Team B3: **Thomas Lee**, Luke Marolda, and Matt Hegi

music mirror

Use-Case

- A comprehensive speaker attachment that seamlessly manages queuing, song recommendations, and crowd engagement
- Users steer the system through a distributed web app that hosts a suite of song request and consensus voting capabilities

Existing Solutions

- Current systems are singular - they focus on one person having full control. We democratize the event listening experience for uniform enjoyment

Areas

- Software Systems, Machine Learning, Hardware Systems

Design Requirements

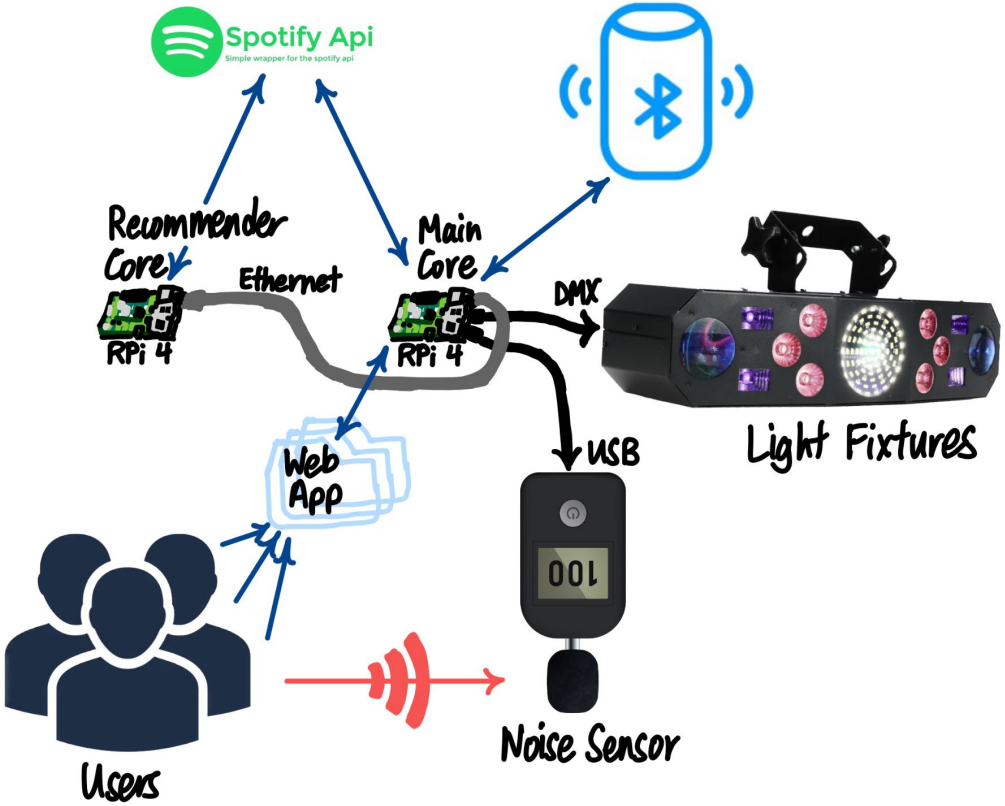
- System ability to mount to any functional Bluetooth speaker
- Light system colors and strobing match song genre, tone, and crowd loudness noise sensor
- Continuously generate recommendations based on previous User requests
- **3** direct song request formats. Implemented with a semantic matching algorithm to map requests with queried spotify resources
  - By name of song
  - By artist or album
  - By songs that have already been played
- **1** additional song request format: Similarity search
  - Ability to generate song requests based on what has already been played

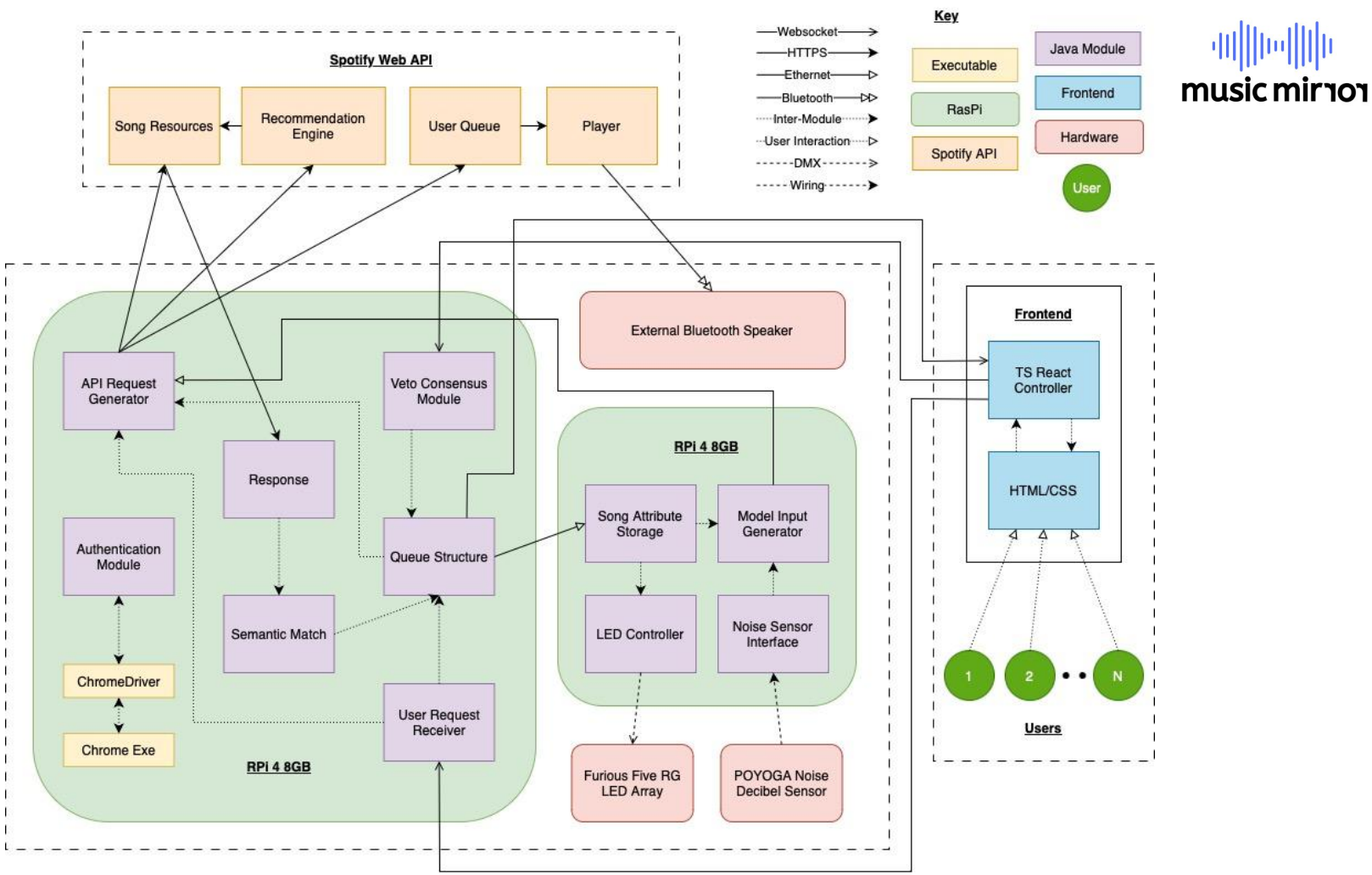
Design Requirements

- User song requests are accurately reflected by the centralized queue within **1 second**
- Easily usable mobile-optimized website
  - Users will be onboarded in under **1** minute on average
- Centralized concurrent queue to accept and maintain ordering of incoming song requests for a target of **100-150 users**
- Consensus voting protocol to support ‘veto’ functionality of songs on queue
- Queue can hold at least **100 songs** (6 hour reception / 3.5 min average song length)

Solution Approach

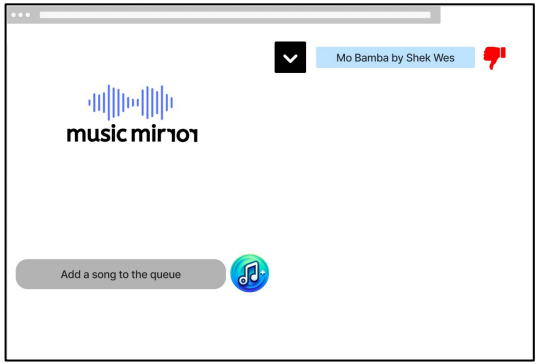
music mirror

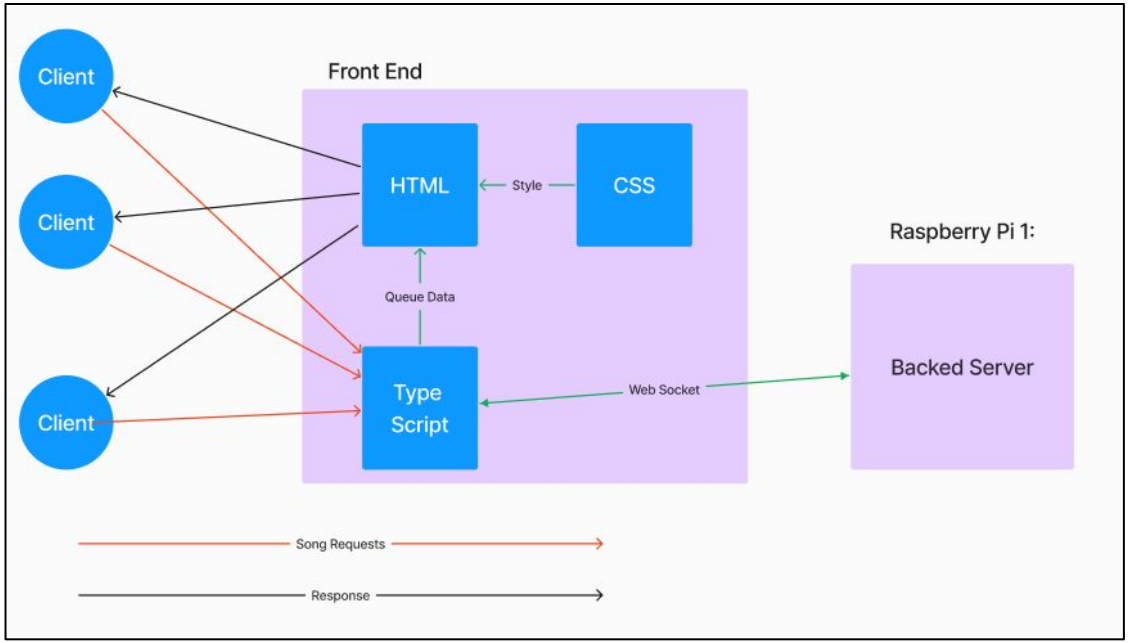



Spotify Web API
Song Resources
Recommendation Engine
User Queue
Player
Key
Websocket
HTTPS
Ethernet
Bluetooth
Inter-Module
User Interaction
DMX
Wiring
Executable
RasPi
Spotify API
Java Module
Frontend
Hardware
User
External Bluetooth Speaker
API Request Generator
Veto Consensus Module
Response
Authentication Module
Queue Structure
Semantic Match
ChromeDriver
Chrome Exe
User Request Receiver
RPi 4 8GB
RPi 4 8GB
Song Attribute Storage
Model Input Generator
LED Controller
Noise Sensor Interface
Furious Five RG LED Array
POYOGA Noise Decibel Sensor
Frontend
TS React Controller
HTML/CSS
1
2
N
Users

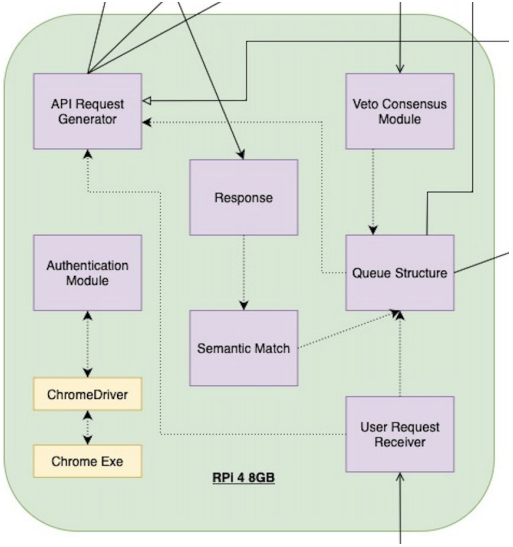
Implementation: Web App

music mirror





Implementation: Main RPi Core



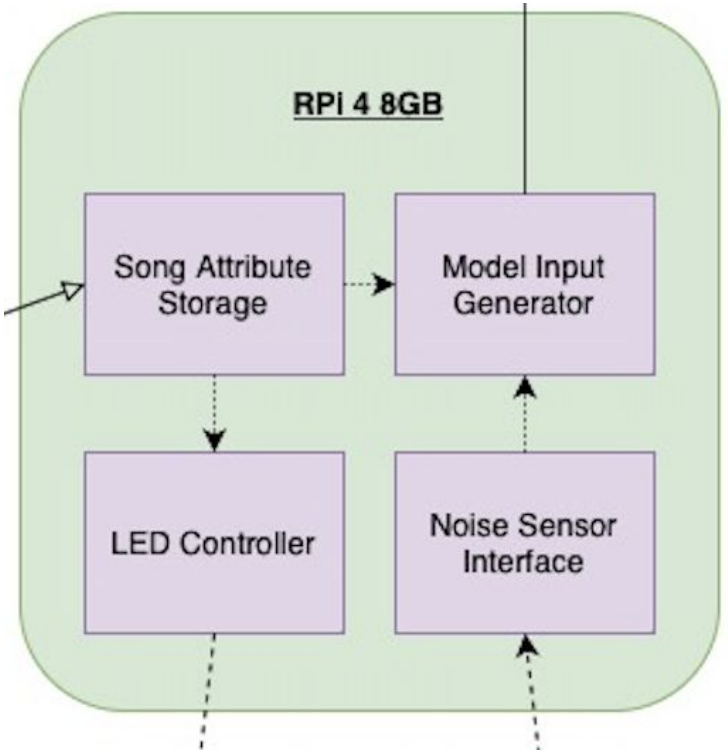
- Accept User requests via onboard WiFi card
- New songs forwarded immediately to Queue Structure, but confirmed on a match response from the Spotify Web API
- Counts song veto votes & prunes vetoed songs from queue
- Next song feeds to Web API for playing through the request generator

```
import com.google.gson.Gson;
import TrackSearchResponse;

public class SpotifySearchRequest {

    public TrackSearchResponse searchForTrack(String access_token,
                                              String song_name, String album_name,
                                              String artist_name,
                                              int encoding_scheme)  {

        URL url = null;
        HttpURLConnection con = null;
        TrackSearchResponse response = null;

        String q = "";
        if (encoding_scheme == 1)   {
            q = this.build_search_query(song_name, album_name, artist_name);
        }
        else if (encoding_scheme == 2)  {
            q = this.build_search_query_2(song_name, album_name, artist_name);
        }
        // first create the URL and open a connection
        try {
            url = new URL("https://api.spotify.com/v1/search?" + q +
                          "&type=track&limit=50");
            con = (HttpURLConnection) url.openConnection();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Implementation: Recommender RPi Core

music mirror

- Store data for each played song:
  - Genre, accousticness, danceability, energy, key, liveness, tempo, etc.
  - Crowd decibel level from noise sensor
- Aggregate these metrics based on user request
- Generate a query to Spotify's recommendation endpoint
- The top recommendation based on our inputs will be queued



Implementation: Physical Interface

music mirror

Speaker

Lighting

Eliminator Furious Five RG 5-in-1 Lighting Effect Fixture Features:

- 4 control modes: DMX, sound activation, auto, primary/secondary for control in a variety of settings

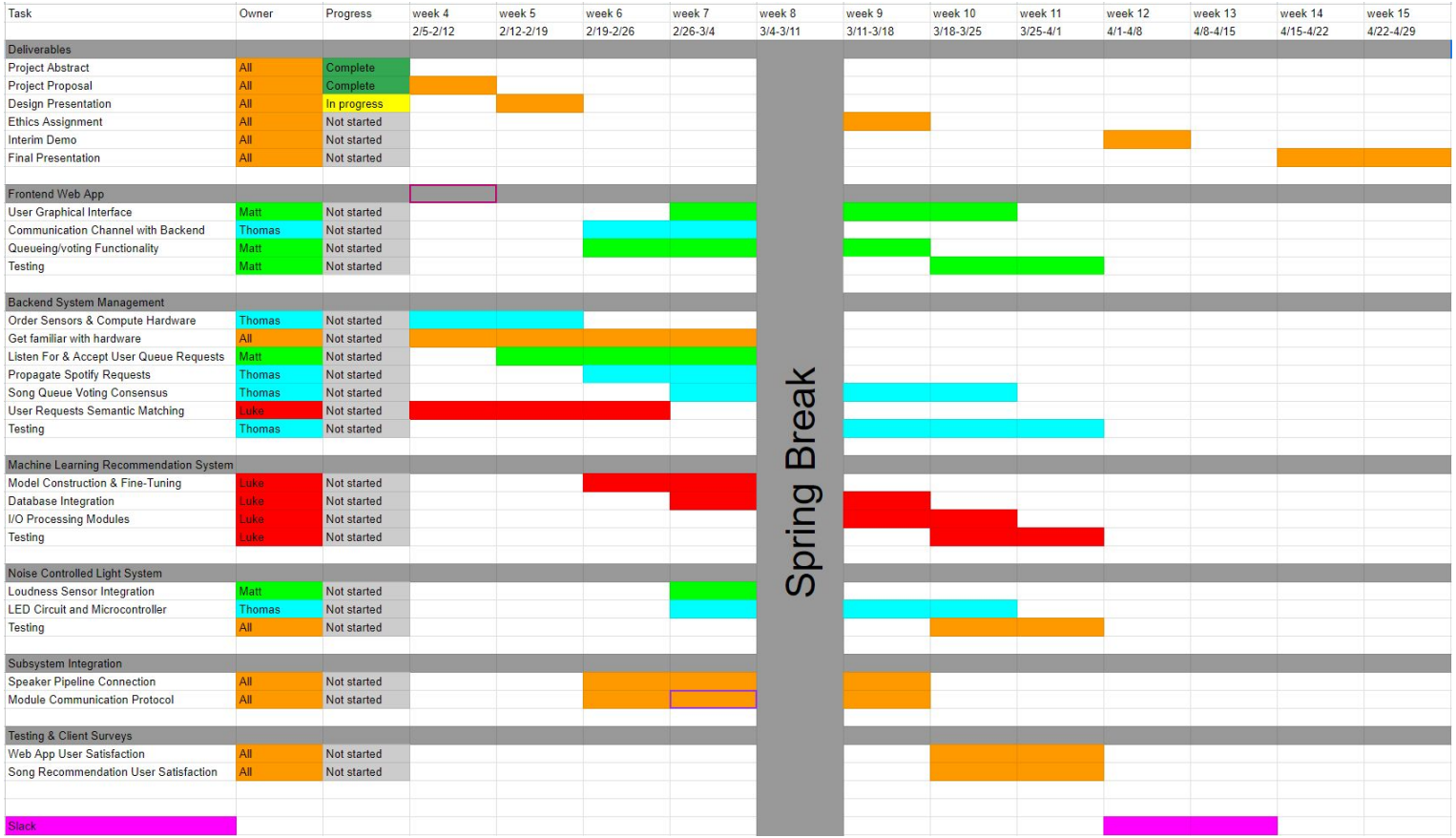
Noise Sensor

POYOGA Noise Decibel Detection Module

Testing, Verification, and Validation

Latency	Web App to System: measure latency for a single time-stamped Play Song request to be reflected on internal queue (< 1 sec)
Capacity	Queue: verify that all Main RPi queue can maintain 100+ songs without running out of memory, and perform operations under max latency User Network: verify that Main RPi can accept ambiguously timed requests from 100-150 concurrently online users
Accuracy	Queue: use test script to issue song requests in a certain order, verify that they appear in that same order on system (and then back on web app) Lighting: use hard coded light script to verify that we can control each light channel independently and to do the intended color & strobing
User Experience	Web App: measure average time to onboard new users, poll on 1-5 scale for ease of use and input responsiveness Recommendations: generate recommendations based on our compound model, poll users on 1-5 scale for quality of recommendations and compare to their ratings for generic Spotify recommendations

Project Management



Task	Owner	Progress	week 4	week 5	week 6	week 7	week 8	week 9	week 10	week 11	week 12	week 13	week 14	week 15
			2/5-2/12	2/12-2/19	2/19-2/26	2/26-3/4	3/4-3/11	3/11-3/18	3/18-3/25	3/25-4/1	4/1-4/8	4/8-4/15	4/15-4/22	4/22-4/29
Deliverables														
Project Abstract	All	Complete												
Project Proposal	All	Complete												
Design Presentation	All	In progress												
Ethics Assignment	All	Not started												
Interim Demo	All	Not started												
Final Presentation	All	Not started												
Frontend Web App														
User Graphical Interface	Matt	Not started												
Communication Channel with Backend	Thomas	Not started												
Queueing/voting Functionality	Matt	Not started												
Testing	Matt	Not started												
Backend System Management														
Order Sensors & Compute Hardware	Thomas	Not started												
Get familiar with hardware	All	Not started												
Listen For & Accept User Queue Requests	Matt	Not started												
Propagate Spotify Requests	Thomas	Not started												
Song Queue Voting Consensus	Thomas	Not started												
User Requests Semantic Matching	Luke	Not started												
Testing	Thomas	Not started												
Machine Learning Recommendation System														
Model Construction & Fine-Tuning	Luke	Not started												
Database Integration	Luke	Not started												
I/O Processing Modules	Luke	Not started												
Testing	Luke	Not started												
Noise Controlled Light System														
Loudness Sensor Integration	Matt	Not started												
LED Circuit and Microcontroller	Thomas	Not started												
Testing	All	Not started												
Subsystem Integration														
Speaker Pipeline Connection	All	Not started												
Module Communication Protocol	All	Not started												
Testing & Client Surveys														
Web App User Satisfaction	All	Not started												
Song Recommendation User Satisfaction	All	Not started												
Slack														