Technology Investigation Summary

& Prototype

Virtual Jukebox Software Project

[v0.1 (Draft 1)]

By:

Austin Bevacqua, Bradley van der Zwan, Dillon Vincent, Ryan Webster, Tanaka Chitete

Table of Contents

[1 Report Overview](#_heading=h.30j0zll) **4**

[1.1 Summary](#_heading=h.1fob9te) 4

[1.2 Project Overview](#_heading=h.3znysh7) 4

[2 Definitions](#_heading=h.2et92p0) **4**

[3 Web Application Frontend Frameworks](#_heading=h.tyjcwt) **4**

[3.1 Application Requirements](#_heading=h.3dy6vkm) 4

[3.2 Frameworks Researched](#_heading=h.1t3h5sf) 4

[3.3 Four Main Frameworks](#_heading=h.4d34og8) 6

[3.3.1 Django](#_heading=h.f4a6ou2cr60z) 6

[3.3.2 React](#_heading=h.2s8eyo1) 6

[3.3.3 Angular](#_heading=h.17dp8vu) 6

[3.3.4 Vue](#_heading=h.3rdcrjn) 7

[3.4 Conclusion](#_heading=h.26in1rg) 7

[4 Web Application Backend Frameworks](#_heading=h.lnxbz9) **7**

[4.1 Application Requirements](#_heading=h.35nkun2) 7

[4.2 Frameworks Researched](#_heading=h.1ksv4uv) 7

[4.3 Four Main Frameworks](#_heading=h.44sinio) 8

[4.3.1 Django](#_heading=h.2jxsxqh) 8

[4.3.2 Ruby on Rails](#_heading=h.z337ya) 9

[4.3.3 ExpressJS](#_heading=h.3j2qqm3) 9

[4.3.4 Spring Boot](#_heading=h.1g974r3zh3qk) 10

[4.4 Conclusion](#_heading=h.1y810tw) 10

[5 Databases](#_heading=h.4i7ojhp) **10**

[5.1 Application Requirements](#_heading=h.2xcytpi) 10

[5.2 Databases Researched](#_heading=h.1ci93xb) 11

[5.3 Three Main Databases](#_heading=h.3whwml4) 11

[SQL vs NoSQL Database Systems](#_heading=h.gfkd38agxtnw) 12

[5.3.1 MySQL vs PostgreSQL](#_heading=h.2bn6wsx) 12

[5.3.2 Redis](#_heading=h.qsh70q) 13

[5.4 Conclusion](#_heading=h.1pxezwc) 13

[6 Messaging Solutions](#_heading=h.49x2ik5) **13**

[6.1 Application Requirements](#_heading=h.2p2csry) 13

[6.2 Pre-made Chat Services](#_heading=h.147n2zr) 13

[6.2.1 TalkJS](#_heading=h.23ckvvd) 13

[6.2.2 SendBird](#_heading=h.ihv636) 14

[6.2.3 ChatEngine](#_heading=h.32hioqz) 14

[6.3 Conclusion](#_heading=h.1hmsyys) 14

[7 Music API](#_heading=h.41mghml) **14**

[7.1 Application Requirements](#_heading=h.2grqrue) 14

[7.2 APIs that were researched](#_heading=h.vx1227) 15

[7.3 Four main SDK’s](#_heading=h.n6yw8994c00q) 16

[7.3.1 Spotify](#_heading=h.1v1yuxt) 16

[7.3.2 Apple Music](#_heading=h.4f1mdlm) 16

[7.3.3 Deezer](#_heading=h.2u6wntf) 16

[7.3.4 Soundcloud](#_heading=h.5owm4x4wekma) 16

[7.4 Legal](#_heading=h.vvq28vqx9v90) 17

[7.4.1 Spotify](#_heading=h.ecxwr7h9pe9r) 17

[7.4.2 Apple Music](#_heading=h.biwlli2o3hf0) 17

[7.4.3 Deezer](#_heading=h.5zo9t8agexzp) 17

[7.4.4 Soundcloud](#_heading=h.4lwuqibvsc9l) 17

[7.4.5 Conclusion](#_heading=h.ew0soy8hq9gb) 18

[7.5 Conclusion](#_heading=h.19c6y18) 18

[8 Best combination](#_heading=h.3tbugp1) **18**

[Appendix](#_heading=h.28h4qwu) **18**

[3.x (Web Application Frontend Frameworks)](#_heading=h.up9qsncupxqm) 18

[4.x (Web Application Backend Frameworks)](#_heading=h.s5m9txu4reyi) 18

[5.x (Database Systems)](#_heading=h.5z2tpzyplhbo) 18

[References](#_heading=h.nmf14n) **21**

[3.x (Web Application Frontend Frameworks)](#_heading=h.oeegomx70vuq) 21

[4.x (Web Application Backend Frameworks)](#_heading=h.72snykgpf6nc) 21

[5.x (Database Systems)](#_heading=h.wjjjh2xyonee) 21

[7.x (Music API)](#_heading=h.zd14o0nbxbrd) 21

# 1 Report Overview

## 1.1 Summary

This report details the research that was conducted on various technologies for the Virtual Jukebox project. This document will summarise research findings, and outline technologies that were found to be suitable for the project

## 1.2 Project Overview

“You are to develop a solution called Virtual Jukebox where music can be played from a single source be it inan indoor setting (e.g., restaurant, café, etc.) or an outdoor setting through portable speakers (e.g., outdoor markets, etc.). The Virtual Jukebox host can create a playlist of songs that is tailored to suit the setting and taste of the host or event and categorising them according to the artist, genre, year, etc. Guests within a certain radius will be allowed to connect to the Jukebox and view the song that is currently playing, add songs(from the playlist) to the queue, and view songs that are currently in the queue. The application will have an algorithm where a particular song cannot be played consecutively (i.e., back-to-back) or only after a certain hop. The application will allow guests to suggest songs to be added to the playlist by the host. Guests can add songs to the queue based on a credit system. Credits can only be earned after fulfilling a certain criterion (e.g.,time-based).“

# 2 Definitions

# 3 Web Application Frontend Frameworks

## 3.1 Application Requirements

The frontend framework must be able to make a web page mobile friendly on multiple mobile platforms. The framework should be easy to learn and adapt, as it will need to be adopted by the group who may or may not have had experience using the framework. This requirement implies that the framework that will be chosen must have a low learning curve, a large ecosystem and a high adoption rate, so that there will be an exhaustive amount of available resources that can be used for development.

The Jukebox application will be playing music over a mobile/desktop, and this means that the framework chosen should be relatively fast as out of sync music playback is not wanted. Performance of the framework should be generally good overall, because the application should have a smooth user experience based on load times.

## 3.2 Frameworks Researched

Ten frameworks researched that could be used in the development of the application. The frameworks were broadly researched in order to see the arguments for and against them, and this information was used to narrow the options to four frameworks that would be researched in depth.

Frameworks that were investigated include (in order): plain JavaScript, React, Angular, Vue, Ember, and Bootstrap. Non-javascript options were also investigated such as .NET, Django, Java (Swing), and Flutter.

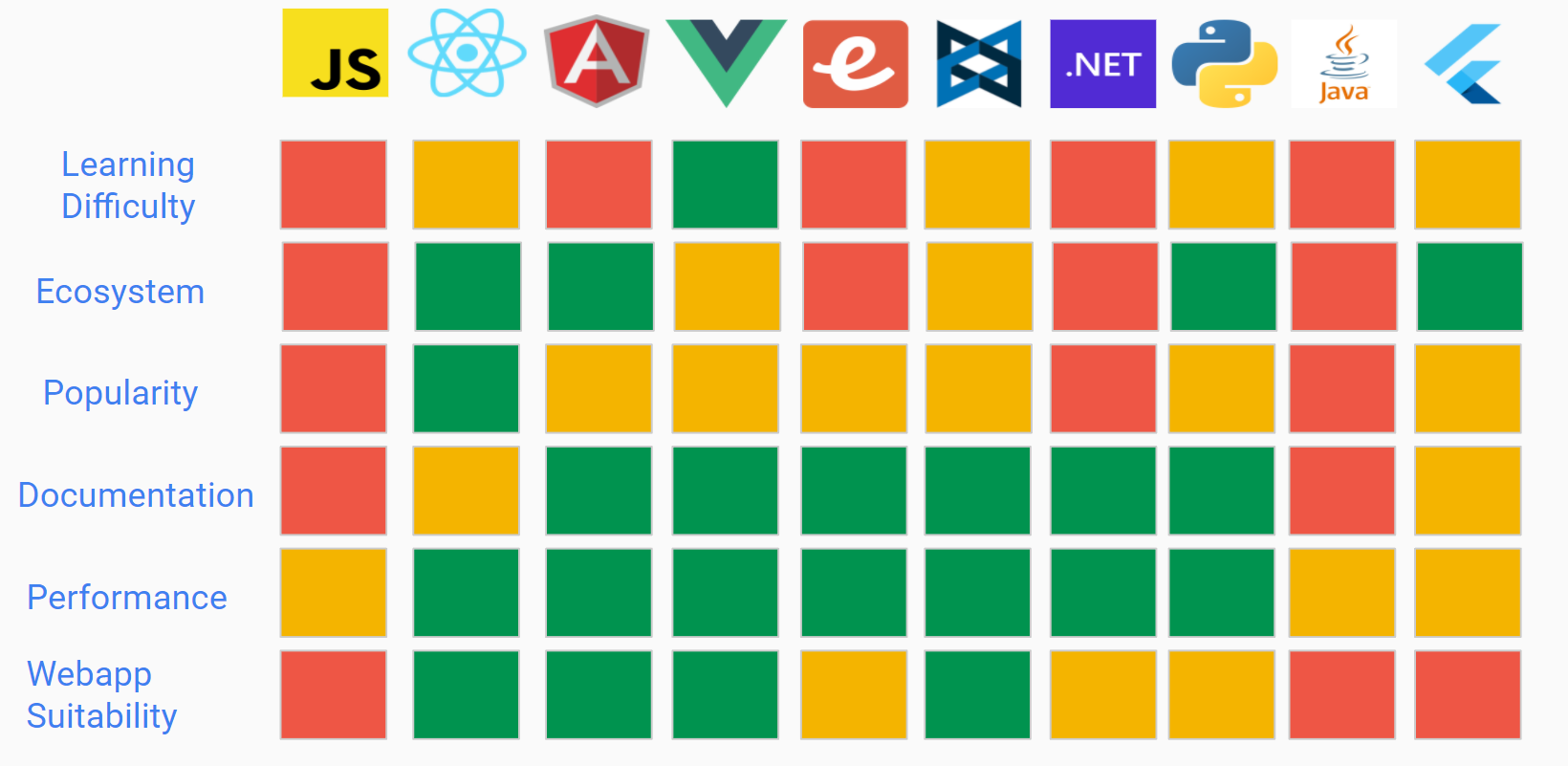


Figure 3.1: A brief comparison of 10 various web technologies

The criteria used for the comparisons are as follows:

Learning Difficulty: How much effort is required to achieve a level of competency with the technology

Ecosystem: How many resources there are for the technology, including forum questions, YouTube videos, tutorials and extensions.

Popularity: How widely used the technology is in 2022. Determined by GitHub stars, and a comparison websites **[ref]**

Documentation: How comprehensive, up to date and exhaustive the official documentation is for the given technology.

Webapp Suitability: How suitable the technology was in regards to creating a web application. Suitability was judged by analysing the technologies core strengths, and if they aligned with the requirements of a web application.

## 3.3 Four Main Frameworks



Figure 3.2: A comparison of the top 4 researched frontend technologies

### 3.3.1 Django

It was found that Django is an inferior choice for frontend development when compared to the JavaScript frameworks React, Angular, and Vue. Django has a huge ecosystem and extensive documentation, but lacks features such as a virtual DOM and two way data binding. Although the ecosystem and community is large, it is generally not recommended to use Django as a frontend tool, but is more suited to being a backend tool due to its excellent REST framework which connects frontend and backends together. Furthermore, it is generally not recommended to use a Django frontend for smaller scale apps, such as the one that will be developed during this project. For these reasons, it is unlikely that Django will be selected as the frontend technology.

### 3.3.2 React

React is a perfectly suitable framework for frontend development. React has many drawbacks, such as its generally out of date documentation, use of JSX, and no two-way data binding, but it makes up for these drawbacks by being relatively easy to use and reliable. React has an enormous community, larger than any existing framework, and is the most popular choice for frontend development. It has features such as the virtual DOM, which improves page responsiveness and makes development easier, and is SEO friendly, so website information will successfully show in search engines. React is seen as fairly easy to learn, and is ideal to create single-page applications with.

### 3.3.3 Angular

Angular was found to be a perfectly suitable framework for frontend development. Angular is the most feature-rich frontend, boasting features such as two-way data binding, the use of the MVC pattern, being SEO friendly and NgRX for state management. The main drawback of Angular is its complexity. Angular is seen as extraordinarily hard to learn, especially compared to React and Vue. Due to this complexity, Angular is seen as better suited to enterprise-level applications. Despite this, Angular is still deemed a perfectly suitable frontend framework due to its immense feature set, and the use of TypeScript compared to JavaScript.

### 3.3.4 Vue

It was found that Vue is a perfectly suitable framework for frontend development. Vue is not as feature rich or as popular as Angular or React, but its main strength is its simplicity. Vue is considered the easiest framework to learn and get started with, boasting that you can have an app started “in minutes” **[ref]**. Vue also has a decent amount of features, such as a virtual DOM, two-way data binding and VueX, which manages component states. Vue’s main drawbacks come from its simplicity and lack of community. Vue has a relatively small ecosystem, making it harder to find help for Vue specific issues. Vue’s simplicity is seen generally as a downside due to it being more work than necessary to implement complex reactivity, and the flexibility of the framework usually leads to more errors than its React and Angular counterparts. Vue is generally seen as a perfect framework for simple, small scale apps, and for this reason, and the reasons above, Vue can be considered as a suitable frontend framework

## 3.4 Conclusion

In conclusion, Vue, Angular and React were found to be perfectly suitable frontend frameworks for this project.

# 4 Web Application Backend Frameworks

## 4.1 Application Requirements

Communication between client and server will be required for implementing features such as a live chat room, and voting for songs. The WebSocket API can be used to establish bidirectional communication between a browser and a server, which allows users to send a message to a server and receive responses to events rather than having to poll for a response. For example, websockets can be used to open a connection between a user and a server, and the same websocket connection is reused when sending and receiving messages or responses to events. The WebSocket protocol is fully supported by all modern web browsers and is not constricted to any specific backend technology, but different backend technologies have different implementations for handling events, and have access to specific libraries. The backend technology selected should also be able to connect to a database management system.

## 4.2 Frameworks Researched

Eight backend frameworks were researched that could be used in the development of the application. The frameworks were broadly researched in order to see the arguments for and against them, and this information was used to narrow the options to three frameworks that would be researched in depth.

Frameworks that were investigated include (in order): ASP.NET, Django, Ruby on Rails, Flash, ExpressJS, CakePHP, Laravel, Spring Boot

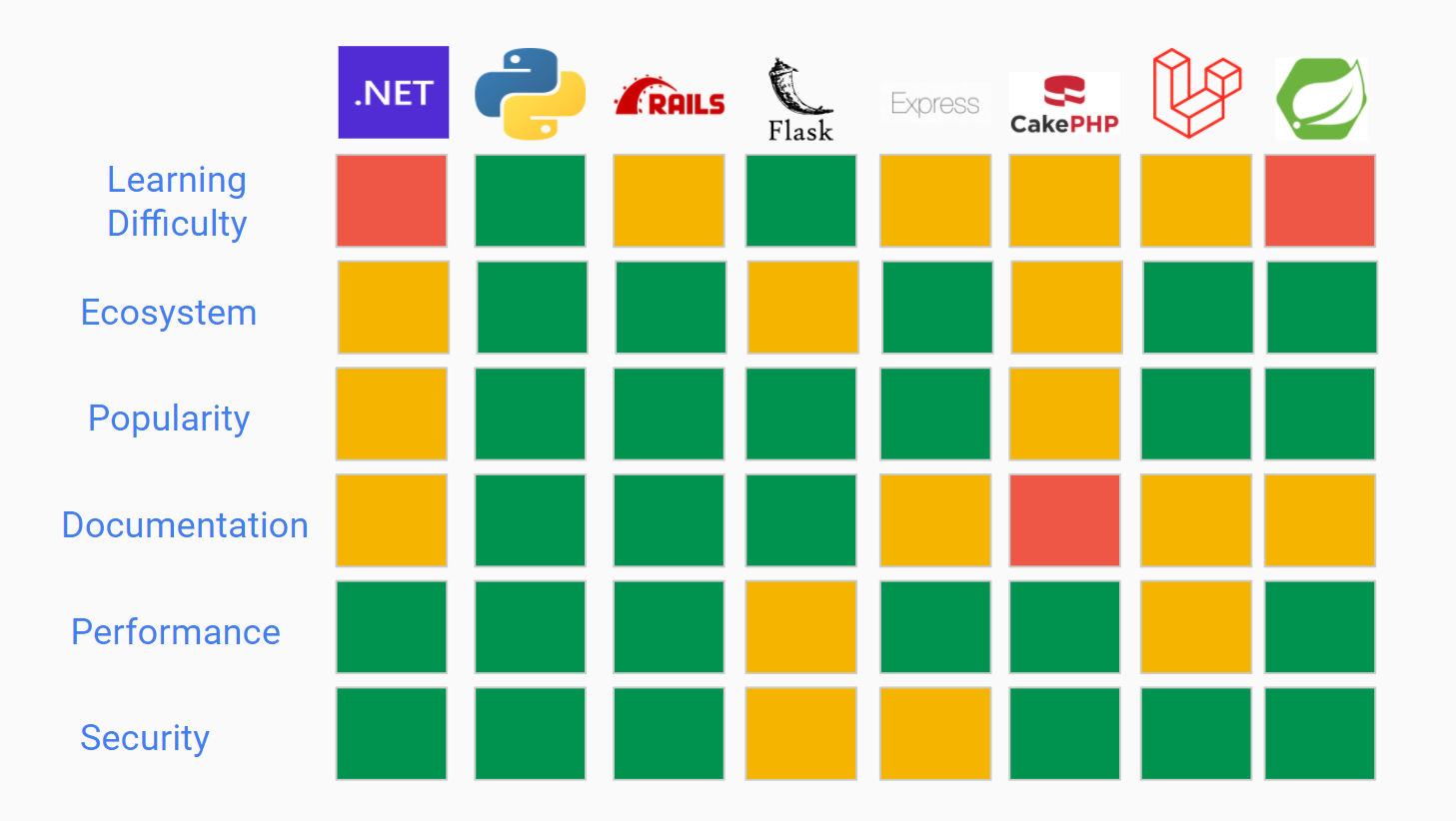


Figure 4.1: A brief comparison of 8 various backend web technologies

## 4.3 Four Main Frameworks

### 4.3.1 Django

Pros:

* Easy to learn
* Has a large amount of resources available
* Has a huge community
* Easy to import packages and modules. Extremely large amount of packages
* Provides a high level of security. Includes protection against SQL injection and cross-site scripting.
* Excellent documentation
* Supports SQLite, PostgreSQL, MySQL, and Oracle.

Cons:

* Not seen as very appropriate for smaller projects

Unique Features:

* Uses Python
* Used for frontend and backend development
* Django comes with a lot of features; Admin site, ORM, Authentication, Cashing
* Uses the REST framework to connect frontend to backend
* MVT (Model - View - Template) Layout
* Can be used for full-stack development (frontend and backend)

Django provides its own support of handling web protocols other than HTTP. Django channels extends the capabilities of Django by allowing projects using Django to utilise long-running connections such as WebSockets. The JavaScript websocket API can be used on the client-side of a connection to initiate a connection, and Django channels is used on the server-side to receive and send requests back to the client.

### 4.3.2 Ruby on Rails

Pros:

* Great performance
* Great documentation
* PostgreSQL and MySQL compatibility
* Quicker to create a MVP (minimum viable product)
* Flexible implementation
* Easy to modify and migrate
* High scalability
* Built in security against a range of attacks

Cons:

* Small web development community when compared to Django
* Difficult to learn if no Ruby experience

Unique Features:

* MVC Template
* Flexible templates
* “Don’t repeat yourself” philosophy
* Built in security features

### 4.3.3 ExpressJS

Pros:

* Extremely large community. Large ecosystem.
* Easily scalable
* Extremely lightweight
* Does not complicate NodeJS any further
* Middleware is built to support catching errors
* “Rapid backend development time”
* Supplies debugger
* Has support for any database.

Cons:

* Modules are frequently used. These modules often have poor documentation
* Decent learning curve to learn NodeJS
* No standardised structure

Unique Features:

* MVC Architecture
* Debugger
* “Express Middleware”
* Offers a thin sheet of essential features.
* Does not complicate things
* Socket.io Library

Node.js has access to the Socket.IO library which is used in real-time web applications which utilises websocket connections and offers HTTP long polling as a fallback when websocket connections cannot be established. Socket.IO is only available when using a node backend, and provides many features which are often manually implemented in projects utilising the websocket API. These features include the ability to broadcast events to all connected clients in specific rooms, automatic reconnection, and automatically removing users from all connected sockets and channels they were a part of upon disconnection. Socket.IO is also compatible with TypeScript.

### 4.3.4 Spring Boot

Pros:

* Lightweight
* In-built security features. Industry standard security by default
* Large ecosystem, plenty of resources online
* Flexible. Can use XML or Java based annotations
* Dependency injection makes testing easier.
* Auto configuration can help get a MVP completed quickly
* Our team all know Java
* Great at concurrency and multithreading. Great scalability

Cons:

* Extremely high learning curve
* More difficult to use than other frameworks
* Mediocre documentation

Unique Features:

* In built security features
* “Auto-configuration”
* Kotlin support

## 4.4 Conclusion

In conclusion, Django, Ruby on Rails and Express were all found to be suitable backend frameworks for the Virtual Jukebox project.

# 5 Databases

## 5.1 Application Requirements

By analysis of the project brief, the Virtual Jukebox web-application will *likely require* the following database-related features:

* Confidentiality in the storage of user account details (Encryption, Authentication).
* Consistency and integrity in database operations and storage of user data (ACID transaction guarantees).
* Web-server compatibility.
* Ease of scalability, in the case that the application needs to upscale to store more types of data, and support more users.
* Efficiency for processing large volumes of data in short timeframes.

The application *may* benefit from these database features:

* Multi-modal data type support such as geospatial data, for storing jukebox location data.
* NoSQL functionality for addition of data that does not fit into a standard tabular RDBMS format (assists in future scalability).

Redundant or irrelevant database features for this use-case include:

* Data warehousing, as data analytics is not relevant for any Virtual-Jukebox function.
* Feature heavy systems (e.g. Block chain support, AI functionality), due to the lack of niche or complex functionality of the program.

The following outlines the potential storage requirements for the Virtual Jukebox application. Note that these may change after the development of a software requirements specification:

* User accounts:
  + Login Credentials (Username, Passwords).
  + User-type (Host or Guest).
* Saved Host playlists:
  + Playlist title
  + List of songs, likely represented by IDs related to the music API.
* Jukebox Data:
  + Guests and Host connected per session
  + Chat logs per session
  + Songs in current session’s playlist
  + Songs in current session’s queue
  + Connection code and QR code
  + Jukebox location data:
    - Geographical location of jukeboxes for guests to find/locate (likely latitude and longitude or similar)
    - Max connection range (in metres)
  + Token count per guest. Update based on token acquisition/spending.

## 5.2 Databases Researched

A total of eight databases were initially considered with the database requirements kept in mind. These systems included Oracle, MySQL, MS SQL Server, PostgreSQL, MongoDB, IBM DB2, Redis, and SQLite. See appendix 5.A for a comparison between each of these systems, showcasing their pros, cons, and ideal use-cases.

## 5.3 Three Main Databases

Three database systems which seem to suit the requirements of the Virtual Jukebox application include *MySQL*, *Redis*, and *PostgreSQL*. As core query functionality and security is desirable, and the scope of the application is not substantial or focused around large-scale analytics, simpler databases are preferred over complex, feature-rich, proprietary ones. Oracle, MS SQL Server, and IBM DB2 were ruled out due to their difficult learning curve and their overwhelming feature-set [5.1]-[5.4]. SQLite was ruled out due to its lack of features, hindering any form of expandability, as well as its embedded nature which is not relevant in a web-server setting [5.5]. Relational databases seem more suitable in this scenario, as little to no unstructured data will be handled, therefore MongoDB was ruled out. Redis is a consideration however, due to its ability to be used alongside other database systems as a caching tool to improve performance to frequently accessed data, even though it is a NoSQL system [5.3].

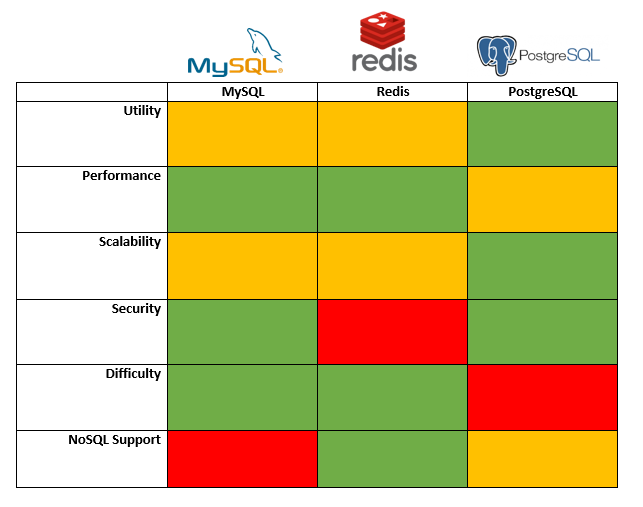
Redis could be used alongside a traditional RDBMS, simply for caching frequently accessed data such as chat-messages, session user-count, guest-credit count, and the songs in queue. PostgreSQL is another consideration as although it is primarily an RDBMS, it offers support for JSON objects due to its object-relational nature. This would be helpful in the future, if the program would ever need to store unstructured data, without the need for a NoSQL implementation, while still providing all the RDBMS features including query access. MySQL is another option to consider due to its simplicity in implementation, with enough features to suit the Virtual Jukebox’s data storage requirements. Each of these three systems are open-source, allowing for commercial usage without fees if needed [5.4]. They are also all relatively low in-terms of resource consumption, except for Redis which utilises random access memory more than the other two to obtain caching capabilities [5.4]. 

Figure 5.1: Brief overview of top three databases compared by core aspects.   
(See appendix 5.B for detailed comparisons in each of these aspects.)

## SQL vs NoSQL Database Systems

An RDBMS is preferred as most if not all data to be stored will follow a specific format, hence will be structured. Structured data is suitable for queries, which would be useful for application functionality such as returning all active jukebox locations, instead of having to extract unstructured data, then search for specific keywords manually. Although unstructured data support would support increased scalability (by not needing to comply with a defined schema), it is not required for the current specification of the software. It is always possible to introduce a NoSQL solution later on for any unstructured data storage requirement, whereas starting with a NoSQL system such as MongoDB would restrict or minimise the ability to query data initially. Additionally, many SQL systems contain support for NoSQL functionality such as JSON data storage in both MySQL and PostgreSQL, which could be used in future scalability if needed [5.6].

### 5.3.1 MySQL vs PostgreSQL

For the RDBMS selection between MySQL and PostgreSQL, although MySQL would be sufficient and is generally considered easier to use according to [5.5], PostgreSQL is preferred due to a variety of reasons. Firstly, PostgreSQL is completely open-source, therefore all its features are available for any form of usage for free, whereas MySQL has fees for increased functionality. PostgreSQL is also more scalable than MySQL as it does not possess a database size limit, and PostgreSQL natively supports Multi-version concurrency control enabling more efficient concurrent operations [5.6]. Concurrent operation support would be useful when multiple people are requesting songs and/or changing song priorities. Additionally, PostgreSQL natively complies with ACID (Atomicity, Consistency, Isolation, and Durability) transaction properties (whereas MySQL does not natively), enhancing its reliability [5.6]. Lastly, PostgreSQL supports a range of data encryption algorithms such as AES and 3DES, as well as providing varied NoSQL support in key-value storage such as JSON indexing, and XML support, enabling storage using both dynamic and static schemas, whereas MySQL simply has JSON storage support [5.11]. This may be useful when upscaling the Virtual Jukebox to include additional features in the future which may require unstructured data storage.

### 5.3.2 Redis

As an RDBMS is preferred for this software, and persistence is required, Redis is ruled out as a standalone/primary database system, however, it is likely to still be used as a caching system alongside a traditional non-volatile SQL RDBMS to enhance performance, which is noted by [5.10] to be a common design pattern. As Redis will be used, disk I/O will be reduced when reading frequently used database contents due to its on-memory nature [5.10]. It is likely that contents that do not need to be frequently accessed or do not need real-time access with the Virtual Jukebox will not be stored in-memory using Redis, but will instead solely be kept in the primary RDBMS. Other contents that would require frequent access however, such as playlist contents, and queue contents/song priorities will likely be stored in the Redis on-memory store, with scheduled writes to the disk for backups/persistence. Redis also natively supports data structures such as sorted sets/lists which would be helpful in tracking queue song orders, and geospatial indexes which would be useful in storing active Jukebox locations [5.9].

## 5.4 Conclusion

Despite PostgreSQL being more suitable in the specified areas, each of these three database systems will be prototyped with simple databases, testing basic functionality to gain a more thorough understanding, and to make a more informed final decision of which system(s) will be used in development. As the difficulties vary between MySQL and PostgreSQL, the prototypes will assist in identifying the feasibility in implementing PostgreSQL versus MySQL.

# 6 Messaging Solutions

## 6.1 Application Requirements

The web-application should provide a live chat-room for each jukebox session where users can communicate through instant messages in a group setting, with chat messages being programmatically moderated for offensive language.

## 6.2 Pre-made Chat Services

### 6.2.1 TalkJS

TalkJS is a pre-built chat API offering all the features that will be required in the application as well as many more. Features that are included in all subscription tiers but are unneeded for the virtual jukebox project include email/SMS notifications, replying via email, file transfers, and live location sharing.

Pricing begins at a minimum of $279 USD per month, with this increasing to $569 per month if a chatroom requires more than 30 users at a time.

### 6.2.2 SendBird

SendBird offers the creation of real-time communities with a live chat functionality. Features include sending live GIFs, audio, images, typing indicators, profanity filters, spam protection, and moderation tools for specific users.

SendBird offers a free trial version under a developer subscription, with up to 100 monthly active users, and up to 10 concurrent connections at a time. The cheapest subscription tier begins at $399 USD per month and allows for up to 2,000 concurrent users, and 100,000 maximum users per month.

### 6.2.3 ChatEngine

ChatEngine is a chat API used in the creation of instant chat messaging systems which can be used with a React.js front-end. ChatEngine aims to provide a simple service with a developer friendly API and much lower pricing than other readily available services. ChatEngine provides the implementation of React components and hooks for use in assembling the user interface and managing state, as well as handling the implementation of a server that can be used to store messages and user accounts.

The cheapest pricing plan begins at $21 USD per month per 3000 users and 50,000 messages. Features include Secure Sockets Layer (SSL) encryption and storing chat logs for 30 days.

## 6.3 Conclusion

ChatEngine is the most suitable option out of the pre-made solutions as it offers all of the features that are required for the application without any extra unnecessary features, however the provided interface components are only available when using a React.js frontend. Due to SendBird and TalkJS charging much higher prices which increase as the scale of the application increases, these may be deemed unsuitable as they also offer many features which are unneeded in the current specifications. If none of these options are deemed suitable, a chatting system can be manually implemented utilising the websocket API.

# 7 Music API

## 7.1 Application Requirements

All requirements of the Playback Controller and API (“elements”) of any candidate streaming Software Development Kit (SDK) mentioned hereafter are mentioned with consideration of the Project Brief (“Virtual Jukebox – Project Brief”)

Virtual Jukebox will, and will likely require the Playback Controller of any candidate SDK to provide the functionality to:

* Play and pause track playback
* Seek to a given timestamp
* Skip to the next track
* Skip to the previous track
* Adjust volume
* View playback history
* View the songs in the queue
* Append a song to the queue

In addition, Virtual Jukebox will, and will likely require the API of any candidate SDK to provide the functionality to:

* Create a playlist in the User’s library
* Retrieve a playlist from the User’s library
* View the songs in a playlist from the User’s library
* Search for a song in a playlist from the User’s library
* Search for a song to add to a playlist from the User’s library
* Add a song to a playlist from the User’s library
* Remove a song from a playlist from the User’s library
* Sort the songs in a playlist from the User’s library by artist
* Sort the songs in a playlist from the User’s library by genre
* Sort the songs in a playlist from the User’s library by year

As an addition to the requirements of the API, it is also required that the API provisions a large enough service call bandwidth suitable for the use of Virtual Jukebox in “an indoor setting (e.g., restaurant, café, etc.) or an outdoor setting through portable speakers (e.g., outdoor markets, etc.)” [7.1].

## 7.2 APIs that were researched

The candidate streaming SDKs for the implementation of Virtual Jukebox are as follows:

* Spotify Web Playback SDK (Spotify)
* Apple MusicKit JS (Apple Music)
* Deezer JavaScript SDK (Deezer)
* SoundCloud JavaScript SDK (SoundCloud)

|  | **Spotify** | **Apple Music** | **Deezer** | **SoundCloud** |
| --- | --- | --- | --- | --- |
| Playback Controller | | | |
| Start/resume playback |  |  |  |  |
| Seek |  |  |  |  |
| Skip to next track |  |  |  |  |
| Skip to previous track |  |  |  |  |
| View history |  |  |  |  |
| Queue song (prepend) |  |  |  |  |
| Queue song (append) |  |  |  |  |
| View queue |  |  |  |  |
| Adjust volume |  |  |  |  |
|  | API | | | |
| Create playlist |  |  |  |  |
| Get playlist |  |  |  |  |
| View playlist |  |  |  |  |
| Search for a song |  |  |  |  |
| Add a song to a playlist |  |  |  |  |
| Remove a song from a playlist |  |  |  |  |
| Sort playlist by artist |  |  |  |  |
| Sort playlist by genre |  |  |  |  |

Figure 7.1: A comparison between streaming SDK’s

## 7.3 Four main SDK’s

### 7.3.1 Spotify

“The Spotify Web Playback SDK is a public JavaScript SDK that allows you to implement local streaming playback of Spotify content in their web applications” [7.2].

### 7.3.2 Apple Music

“MusicKit JS lets users play songs from Apple Music and their Cloud Library inside your JavaScript app. When a user provides permission to access their Apple Music account, your JavaScript app can create playlists, add songs to their library, and play any of the millions of songs in the Apple Music catalogue directly in a browser, with no additional plugins” [7.3].

### 7.3.3 Deezer

“The JavaScript SDK allows you to stream music, authenticate users and lets you access all the Deezer API endpoints. [7.4]”.

### 7.3.4 Soundcloud

“The JavaScript SDK lets you easily integrate SoundCloud into your website or webapp. [7.5]”.

## 7.4 Legal

It is to be noted that none of the SDKs allow for non-personal use and, in turn, commercialisation.

### 7.4.1 Spotify

“As laid out in our Terms and Conditions, Spotify is for personal, non-commercial use.

This means that you cannot broadcast or play Spotify publicly from a business, such as bars, restaurants, schools, stores, salons, dance studios, radio stations, etc.

To play in a commercial environment, check out our friends at Soundtrack Your Brand.”

### 7.4.2 Apple Music

“You agree not to require payment for or indirectly monetize access to the Apple Music service (e.g. in-app purchase, advertising, requesting user info) through Your use of the MusicKit APIs, MusicKit JS, or otherwise in any way…”

### 7.4.3 Deezer

“The Developer agrees that the use of the Services is strictly limited for a non-commercial purpose and in a non-commercial environment.

It means that the Developer shall not perceive, receive, generate, benefit or create directly or indirectly, any moneys, incomes, revenues, data or any other consideration in connection with the use of neither the Services themselves, nor any and all Content accessed through the Services.

…

The Developer undertakes to inform by any means any person accessing the Content through the Webpages and/or Personal Applications of its conditions of use and notably that the streaming of the recordings is limited to a strictly private use within a family scope.

The use of the Content is limited to a strictly private use within a family scope.”

### 7.4.4 Soundcloud

“Only the following commercial uses are acceptable:

* the sale of an app, the primary purpose of which is the creation of User Content and the sharing of that content by the creator in accordance with the User Terms;
* the use of the SoundCloud® API to deliver User Content to the Uploader's own ad-enabled website, blog or social media profile;
* services aimed at helping a user to promote his/her User Content via authenticated access to the user's account, provided this does not constitute a breach of the User Terms; and
* other cases that SoundCloud may approve on a case by case basis.

…

The following commercial uses are not acceptable (this is not an exhaustive list):

* any in-app purchases which allow access to content or features already available via the SoundCloud® platform;
* any app that features any advertising, sponsorship or promotion around User Content or enables User Content (other than a user's own User Content) or SoundCloud® functionality to be embedded into the third party commercial services. The only exception to this would be an app that allows an Uploader to connect his/her SoundCloud® account in order to make his/her own User Content available in a commercial context (for example, an app that allows an Uploader to deliver his/her User Content to an online retail store, or to his/her own ad-supported website, blog or social media profile).”

Given the documentation mentioned here, it is recommended that a disclaimer be issued upon start-up of Virtual Jukebox, mentioning that usage be limited to personal use only (e.g., family gatherings, parties, etc).

### 7.4.5 Conclusion

Given the documentation mentioned here, it is recommended that a disclaimer be issued upon start-up of Virtual Jukebox, mentioning that usage be limited to personal use only (e.g., family gatherings, parties, etc).

## 7.5 Conclusion

In conclusion…

# 8 Best combination

Talk about what would be some good combinations of frameworks/database/API’s that might work for the Jukebox application.

Vue/React/Angular +   
Django/Express/Ruby +   
PostgreSQL (with redis for caching) +   
Spotify api/web sdk

# Appendix

### 3.x (Web Application Frontend Frameworks)

### 4.x (Web Application Backend Frameworks)

### 5.x (Database Systems)

(5.A) Comparison table between eight database systems  
 Points derived and summarised from [5.1]-[5.5]:

| Database | Pros | Cons | Ideal Use-case |
| --- | --- | --- | --- |
| Oracle | - Highly scalable  - Multimodal data-types | - Expensive commercial licensing  -Resource intensive | - Very large databases  - Data warehousing |
| MySQL | - Native security  - Prioritising speed over features | -Paid license for enterprise edition  - Difficult to upscale  - Not natively ACID compliant | - For databases focusing on security and speed over features  - Small/Medium web-solutions |
| MS SQL Server | - Scalable cloud support  - Dynamic Data masking  - Multi-modal data support | - Expensive commercial licensing  - Resource intensive | - For Microsoft environments  - To shift databases to the cloud |
| PostgreSQL | - Open source, no commercial fees  - Highly scalable, with many interfaces | - Difficult to configure advanced features  - Lacking speed in read-only operations | - For high frequency operations, and distributed databases  - Scalability over speed |
| MongoDB | - Very fast operations  - No set structure required  - Open source, no commercial fees. | - No support for relational storage  - Does not use SQL  - Requires extensive setup | - For unstructured document storage  -When data does not require relationships |
| IBM DB2 | - Native AI support  - Cloud, server, physical support  - Multi-tasking support | - Very expensive fees for all  - Database clusters require 3rd party software | - For large data-heavy databases requiring AI processing  - Data warehousing |
| Redis | - in-memory caching support  - Very fast operations  - Open source, no fees | - Little relational storage support  - Restricted to commands only (no query lang.)  - High RAM usage | - When data does not require relationships  - When data will be frequently accessed |
| SQLite | - Simple to set up.  - Reliable for low-traffic websites | - Limited throughput  - Limited scalability  - Server-less | - For low traffic web-applications  - For local or personal applications |

Difficulty and Licensing derived from [5.4] and [5.5]:

| Database | Difficulty | License |
| --- | --- | --- |
| Oracle | Hard learning curve. | Proprietary |
| MySQL | Mild learning curve | Open source, but under GNU: has paid editions for more features. |
| MS SQL Server | Hard learning curve. | Proprietary |
| PostgreSQL | Hard learning curve. | Open-Source |
| MongoDB | Mild learning curve | Open-Source |
| IBM DB2 | Hard learning curve. | Proprietary |
| Redis | Mild learning curve | Open-Source |
| SQLite | Small learning curve | Open-Source |

(5.B) Three selected database system’s detailed comparison table  
 Points derived and summarised from [5.4]-[5.10]:

| Database | Utility | Performance | Scalability |
| --- | --- | --- | --- |
| MySQL | - SQL RDBMS  - Aims to be fast and reliable with a limited feature-set  - Easy to learn  - Base edition is free. Other editions are paid | - Faster in read-only operations (compared to other SQL DBMS)  - Memory Storage Engine provides frequently used table support  - Query cache for frequently used queries | - Scales vertically (Requires more powerful hardware to upscale)  - Less features to support large-scale operations |
| Redis | - NoSQL in-memory database/caching system  - Volatile (data stored in RAM) with options to enable non-volatility  - Key-value storage architecture  - Native data structures including strings, hashes, lists, sets, bitmaps, geospatial indexes, streams, etc.  - Open-source (BSD License) | - High memory (RAM) usage as it is an in-memory cache.  - Low disk usage if used solely as a volatile cache.  - In-memory cache provides very quick  read/write operations (no disk I/O needed except for non-volatility)  - Very fast reads with keys due to its key-value storage structure - O(1) via hashing. | - Larger databases will require more RAM (Virtual memory not supported).  - Unstructured data support enables ease of addition of new data types/structures.  - Horizontally scalable (more machines and/or more RAM to upscale). |
| PostgreSQL | - SQL RDBMS with object support such as JSON  - Aims to provide large-scale performance and scalability with a larger feature-set  - Integration of functions made in programming languages (Java, C, etc.)  - Different user roles/privileges  - Completely Open-source | **-** Faster with complicated queries on large data sets.  - Faster with read-write operations on large data sets with concurrent support (compared to other SQL DBMS)  - Table Partitioning can enable increased performance to frequently accessed parts of a table | - Scales vertically (Requires more powerful hardware to upscale)  - No restriction on database size  - Multi-Version Concurrency Control to support many concurrent users (via parallel queries). |

| Database | Security | Difficulty | NoSQL Support |
| --- | --- | --- | --- |
| MySQL | - TLS encryption support | - Not case sensitive  - Considered easy to set-up and use  - Well-documented | - Later releases provide some NoSQL capabilities including JSON data management/storage. |
| Redis | - Potential data loss from storage of volatile in-memory data.  - Designed for use in trusted environments  - TLS encryption support | - Considered easy/medium to set-up and use  - Mild learning curve compared to other NoSQL and SQL systems  - Commands are case-insensitive, but keys are case sensitive | - Fully NoSQL without native SQL support.  - Unstructured data support |
| PostgreSQL | - Data encryption algorithms supported: AES, 3DES  - Native SSL encryption support for connections | - Case Sensitive  - Generally considered harder to set-up and use | - Some NoSQL support allows for the storage and management of unstructured data (JSON/key-value). |

# References

### 3.x (Web Application Frontend Frameworks)

### 4.x (Web Application Backend Frameworks)

### 5.x (Database Systems)

[5.1]C. Arsenault. “The Pros and Cons of 8 Popular Databases.” Keycdn.com. https://www.keycdn.com/blog/popular-databases (accessed Mar. 23, 2022).

[5.2]Idslogic. “The Pros and Cons of Some of the Most Popular Databases.” Idslogic.com. https://www.idslogic.com/blog/2019/09/the-pros-and-cons-of-some-of-the-most-popular-databases.html (accessed Mar. 23, 2022).

[5.3] M. Kamaruzzaman. “Top 10 Databases to Use in 2021.” TowardsDataScience.com. https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba (accessed Mar. 23, 2022).

[5.4] Altexsoft. “Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB. Elasticsearch, and others.” Altexsoft.com. https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/ (accessed Mar. 23, 2022).

[5.5] A. Duggal. “SQLite vs MySQL: 5 Critical Differences.” Hevodata.com. https://hevodata.com/learn/sqlite-vs-mysql/ (accessed Mar. 23, 2022).

[5.6] A. Phajdar. “PostgreSQL vs MySQL: 8 Critical Differences.” Hevodata.com. https://hevodata.com/learn/postgresql-vs-mysql/ (accessed Mar. 23, 2022).

[5.7] S. Kakarla. “PostgreSQL vs MySQL.” Sumologic.com. https://www.sumologic.com/blog/postgresql-vs-mysql/ (accessed Mar. 23, 2022).

[5.8] Redis. “Redis Security.” Redis.io. https://redis.io/docs/manual/security/ (accessed Mar. 27, 2022).

[5.9] Redis. “Data Structures.” Redis.com. https://redis.com/redis-enterprise/data-structures/ (accessed Mar. 27, 2022).

[5.10] Redis. “Redis FAQ.” Redis.io. https://redis.io/docs/getting-started/faq/ (accessed Mar. 27, 2022).

[5.11] 2ndQuadrant. “PostgreSQL vs MySQL.” 2ndQuadrant.com. https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/   
(accessed Mar. 29, 2022).

### 7.x (Music API)

[7.1] Amristar Solutions Pty Ltd, Virtual Jukebox – Project Brief. Perth, Western Australia. 2022.

[7.2] Spotify. Web Playback SDK Reference, Spotify. Accessed on: Mar. 30, 2022. Available: https://developer.spotify.com/documentation/web-playback-sdk/reference/

[7.3] Apple. MusicKit JS: Add an Apple Music player to your web app, Apple. Accessed on: Mar. 30, 2022. Available: https://developer.apple.com/documentation/musickitjs

[7.4] Deezer. Javascript SDK, Deezer. Accessed on: Mar. 30, 2022. Available: https://developers.deezer.com/sdk/javascript

[7.5] Deezer. API Wrappers, Deezer. Accessed on: Mar. 30, 2022. Available: https://developers.soundcloud.com/docs/api/sdks