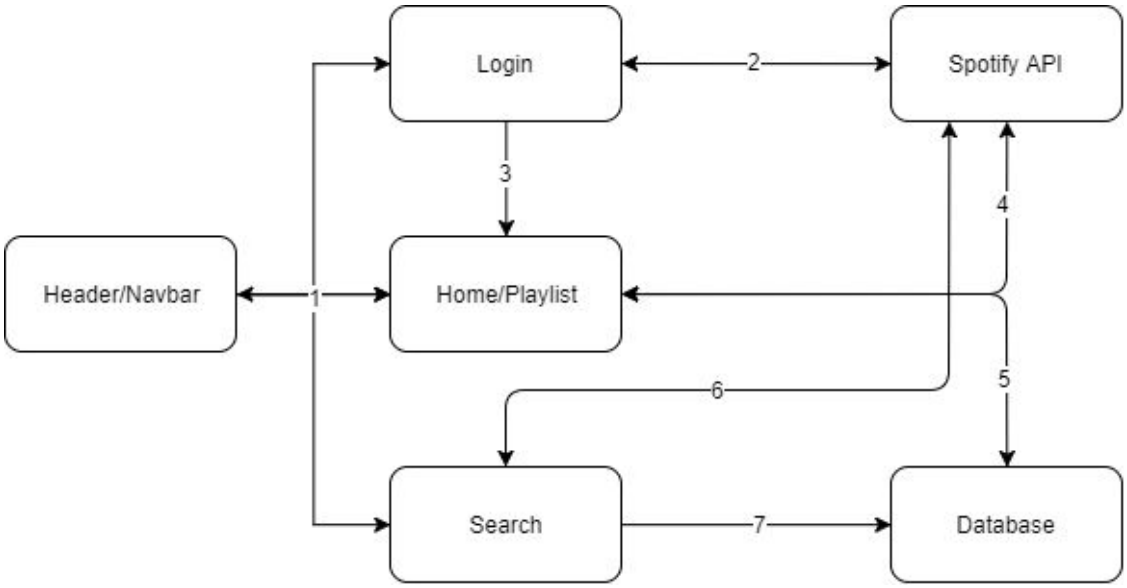
Incremental and Regression Testing

Adam Johnston, Brian Duffy, Akhil Agrawal, Shivan Desai

1. Classification of Components



1. Navigates between components
2. Authenticate User and Retrieve Auth Token
3. Redirect After Login
4. Check if Playlist Exists, Update with New Tracks
5. Request and Retrieve Tracks for Playlist
6. Search for and Retrieve Tracks to Add to Playlist
7. Add Tracks to Database

Input/Output/Dependencies

1. Header Component
    a. Inputs:
        i. Button press
        ii. Upvote/ Downvote
    b. Outputs:
        i. Local redirection to other components
    c. Dependencies (Called by):
        i. App Component (the app itself)
    d. Dependencies (Calls):
        i. Login, Search, Home, and Callback Components
2. Login Component
    a. Inputs:
        i. Button press to login
    b. Outputs:
        i. Redirection to the Home Component
        ii. Spotify authentication token
    c. Dependencies (Called by):
        i. Header Component
    d. Dependencies (Calls):
        i. Home Component
        ii. Spotify authentication API
3. Home Component
    a. Inputs:
        i. Button press to play a song
        ii. Button press to open the spotify app
        iii. Button press to navigate to login page
    b. Outputs:
        i. A playlist of songs in the database
        ii. An iframe that the user can click to play a song
    c. Dependencies (Called by):
        i. Header and Login Components
    d. Dependencies (Calls):
        i. Spotify API
        ii. Firebase realtime database
4. Search Component
    a. Inputs:
        i. Search query term to search for tracks
        ii. Submit button
    b. Outputs:
        i. List of tracks that relate to the search query term

ii. Button to add tracks to the playlist

c. Dependencies (Called by):
    i. Header Component
d. Dependencies (Calls):
    i. Spotify API
    ii. Firebase realtime database

Which Form of Incremental Testing

We are using Bottom-Up Testing rather than Top-Down, as our test cases contain “Drivers” that call our submodules. We have already created our submodules and are testing their interactions and outputs through the use of said drivers. We do this instead of calling submodules that do not exist, which would be replaced with stubs in Top-Down testing. As an example, we have a test case making sure the Search component is retrieving results correctly. The Search feature is implemented, so we use a Driver with preset search values to make the search call to query the Spotify API and retrieve the data.

2. Incremental and Regression Testing

Automation

The last sprint we used exclusively karma and jasmine which was somewhat automated. Once we got more and more components working, we decided to migrate our testing to Protractor. Protractor does use jasmine testing framework and further builds on it for more wholesome testing. Now our testing methods are fully automated in the sense that we are testing backend and front end with protractor in a headless chrome browser. We also combined the jasmine tests and the Protractor tests in one test all script.

Home Component

Home Component	Incremental Testing		
Defect #	Description	Severity	How To Correct

1	When we added the spotify web player iframe, it was visible when it shouldn't be.	2	We added conditional statements in the html to essentially change the visibility of the elements based on criteria like user id and spotify authentication token validity.
2	Sometimes when we redirected to the home page, a graphical issue would arise in the place of the iframe.	1	We fixed this graphical issue with async await statements forcing the page to wait until the necessary promises were handled before showing unresolved data.
3	The homepage wasn't very user friendly when it came to directing an unauthenticated user to use the app.	3	We added a button that redirects to the login page. The button is invisible whenever the user id is valid.
4	The voting system was having problems whenever one track was bypassing another track (e.g. a track with 4 votes was being voted to a track with 5 votes).	1	We figured out that we were not properly associating keys retrieved from the database with their respective tracks due to the tracks being sorted and the keys being left unsorted. Additionally, the upvote methods were unnecessarily incrementing the track as they would be refreshed as soon as the database realized the change anyways.

Home Component	Regression Testing		
Defect #	Description	Severity	How To Correct

1	We previously had a bug where users who were not logged in could see the playlist	1	We added ngIf statements checking if the user was not the default user id that we originally set. We also checked to make sure that they had a spotify authentication token as well.
2	We had previously had some issues where the user was redirected to the home page before finishing logging in.	2	We were sometimes receiving defer errors. So we further increased the timeout duration of the authentication window.
3	We had issues with duplicates items showing up in the database.	2	We added checks to see if the database already had a particular track id. If it did, then we displayed an alert message.
4	We noticed that overflow and underflow for the voting system could occur.	1	We set upper and lower bounds to voting so those issues would not occur.

Login Component

Login Component	Incremental Testing		
Defect #	Description	Severity	How To Correct
1	Sometimes the login page failed to redirect to the home page.	1	We found out that the timeout for the authentication window was far too short. So we lengthened it.

2	Occasionally we would see a graphical issue where the app was embedded into where the iframe would be.	1	We added async await statements in the login component logic to wait for the unresolved promises to become resolved. Then we navigated to the home page.

Login Component	Regression Testing		
Defect #	Description	Severity	How To Correct
1	In order to do anything with our app, we must login to spotify.	1	Extensive testing was done with logging in out of necessity in order to test other functionality that deals with access to the spotify api.
2	After implementing the defer statements in order to keep the app from navigating to the home page before authentication was complete, we ran into some issues with defer statements timing out.	1	We figured out that the issue was to short of a timeout for the authentication window.

Search Component

Search Component	Incremental Testing		
Defect #	Description	Severity	How To Correct
1	We previously had issues with users able	2	We examine the query term beforehand to make

	to enter invalid input like an empty string.		sure it is valid. If it is not, then we open up an alert dialog message.
2	We previously gave no information for why a search did not complete.	2	We now check to make sure the user is in fact logged in through the use of their spotify user id and the presence of a valid spotify authentication token.

Search Component	Regression Testing		
Defect #	Description	Severity	How To Correct
1	We needed to make sure that the search functionality was robust in handling odd user input.	2	We added a few tests that try to search when a user is not logged in and the input is invalid.
2	We previously did not handle duplicates in the playlist.	1	We added checks to see if the database already had the track id that a user is trying to add to the playlist. If it is present, we show an alert dialog box.

Header Component

Incremental Testing

As of the last sprint, we had already connected all pieces of the app to this central component. So as for integration testing there is nothing more to test. We did however do some regression testing to ensure the robustness of this central component.

Header	Regression Testing

Component			
Defect #	Description	Severity	How To Correct
1	We needed to test the robustness of our app when navigating.	1	Instead of using page.navigateTo() method which basically hardcodes in a url then goes to it, we tested it by invoking clicks on the pieces of the header to better test the header component's robustness.
2	We felt it necessary to try and invoke any graphical issues from navigation through our app.	2	In order to test this effectively we minimized the amount of time the browser spent sleeping during automated testing in protractor. In some instances it was necessary to make the browser sleep to accommodate large responses from large promise chains. An example of this would be when logging in.