4th year Computer Science student:

"I installed the Spotify Playlist Enhancer on my Chrome browser and tried various ways to test its limits. To my surprise, I couldn't make it crash or encounter any significant issues, which speaks volumes about its stability. What I really liked about the app was its user-friendly interface. It was incredibly smooth to create playlists that felt tailored to my music preferences. Being able to play these playlists directly within the extension was a delightful feature. However, I think it would be great if there was a way to modify the playlists I've created. This added functionality would provide more flexibility and enhance the user's experience. Another tiny detail that I've noticed, is that when the extension UI is opened, there is a flash of white background for a split second, and it bothers me."

Bartek Kowal, 4th year Computer Science student:

"After installing the Spotify Playlist Enhancer, I thoroughly explored its features, attempting to find any potential weaknesses. Impressively, the app remained stable and responsive throughout my testing. The accuracy of the music recommendations stood out to me; it seemed to understand my musical taste quite well. I also appreciated the seamless transition between different functions in the extension – it made the experience very fluid. One suggestion I have is to include a feature for tracking the history of created playlists. Being able to revisit and possibly edit past playlists would make the app even more useful and engaging."

After receiving the insightful feedback from the two students, our team took the time to carefully review each comment and suggestion. Their experiences and perspectives were invaluable in understanding how real users interact with the Spotify Playlist Enhancer Chrome extension. After reading Stillian's review, our team was shocked to learn about the white background flashing for a brief moment before loading the background image, as nobody in our team noticed this before. This was a great catch, and we immediately added this suggestion to "Things to work on" document. As for the modifying playlists feedback, this was already planned and development has started. Bartek's feedback was also very valuable to us. A feature for tracking the history of all the playlists created previously would greatly enhance the user experience, as sometimes you hear a good song but lose it after time, and finding the playlist where it was could solve this problem.

Conclusion

Looking back on the entire process of making and testing the Spotify Playlist Enhancer, it's clear we've learned a lot. We set out to make something that was easy and fun to use, and the tests we did showed us that we're on the right track. We chose to test things in a simple, hands-on way, using tools that are easy to understand but are efficient. This approach worked really well for our project, and we are confident it will continue to do so.

All in all, this project has been a great experience. We've made something that works well and that people seem to enjoy. We're excited to keep improving it, using the feedback and ideas we've gathered along the way.