**UX**

* Goal is to create a tool that allows many people to contribute to and manage a playing list of songs – but that also can be ignored and untended safely at any point
  + Easy setup for playing user
  + Easy contribution for other people
  + Hands-off capable
* Ideally the onboarding would be as easy as possible too, however this isn’t a priority as the setup and interaction is the main focus
* This is not a discovery tool
  + Therefore, no searching meta data, queries, etc. just a simple artist, track, album lookup
    - Allow importing, exporting playlists
    - Allow direct id input
* This is not a networking / following / social tool
  + Therefore no need for special profile elements
  + No searching for other users, one must know them by real-life referral
  + No need for private playlists? (visibility wise, not edit/playback permissions)
* This is a playlist curation & storage tool
  + Copy & follow (?) other people’s playlists
  + Organize playlists
  + Reasonably useful merge, move, pair, etc. functionalities
  + It should be possible to import and export and sync(?) likes and playlists with different services
* It should be possible to do every non-playback functionality while another playlist is playing (a user playing a playlist should not have any reduced functionality)
* Would the user ever need to be editing two playlists at once?
  + No? but the search and add functionalities should have two modes:
    - Targeting a playlists which gives users more control of where and how tracks are inserted
    - General, which lets users search for tracks and select which playlists to put it into
  + But this should also be able to be done with other people’s playlists
* Search/adding to playlists should be an ‘edit’ mode with two distinct panels, the search panel (which can also be swapped out to look at other user’s playlists), and the focused playlist panel (which can also be zoomed out to look at all a user’s playlists/folders)
* Maybe if the user can create folders of playlists they will also have a section of foreign playlists:
  + Followed
  + Edit access
  + Playback access
  + These could actually just have different borders or ‘types’ where the user can then organize them with their own playlists
* Permission types:
  + View only
  + Edit
    - Add only
    - Add, move
    - Add, delete, move (all)
  + Playback

**Comments**

* // TODO todo
* // ! important note
* // ? confusion
* // E explanation
* // L resource link
* // R reflection
* //

**Feedback**

* Give me all your use-cases
* Did a button look at you funny? I want to know

**Inspiration**

* PartiQ

**Postgres Database**

* TODO
  + ~~Ensure database uses ‘title’ instead of ‘name’ (for consistency) as it is a reserved word in javascript~~
  + Name is only reserved as a property of **functions**, will never have any cross over, also, any reserved words in SQL can be used via “quotes”
* Use surrogate primary keys (id column instead of a combination of existing columns (natural key))
  + as primary keys should not be changed
    - <https://stackoverflow.com/questions/3838414/can-we-update-primary-key-values-of-a-table>
  + Surrogate keys are simpler to implement
  + <https://dba.stackexchange.com/questions/8334/why-should-i-create-an-id-column-when-i-can-use-others-as-key-fields>
  + <https://www.techrepublic.com/blog/10-things/10-tips-for-choosing-between-a-surrogate-and-natural-primary-key/>
* camelCase
* Storing Playlists:
  + <https://dba.stackexchange.com/questions/21664/how-to-store-several-lists-and-their-items>
  + (artists won’t get its separate table as I’ll never want to search artists or split them from the track)
* **Primary and Foreign Key naming conventions**
  + <https://stackoverflow.com/questions/1369593/primary-key-foreign-key-naming-convention>
  + Only foreign keys should be named (foreignId)
* **Create Primary Key**
  + <https://www.techonthenet.com/postgresql/primary_keys.php>
  + Use CONSTRAINT
    - CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, …)
    - Or
    - column\_name data\_type CONSTRAINT constraint\_name PRIMARY KEY
  + **Auto increment**
    - Use SERIAL
      * <https://stackoverflow.com/questions/787722/postgresql-autoincrement>
      * column\_name SERIAL
  + Together
    - column\_name data\_type SERIAL CONSTRAINT constraint\_name PRIMARY KEY

**New Server Setup**

* Download and install node js (LTS)
  + <https://nodejs.org/en/>
  + <https://github.com/LeCoupa/awesome-cheatsheets/blob/master/backend/node.js>
* npx
  + default with npm, runs local packages
  + <https://blog.npmjs.org/post/162869356040/introducing-npx-an-npm-package-runner>
  + <https://stackoverflow.com/questions/50605219/difference-between-npx-and-npm>
* **nodemon** for live updates
  + npm install nodemon –save-dev
    - will install locally in the devDependencies
  + start server with nodemon instead of node
* npm-run-all
  + for running webpack and server at same time
  + <https://github.com/mysticatea/npm-run-all>
  + <https://medium.com/netscape/npm-task-running-techniques-15fe5b697f15>
  + other options were pm2 and concurrently
    - <https://medium.com/dailyjs/vue-js-simultaneously-running-express-and-webpack-dev-server-292f4a7ed7a3>
* use scripts inside package.json to make a shorthand for commands
  + used with ‘npm run [script]’
  + <https://stackoverflow.com/questions/33558396/gulp-webpack-or-just-webpack>
* --experimental-modules flag lets node use the module system
* Download and install postgre sql <https://www.postgresql.org/download/>
  + Start postgres via taskmanager services – postgres service
  + DBeaver for the gui <https://dbeaver.io/>
    - Username: postgres Password: my own password
* Create database (using dbeaver)
* npm install pg-promise

**Future Ideas**

Include not a commenting system but sort of a tracking system for when a user ‘likes’ a track, or if they ‘highlight’ a part of a track (sortof like new websites that allow highlighting and commenting on certain parts of an article), or maybe a most listened part of the track (like pornhub)

Allow users to do basic account functions from the site (liking, sharing, reposting, add to playlist, etc) based on the accounts connected, also have a sort of toggle system for if the accounts should be personal (library generating) or general (permission granting)

**Aesthetic**

Non cliché buttons/icons? <http://www.abc.net.au/triplej/programs/mix-up/whatsonot/9861470> , like just a simple box with a word to say ‘listen’

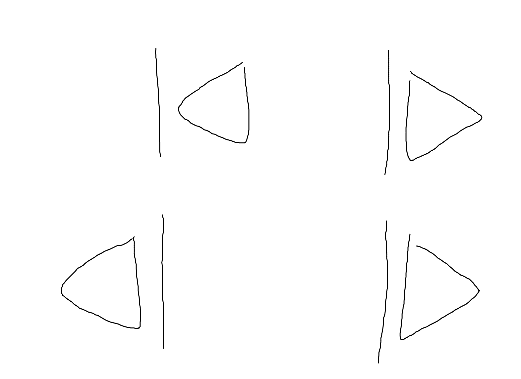
Large splashes on everything (using album art, waveforms, set colors, procedural etc)

Or, maybe a sort of designy, albumArt, style (check out DELAY. Cover art for example)

Or, minimalistic black and white style like dbr.ee

**Behavior**

Prev track button changes icon based on if it will start the previous track or just skip to the start of the current one



**Schema**

Accounts probably cant be stored with the user because they have no use, cant use stored name/passwords to log in anyways, these would probably be limited to the session

users(userID, username, password, email, *preferences:* image, color, feedSetting,

playlists(ownerID, playlistID, title, description, *settings:* image, color, visibility,

tracks(playlistID, index, uri, source, *stored details:* title, artists, length, image

private

link-only

public

PHP Web API wrapper and Javascript Web API

Can use api to identify available devices

Can use api to actively play on a targeted device

Account connected must be the one also connected to the device

Php curl is a synchronous request

Curl sends a request to a url and gets the response back to use in code

Ajax is an asynchronous request

Connect response types

<https://developer.spotify.com/web-api/working-with-connect/>

<https://beta.developer.spotify.com/documentation/web-api/>

PHP Wrapper:

<https://github.com/jwilsson/spotify-web-api-php>

Javascript Wrapper:

<https://github.com/JMPerez/spotify-web-api-js>