HeyJuke

Web Server: Joe Hirschfeld - hirschjb@mail.uc.edu

Audio Player: John D'Alessandro - dalessjj@mail.uc.edu

Mobile App: Luis Finke - finkelj@mail.uc.edu

Advisor: Dharma Agrawal - agrawadp@ucmail.uc.edu

Goals

HeyJuke is a home jukebox that is easy for end users to set up and operate in a group setting. This will involve:

- Creating a host program on a Raspberry Pi that can run headless
- Creating a paired phone app that is easy and intuitive to use

Intellectual Merit

What HeyJuke offers that other streaming systems do not is the ability to combine media in the same queue from different sources. This is achieved by having queue management on the device instead of relying on streaming services to handle the queue. This is a limitation on other streaming solutions like the Chromecast and the Amazon Fire Stick.

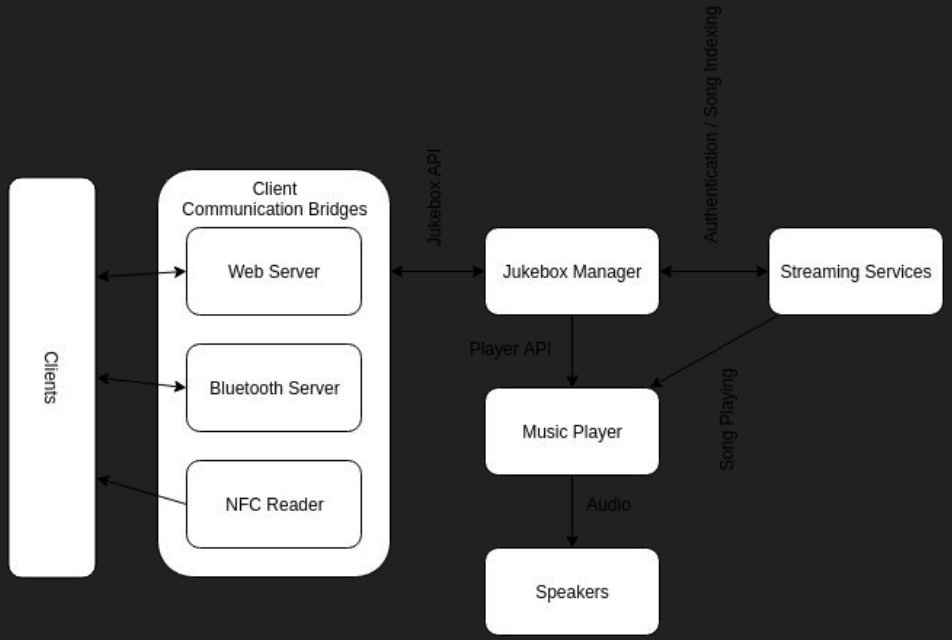
Broader Impact

Having a free open source solution for streaming that can outperform the commodity solutions in ease of use could force the major players in the industry to focus on making their own products more service agnostic, and increasing the usability of all software across the industry.

Design Specifications

A Raspberry Pi runs both the music player and web server. The web server takes in requests from the mobile app and manages the queue. The music player hooks into streaming services to play music through the device's hardware.

The multi-platform mobile app can see the queue and allow users to interact with it.



Technologies

- Our web server is a node server running on a Raspberry Pi 3.
- The mobile app is written in React Native in order to run on any major mobile platform.
- The media player is a headless Electron app that opens a websocket with the web server running on the same device.

Milestones

- Senior Design Expo: Hardware complete and functionality demo
- April 24: MVP Release

Results

- Communicating server and player
- Player can play local files
- Mobile app can search and manage queue

Challenges

- Creating a headless Electron instance in order to use APIs from multiple streaming services
  - Solved by having JS on the Electron page that acts as a websocket client to allow for control from the web server.
- Standardizing the search interface in the app
  - Use generic object wrapper to mold each api including the server’s local file system to the same api
- Learning a new language and framework
  - Two of us are not JS developers. This problem was overcome with pair programming and spending extra time to learn the technologies.