Design Inspection, Code Inspection and Unit Testing

Brian Duffy, Akhil Agrawal, Adam Johnston, Shivan Desai

Design Inspection

Defect #	Description	Severity	Correction
1	The Login component would redirect before a user could enter their credentials	1	We implemented defer statements which act like mutex locks. This essentially blocks until a timer runs out, an error occurs, or the login is successful.
2	In order to test our web app, we had to enable a plugin that adds access control allow origin. This was a temporary solution which didn't work out when it came to testing the app.	2	We hosted the app on firebase and now the plugin is no longer needed.
3	During testing, we were having issues with asynchronous functions finishing before others. This made testing pointless in the sense that the intended control of the app was not being achieved.	1	We implemented promises and chained them one after another. This enforces the intended control flow of our app.
4	Jasmine tests ended up being more of a hassle to set up and work with synchronization. They were especially problematic when calling the spotify api.	1	We switched testing methods from unit tests to end to end tests using Protractor. This gave us more valuable tests because it tested multiple components including the interaction with the spotify api.
5	Upon hosting of our app on firebase, we ran into database privilege issues. Even though someone was logged in, they could not view what was in the database.	1	We ended up deciding that making our database "public" fixes this issue. This is accepted as a temporary fix mostly because people who aren't

			logged in cannot add to the playlist without being authenticated through spotify. So we kind of have read only access to those who aren't logged in to spotify.
6	The testing and current build use the same callback address. In other words the testing uses the actual deployed website as it's means of running the tests.	2	This isn't inherently wrong, but it is bad practice due to potentially overusing our quota for the amount of data provided by firebase for free. If we exceed this, we pay real money. Luckily firebase can run locally so we changed the addresses to use the local server as opposed to the firebase server.

Code Inspection

Defect #	Description	Severity	Correction
1	During testing, a lot of await calls for asynchronous testing lead to race conditions.	2	Using chained promises we were able to essentially wait until one asynchronous call finished until going on to the next.
2	Searching through arrays returned by observables proved to be difficult to parse into readable data.	2	We realized we didn't need all of the data in most cases. Most of the time, we simply needed to know whether or not the array contained a value, etc, so we simply used filter statements to reduce

			down the information we had to parse through to a single statement.
3	When using Observables, we were not properly handling error statements/conditions	2	We have begun handling this by adding not only the onNext method inside .subscribe, but onError as well, which allows us to trace the error more easily or handle the errors.
4	We had a lot of repeated code in testing and it made testing unorderly and hard to follow.	2	We fixed this by added functions in app.po.ts that act as macros cleaning up the code.

Unit Testing Defects

At first we were using Jasmine as our testing framework but due to complications with testing the actual app interactions with spotify, we chose to test the app as an app. At this point there was no need to use drivers and stubs as we have a semi working app at the moment. We are now using Protractor which is fully automated and actually interacts with elements of the app.

Login Component:

This component is where users get access to our web app. Here they click the login button and a new window opens where they input their Spotify username and password. They are then redirected to the Home Component.

Defect #	Description	Severity	Correction
1	When logging in, users were redirected before they entered their credentials.	1	We ended up using a defer statement which waits until a condition occurs, an error occurs, or a successful event. Through testing we figured out

			that the issue was caused by not managing asynchronous method calls properly.
2	When logging in for the first time, users had to login twice.	2	This was fixed by using defer just like the previous defect. In doing this, we fixed the control flow of the app when authorizing.
3	After logging in, the webpage failed to redirect after a successful login.	2	We ended up using async/await to wait for the promise returned from the authenticate method. Using the await keyword we were able to delay the async methods to behave like synchronous methods. Then we navigated to the home component.

Home Component:

The home component is going to be where we display the songs of the playlist and the users listen to the songs in the playlist. In other words this will be our main component for user interaction.

Defect #	Description	Severity	Correction
1	Upon navigation to the home page after login, users could not see what is in the UpNext playlist.	1	The cause of this was from an issue with authentication. The error occurred every first time login. The issue was solved by using defer statements during authentication. When the defer statement was resolved we navigate to the home page.
2	The Home component doesn't	1	We fixed this by retrieving the

	render the playlist properly in the Spotify web player iframe.		playlist before navigating to this page after login. This ensures that there is relevant data being retrieved from the spotify api.

Search Component:

This component is named the Radio Component but we refer to it as the Search Component because the name fits better. Here users can search for tracks, artists, and albums. In addition to searching, they can also add a song to the UpNext Playlist so long as they are logged in.

Defect #	Description	Severity	Correction
1	Searches would return a 401 error even though a user had been authenticated.	1	The issue derived from the temporary “fix” of the plugin that adds access control allow origin. Hosting the app on firebase fixed this issue.
2	When adding a track to a playlist, we would continually create duplicate playlists.	1	It turned out the method in which our Observable determining whether the playlist already existed or not was always returning false, as it would continue the program even while the Observable was still processing. So even if the name of the playlist was contained in our return values, they didn’t get observed until the method had already returned. We fixed this by using the filter method to check if a playlist already existed for a spotify user’s account in combination with the array .some operator.

3	Users can add duplicate tracks to the playlist.	3	We might be able to fix this issue with a function call that removes duplicate tracks. We could also get the playlist and iterate through the track names. If there is a match, then log an error and don't add the requested song to the playlist.
4	We were trying to get json data and the values we needed were wrapped inside a json key named items.	3	We had to map on the data returned by the spotify call before we subscribed to it. Instead of subscribing on the whole json result, this allowed us to process just the items array.

Header Component:

The Header Component is where users can easily navigate our app. A menu with all the possible webpages is present and clicking a link switches out the injected html with the component that corresponds to the link clicked.

Defect #	Description	Severity	Correction
1	In mobile view, when a user taps an item in the menu, the app navigates to the page but the menu remains open.	3	This issue can probably be fixed using an event listener or possibly an http interceptor. Upon a url change, the menu collapses.