

# Change History

| Version | Changes |
| --- | --- |
| 0.1 | * First Iteration |
| 0.2 | * Added Glossary, Change History, Cover Page, Context. * Restructured and reworded Sections 1-7. * Expanded section 8. |
| 1.0 | * Reformatted structure and wording in sections 3-7 for consistency * Removal of redundant appendix contents * Completed referencing * Iterated on section 2.2 architecture diagram |

# Glossary

| Term | Definition |
| --- | --- |
| Framework | A platform for developing software applications. Contains in-built functionality to assist in the development of software. |
| Front-end | The front-end of an application is defined as what the users can see and directly interact with. |
| Back-end | The back-end of an application relates to the server-side tasks and database communication. |
| Database | A structured set of data held in a computer or server. |
| SDK  (Software Development Kit) | A collection of software development tools in one package. |
| API  (Application Programming Interface) | A software interface which offers a service to other pieces of software. |
| WebSocket | A web communication protocol allowing two-way communication between a client and a server. |
| RDBMS  (Relational Database Management System) | Software that enables the creation and management of a relational database. A relational database is a structured database supporting the ability for data to link from one table to another, to form relationships. This enables ease of multi-table queries. |
| SRID  (Spatial Reference Identifier) | Indicates a specific spatial reference system (i.e. a set of parameters to describe geometry). e.g. Longitude and Latitude. |
| ACID  (Atomicity Consistency, Isolation, Durability) | Principles for data reliability.  Atomicity - uninterruptible operations,  Consistency - data integrity after deletion, insertion and updating,  Isolation - transactions will not affect others,  Durability - changes guaranteed to remain. |
| ORM  (Object-relational mapper) | Object-relational mappers (ORMs) are software libraries that enable object-relational mapping when interacting with data from a database system. Object-relational mapping refers to modelling and accessing database schemas using objects within an object-oriented programming language. |
| SQL  (Structured Query Language) | A programming language used to interact with databases, providing the ability to directly query for data, insert data, update, and remove data. Typically used within relational databases. |
| NoSQL | A database paradigm that stores data in an unstructured manner, typically without a predefined schema, as opposed to the relational database paradigm. |
| JSX  (JavaScript Syntax Extension) | An extension to the JavaScript programming language which provides methods to render React components. |
| MVC  (Model-View-Controller) | A software design pattern which separates software into three interconnected components. |

Contents

[**Change History**](#_heading=h.gjdgxs) **1**

[**Glossary**](#_heading=h.4d34og8) **1**

[**1 Introduction**](#_heading=h.3j2qqm3) **5**

[1.1 Purpose](#_heading=h.2bn6wsx) 5

[1.2 Audience](#_heading=h.3o7alnk) 5

[1.3 Project Background](#_heading=h.23ckvvd) 5

[**2 Context**](#_heading=h.dgqf6xbl1gw4) **6**

[2.1](#_heading=h.28h4qwu) Summary of Potential Technology 6

[2.1.1 Application Requirements](#_heading=h.sqyw64) 6

[2.1.2 Front-end Framework](#_heading=h.3cqmetx) 6

[2.1.3 Back-end Framework](#_heading=h.1rvwp1q) 6

[2.1.4 Database](#_heading=h.4bvk7pj) System 6

[2.1.5 Streaming SDK](#_heading=h.2r0uhxc) 6

[2.2 Potential Application Architecture](#_heading=h.1664s55) 7

[**3 Front-end Framework**](#_heading=h.mnlkefwgmrwk) **8**

[3.1 Application and Evaluation Criteria](#_heading=h.25b2l0r) 8

[3.2 Top Three Candidates](#_heading=h.2iq8gzs) 8

[3.2.1 React](#_heading=h.xvir7l) 8

[3.2.2 Angular](#_heading=h.3hv69ve) 8

[3.2.3 Vue](#_heading=h.1x0gk37) 9

[**4 Back-end Framework**](#_heading=h.gpccev5zebw5) **10**

[4.1 Application and Evaluation Criteria](#_heading=h.2w5ecyt) 10

[4.2](#_heading=h.39kk8xu) Top Three Candidates 10

[4.2.1 Django](#_heading=h.1opuj5n) 10

[4.2.2 Ruby on Rails](#_heading=h.48pi1tg) 11

[4.2.3 ExpressJS](#_heading=h.2nusc19) 11

[**5 Database System**](#_heading=h.a0zudg8iyzo2) **12**

[5.1 Application and Evaluation Criteria](#_heading=h.3mzq4wv) 12

[5.2 Top Three Candidates](#_heading=h.40ew0vw) 12

[5.2.1 MySQL](#_heading=h.2fk6b3p) 15

[5.2.2 PostgreSQL](#_heading=h.q6lbhfc6qfdz) 15

[5.2.3 Caching Systems](#_heading=h.esbfo6kyfih6) 16

[5.2.3.1 Redis](#_heading=h.3ep43zb) 16

[5.3 Object-Relational Mappers](#_heading=h.1tuee74) 16

[5.3.1 Node.js ORMs](#_heading=h.4du1wux) 16

[5.3.2 Python ORMs](#_heading=h.2szc72q) 17

[**6 Messaging Solutions**](#_heading=h.n4ostjuspcty) **18**

[6.1 Application and Evaluation Criteria](#_heading=h.meukdy) 18

[6.2 Top Three Candidates](#_heading=h.36ei31r) 18

[6.2.1 TalkJS](#_heading=h.1ljsd9k) 18

[6.2.2 SendBird](#_heading=h.45jfvxd) 18

[6.2.3 Chat Engine](#_heading=h.2koq656) 18

[**7 Music Streaming**](#_heading=h.809booqvbwqg) **SDKs 19**

[7.1 Application and Evaluation Criteria](#_heading=h.3jtnz0s) 19

[7.2 Top Three Candidates](#_heading=h.1yyy98l) 19

[7.2.1 Spotify](#_heading=h.4iylrwe) 19

[7.2.2 Apple Music](#_heading=h.2y3w247) 20

[7.2.3 Deezer](#_heading=h.1d96cc0) 20

[**8 Conclusion**](#_heading=h.jnfam1fyk1t9) **22**

[8.1 Front-end Framework](#_heading=h.2ce457m) 22

[8.2 Back-end Framework](#_heading=h.rjefff) 22

[8.3 Database System](#_heading=h.3bj1y38) 22

[8.4 Messaging Solution](#_heading=h.1qoc8b1) 22

[8.5 Music Streaming Platform](#_heading=h.4anzqyu) 22

[**References**](#_heading=h.gqy1rmapvjeu) **24**

3.x [Front-end Framework](#_heading=h.42ddq1a) 24

4[.x Back-end Framework](#_heading=h.2hio093) 25

5.x [Database System](#_heading=h.wnyagw) 25

6[.x Messaging Solution](#_heading=h.3gnlt4p) 26

7[.x Streaming SDK](#_heading=h.1vsw3ci) 27

# 1 Introduction

## 1.1 Purpose

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

## 1.2 Audience

This report is aimed at the stakeholders involved in the development of the Virtual Jukebox application. The main stakeholder is Amristar, which is the supervisor and client of the project. Stakeholders also include the staff involved with Computing Capstone Project 1 at Curtin University, which act as the co-supervisors and the Developers of the application.

## 1.3 Project Background

Jukeboxes were once a common sight at bars, discos and any social gathering. They allowed users to choose a song from a playlist to add to a queue that would eventually be played at the gathering. However, due to the explosive rise of music streaming services, jukeboxes have since become nothing more than a novelty. The Virtual Jukebox application aims to reproduce the functionality of a physical jukebox in a digital, web application format. Users will be able to connect to a local jukebox by selecting a location on a map or by scanning a QR code, and be able to vote for songs from a playlist which will be added to the queue.

# 

# 2 Context

## 2.1 Potential Technology Summary

### 2.1.1 Application Requirements

The Virtual Jukebox application must provide a mobile compatible website for the users to interact with the application. This will be accomplished using a front-end web development framework to create the interface. Users must also be able to create an account as either a host or a guest and be able to log into their accounts. User data must also be persistent, and a database management system will be required to achieve this. To read and write data to a database, a back-end framework will be required to serve as an interface between the client and the database. The back-end will also be responsible for receiving events from connected users and broadcasting these events to all connected users in the same session to create a live-chat room and a voting system. The Virtual Jukebox application must also provide host user accounts with the ability to create their own rooms and provide the functionality for playing music which will be achieved using a music streaming SDK.

### 2.1.2 Front-end Framework

The most suitable front-end framework researched was Vue. As the project has a fairly low number of front-end requirements, a simple and effective framework such as Vue was found to be suitable to be able to meet the requirements of the Virtual Jukebox application.

### 2.1.3 Back-end Framework

The most suitable back-end framework researched was Django. Django provides many of the application’s requirements natively such as user authentication, protections against common security vulnerabilities, and was found to be the most similarly aligned with the current skills of the team.

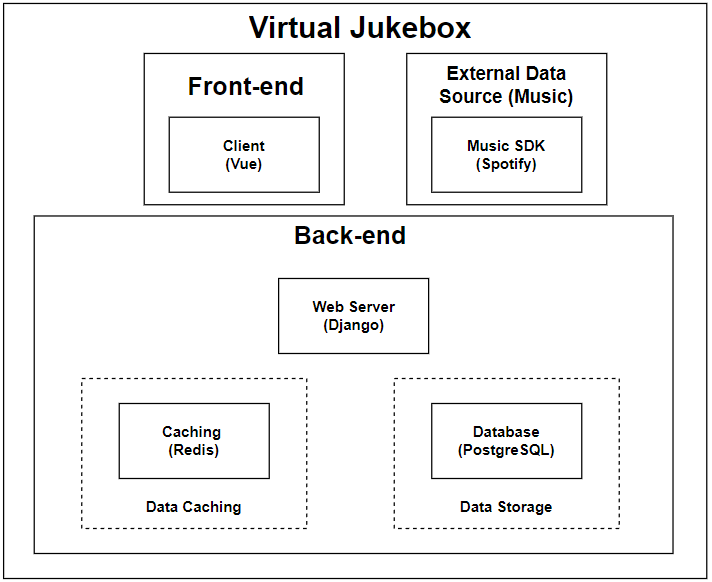
### 2.1.4 Database System

The most suitable database system researched was PostgreSQL, due to a variety of reasons including more support for geo-location data type storage, and more scalability support, alongside improved large-scale performance, (when compared to other DBMSs including MySQL). Django ORM will be used alongside PostgreSQL to enable ease of integration with the back-end and its database. The ORM will also integrate without much additional configuration with the back-end due to Django supporting the Django ORM natively. This will remove the need to learn and implement an entirely different language to the back-end when interacting with the database.

### 2.1.5 Streaming SDK

The most suitable music streaming SDK researched was Spotify. Spotify provides all the features required for the Virtual Jukebox application, has more in-depth documentation than the alternatives and has the highest level of engagement from both Spotify and Developers.

## 2.2 Potential Application Architecture



***Figure 1*** Architecture for the Virtual Jukebox application.

Figure 1 represents the architecture of the Virtual Jukebox application. Users will interact with the application through the front-end using the interface created with Vue, and the retrieval and sending of data will occur through the Django web server. Data will be stored in a PostgreSQL database, and frequently used data will be stored in a caching system provided by Redis. Music will be streamed for host profiles through the Spotify Web Playback SDK.

# 

# 3 Front-end Framework

## 3.1 Application and Evaluation Criteria

The Virtual Jukebox application must provide the user with a mobile compatible interface to allow the user to interact with the application. The front-end must be responsive, as the application will provide real-time updates to the user such as new chat messages and the current song queue list.

* **Learnability:** The front-end framework is required to be easy to learn and adapt, as it will need to be adopted by the group which has limited experience with front-end web development frameworks.
* **Resources:** The selected framework must have extensive and up-to-date documentation and resources available. A framework that may be considered easier to learn, but possesses incomplete or out-of-date documentation will be considered as inferior to one with extensive documentation.
* **Performance:** The Virtual Jukebox application will be required to stream music over a mobile or desktop internet browser. This requirement means that the selected front-end framework must be able to provide high performance, as out-of-sync music playback is undesirable. The selected front-end framework must be responsive, as the user experience is of high importance.
* **Mobile Compatibility:** The front-end framework must be able to be used to create a webpage that is compatible with both desktop and mobile environments, as it is expected that most users will be accessing the webpage on a mobile device.

## 3.2 Top Three Candidates

### 3.2.1 React

React is a perfectly suitable front-end framework that satisfies all given requirements of the Virtual Jukebox application. Through prototyping, React was found to be relatively simple to read and understand. The team has limited React and JSX experience, but it was concluded through prototyping that it would be achievable to learn React in the given time frame. React has an enormous community [3.1], and therefore a large number of resources [3.1]. Due to frequent updates, React can often have out of date official documentation [3.2], being generally out of date [3.2]. React is an extremely high-performance framework [3.2], and can be used to create responsive applications [3.3]. React is also mobile compatible, with all of the components being able to scale to fit the display of a mobile device [3.4]. React is the most popular front-end JavaScript framework available [3.5], and given that it satisfies all of the requirements of the Virtual Jukebox application, it is a perfectly acceptable framework to use for the project.

### 3.2.2 Angular

Angular is a perfectly suitable front-end framework that satisfies all given requirements of the Virtual Jukebox application. Through prototyping, Angular was found to be difficult to learn due to its complex and structured MVC approach, however excellent official documentation and a large number of online resources led to the conclusion that Angular has mild learnability. As mentioned, Angular has a large ecosystem [3.6], excellent official documentation, and therefore an extremely large set of online resources [3.7], satisfying the resources requirement. Angular is an extremely high-performance framework [3.7], and thus can be used to create highly responsive interfaces. Angular is also mobile compatible, with all of the components being able to scale to the size of a mobile device [3.8], especially the Angular-Material component library provided by Google. Angular is a feature-rich and complicated framework, given that it satisfies all of the requirements of the Virtual Jukebox application, it is a perfectly acceptable framework to use for the project.

### 3.2.3 Vue

Vue is a perfectly suitable framework for front-end development although it is not as feature rich or as popular as Angular or React [3.7], but its main strength is its simplicity [3.9]. Vue is considered the easiest framework to learn and get started with, boasting that you can have an app started “in minutes” [3.10]. Through prototyping, Vue was found to be the easiest framework to learn and adapt. Vue has a relatively small ecosystem and number of resources [3.9], making it more difficult to find help for Vue specific issues and find useful extensions. Vue’s simplicity is generally seen as a downside due to it being more work than necessary to implement complex reactivity [3.13], and the flexibility of the framework usually leads to more errors than its React and Angular counterparts [3.12]. Vue, like React and Angular, is seen as completely mobile compatible [3.12], and its components can easily scale onto a mobile screen [3.13]. Vue is generally seen as a perfect framework for simple, small-scale applications, and for this reason, and the reasons above, Vue is considered a suitable front-end framework for the Virtual Jukebox.

# 

# 4 Back-end Framework

## 4.1 Application and Evaluation Criteria

The Virtual Jukebox application will require the retrieval of data from an external server and it will not be possible to have all processing occur on the client side of the application. Virtual Jukebox will require communication between a client and a server to allow for the client side of the application to provide updates in real-time. In order to implement features including the live chat room and users having the ability to vote for songs, the WebSocket API can be used. The WebSocket API is used to establish bidirectional communication between a browser and a server, which allows users to send an event to a server and receive responses to events rather than having to poll for a response. The WebSocket protocol is fully supported by all modern web browsers and is not constricted to any specific back-end technology. All proposed back-end technologies have different implementations for handling events and have access to specific libraries [4.1]. Back-end frameworks will be assessed using the following criteria:

* **Learnability:** The framework is required to be easy to learn and adapt, as it will need to be adopted by the group which has differing levels of experience. The selected back-end framework must be considered to have a low learning curve. A framework may be deemed to have a steep learning curve, but if it aligns with the current skills of the group, it can be deemed as a viable option.
* **Resources:** The adoption rate of the framework will be used in the consideration of its available resources, as more resources are typically available for more popular technologies. A framework may be considered easy to learn, but have limited or out-of-date resources available which could hinder the production of the application.
* **Database Support:** The back-end technology selected must also be able to connect to a database management system in order to store and retrieve user data. Due to the frequent access and modification of data required by the project, an ORM solution is essential.
* **Security:** Virtual Jukebox will allow users to create and log into their own accounts. Users will be able to create accounts as a host or a guest, and each role will have access to different functionality. Authentication will be required to verify the identity of the individual attempting to login and restrict access to certain features and pages. Due to this, security will also be considered to prevent any possible data breaches.

## 4.2 Top Three Candidates

### 4.2.1 Django

Django was found to be a perfectly suitable back-end framework for the Virtual Jukebox application given the requirements. Through prototyping, Django was found to be extremely simple to learn and adapt, especially when compared to Ruby on Rails and ExpressJS. Django has an extremely large ecosystem [4.2], including an enormous community with thousands of extensions [4.3], and great up-to-date official documentation [4.3]. Prototyping showed that Django has no issues connecting to a PostgreSQL database, and Django’s ORM solution was found to be satisfactory. Django was found to have an impressive number of in-built security features [4.2], providing protection against SQL injections, cross-site scripting and other common back-end attacks [4.2]. Overall, Django satisfies all of the requirements of the Virtual Jukebox application, and is therefore perfectly suitable as a back-end framework.

### 4.2.2 Ruby on Rails

Ruby on Rails (RoR) was found to be a perfectly suitable back-end framework for the Virtual Jukebox application given the requirements. Through prototyping, RoR was found to be moderately simple to learn, although less so than Django. Ruby’s flexible implementation allows the code to be easily modified and migrated, allowing for faster development [4.4]. RoR has a large ecosystem, including a large number of community resources, extensions, and excellent official documentation [4.4]. Prototyping showed RoR had no issues connecting to a PostgreSQL database, and the ORM solution of ActiveRecord was deemed to be satisfactory. Like Django, RoR comes built in with a large number of security features, protecting against a large range of common attacks [4.4]. RoR is scalable, high performance and a perfectly suitable back-end framework for the Virtual Jukebox application.

### 4.2.3 ExpressJS

ExpressJS is an extremely suitable back-end framework that can be used in the development of the Virtual Jukebox application given the requirements. Through prototyping, ExpressJS was found to have a moderate learnability. ExpressJS supplies the user with a debugger, which is an excellent tool for development, but it uses NodeJS, which was found to have a moderate learning curve. ExpressJS has an extremely large community and ecosystem, with a large number of resources available [4.5], however some of the modules that are frequently used have poor documentation [4.5]. ExpressJS contains the library Socket.io, which if used, could simplify messaging requirements of the Virtual Jukebox application. ExpressJS was found to support any database available [4.6], satisfying the database support requirement. Furthermore, NodeJS and ExpressJS both have a large number of ORM solutions [4.6]. ExpressJS has security capabilities, and can be a Secure back-end framework to use, but it does not come built in with any built-in security features [4.7] like RoR or Django.ExpressJS satisfies all the criteria for the Virtual Jukebox application and is a suitable back-end framework that would be able to be used to develop the application.

# 

# 5 Database System

## 5.1 Application and Evaluation Criteria

The Virtual Jukebox application requires a database implementation as it involves multiple user types (guests and hosts), and (virtual) jukeboxes at multiple locations, each supporting playlists of songs. Specifically, the database will be responsible for storing user accounts including login credentials and user types, saved host playlists including their list of songs, and jukebox data including its host, chat logs per session, songs in the queue, a connection code, guest tokens, and location data (geographically based). The following criteria outlines the technology needs and considerations in the storage of this data:

* **Performance and Scalability**:Although the scale of the application will be small initially, it has the potential to be exposed to a vast number of users, hence requires sufficient scalability in data storage in terms of quantity, and data types. It will also need to be efficient for processing large volumes of data in short timeframes, therefore performance is a prime consideration. Caching will also be considered to provide quicker access to frequently used data. Indexes will additionally be utilised for access to frequently used data that must remain consistent/persistent as opposed to a cache which possesses risk of data loss due to on-memory storage
* **Security**: Security is another concern for the application, due to the fact that user data will be stored. Confidentiality will need to be addressed regarding the storage of user data, especially regards to authentication methods such as passwords. Encryption methods will provide security via data confidentiality, whereas databases with consistency and atomicity (ACID compliance properties) will enable integrity across all types of data when performing queries/operations. Accessibility must also be considered to prevent data leakage via authentication to databases.
* **Learnability**:As the development of the application will take place within a restricted time-frame of approximately one year (half development, half planning), it is crucial that the database system selected does not possess a steep learning curve for ease of development. The system decided upon will need to possess enough features to be suitable for the application’s requirements, while still being quickly learnable in comparison to database systems considered difficult.

## 5.2 Top Three Candidates

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. After several analyses of these eight systems, three have stood out as being suitable for use in production.

Two types of database types have been considered for use with the Virtual Jukebox application including relational databases (structured data driven), and NoSQL (unstructured data driven) databases. After consideration between the two, 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 to assist in future scalability [5.1].

The 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 [5.2]-[5.5]. 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 [5.6]. As relational databases seem more suitable in this scenario, MongoDB was ruled out. Redis is a consideration, however, due to its ability to be used in conjunction with other database systems as a caching tool to improve performance of frequently accessed data, even though it is a NoSQL system [5.4].

Redis could be used alongside a separate traditional RDBMS, simply for caching frequently accessed data such as chat-messages, session user-count, guest-credit count, and the songs in a 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 application’s data storage requirements. Each of these three systems are open-source, allowing for commercial usage without fees if needed [5.5]. 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 [5.5]. The following table describes comparison points between the three systems in terms of their evaluation criteria, and other relevant properties.

Points derived and summarised from [5.5]-[5.11] and [5.1]:

| **Criteria** | **MySQL** | **PostgreSQL** | **Redis** |
| --- | --- | --- | --- |
| **Utility** | - SQL RDBMS, which aims to be fast and reliable with a limited feature-set.  - Base edition is free. Other editions are paid. | - 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. | - 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). |
| **Performance** | - Faster in read-only operations (compared to other SQL DBMS).  - Memory Storage Engine provides frequently used table support.  - Query cache for frequently used queries. | - 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. | - 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 I/O.  - 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) time via hashing). |
| **Scalability** | - Scales vertically (Requires more powerful hardware to upscale).  - Less features to support large-scale operations. | - Scales vertically (Requires more powerful hardware to upscale).  - Multi-Version Concurrency Control to support many concurrent users (via parallel queries).  - More indexing methods compared to MySQL. | - 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). |
| **Security** | - TLS encryption support. | - Data encryption algorithms supported: AES, 3DES  - Native SSL encryption support for connections. | - Potential data loss from storage of volatile in-memory data.  - Designed for use in trusted environments.  - TLS encryption support. |
| **Learnability** | - Not case sensitive.  - Considered easy to set-up and use.  - Well-documented. | - Case Sensitive.  - Generally considered harder to set-up and use, largely due to its inconsistent documentation. | - 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. |
| **NoSQL Support** | - Later releases provide some NoSQL capabilities including JSON data management and storage. | - Some NoSQL support allows for the storage and management of unstructured data (JSON/key-value). | - Fully NoSQL without native SQL support.  - Unstructured data support. |

### 5.2.1 MySQL

MySQL is a suitable RDBMS for implementation within the Virtual Jukebox application due to its simplicity of use, and read performance. MySQL is considered easy to set up and operate due to its low complexity and feature set, which would be beneficial to the development time of the Virtual Jukebox application [5.1]. It is also fairly easy to understand syntactically, as most commands are similar to that of plain English (e.g. showing all tables would require a SHOW TABLES command). MySQL is also known for good read performance compared to other RDBMSs, which would be beneficial for the application’s requirements of multiple users reading song queue statuses [5.1]. There are two primary disadvantages of MySQL’s usage. Firstly, its licensing which although is free in its base form, requires payment for additional features (including enterprise tools to assist in scalability) [5.24]. Secondly, MySQL loses in terms of read and write speeds in larger databases when compared to PostgreSQL, which would be a hindrance to consistent performance for when the Virtual Jukebox application up-scales [5.1].

### 5.2.2 PostgreSQL

Although MySQL would be sufficient and is generally considered easier to use according to [5.1], PostgreSQL is also highly suitable for implementation within the Virtual Jukebox application 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 highly scalable and supports multi-version concurrency control enabling efficient concurrent operations which would reduce the change of locking for when multiple users access the same data [5.1]. Concurrent operation support would be useful when multiple users of the Virtual Jukebox application are requesting songs and/or changing song priorities.

Both PostgreSQL and MySQL provide several indexing methods, but PostgreSQL additionally supports expression indexing (for faster function calls in queries), and partial indexing (for specific queries based on a condition) [5.12]. PostgreSQL natively complies with ACID transaction properties, enhancing data integrity, although MySQL does too with the help of the InnoDB storage engine (which comes with MySQL by default) [5.13]. PostgreSQL also 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 [5.13] [5.4]. This would be useful in the case of upscaling the application to include additional features in the future which would require unstructured data storage. One aspect of PostgreSQL which contributes to its learning difficulty includes its syntax, which is more symbolic compared to MySQL's (e.g. showing all tables requires a ‘\dt’ command). This wouldn’t be of much concern in implementation with the help of an ORM, as it would abstract its native syntax into library calls.

### 5.2.3 Caching Systems

Caches are often used when users or software require frequent access to some data store that utilises disk I/O and/or web-based access methods. This is to enable faster access to such data, which provides an overall faster application by temporary storage on-memory instead of on-disk. The Virtual Jukebox application would benefit from such a feature for scenarios such as when guests are accessing data from the server multiple times in short time-frames. For example, accessing the current song queue several times between requesting songs.

#### 5.2.3.1 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 will 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]. Contents that do not need to be frequently accessed or do not need real-time access in the Virtual Jukebox application, 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 be stored in the Redis on-memory cache, with scheduled writes to the disk for backups/persistence depending on the type of data. 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].

## 5.3 Object-Relational Mappers

Object-relational mapping refers to the process of writing database queries and accessing data (from a database) via the use of object-oriented techniques using a selected programming language, instead of relying on writing SQL queries [5.14]. By utilising an ORM, there is less need for writing complex SQL queries in database management by instead using calls to a library. ORMs typically form a connection between a software system’s back-end, and its database system. By doing so, ORMs provide ease of use for developers not well versed in specific database systems, especially for the intricacies of query languages between database systems. An ORM has been selected for use within the back-end of the Virtual Jukebox application for a variety of reasons. Firstly, as PostgreSQL possesses a higher learning curve compared to other relational database systems [5.5], and is the prime candidate for the application’s database implementation, an ORM will assist in reduced development time, especially in writing and accessing queries. Also, although ORMs typically reduce reading and writing performance (as a library call must be made to produce the corresponding SQL query), it realistically wouldn’t be a concern in the context of the Virtual Jukebox application, as queries do not require complex logic (the most complex would be jukebox location queries). If the complexity of the software increases significantly, effort would be required in shifting from an ORM implementation to either a more efficient one, or to a pure SQL querying system.

### 5.3.1 Node.js ORMs

Three Node.js-based ORMs were selected for consideration including Sequelize, TypeORM, and Prisma. As each of these ORMs are Node.js-based, they will pair well with a node.js back-end framework such as Express.js. Each ORM also supports PostgreSQL, and MySQL as well as many other database systems [5.15]-[5.17]. Prisma was ruled out due to its lack of functionality in supporting PostgreSQL’s PostGIS data types such as geometry types which would be useful for storing the Virtual Jukebox application’s host locations [5.18]. Sequelize was also ruled out due to its lack of support, and poor documentation expressed in the community, which would be a detriment to the restricted time-frame of development [5.19]. TypeORM was selected to be the most suitable out of these three considered ORMs, due to its feature set, including the ability to be used alongside a Redis cache [5.20]. Due to the fact that ORMs abstract database interaction using a library as an interface, it will restrict complex SQL from being executed via the ORM which could affect ease of scalability if extended features are added in the future. TypeORM combats this by incorporating a raw query option, where raw SQL may be executed as per the base database system’s standard [5.21]. A disadvantage of TypeORM however, includes the requirement of defining relationships at both ends of a two table relationship, which is syntactically complicated. The documentation also does not elaborate on this requirement and is overall lacking in terms of detail, which could hinder development of the Virtual Jukebox application.

### 5.3.2 Python ORMs

Two Python-based ORMs taken into consideration include SQLAlchemy, and the Django ORM. For the case of a Python-based back-end framework (such as Django), these ORMs will integrate well with it due to their shared language and support. Comparing the two, they are both suitable for the Virtual Jukebox application due to the ability to support PostgreSQL and MySQL, and both possess large communities with extensive documentation with SQLAlchemy having the most support [5.22]. This would be useful for speeding up development when issues arise regarding the ORM. Django ORM’s learning curve however, is considered much lower in comparison to SQLAlchemy due to its lower complexity [5.22]. Django ORM is a prime candidate for the ORM for use with the application’s database due to its low difficulty, and its ability to model spatial data from PostGIS using GeoDjango [5.23].

# 

# 6 Messaging Solutions

## 6.1 Application and Evaluation Criteria

The Virtual Jukebox application must provide a live chat-room for each session where users can communicate through instant messages in a group setting, with chat messages being programmatically moderated for offensive language. Messaging solutions are assessed based on the following criteria:

* **Price:** The price of the premade messaging solution is to be considered due to the low budget of the project.
* **Provided Features:** The messaging solution must provide the requirements of the Virtual Jukebox application, and extra features may be deemed unnecessary.
* **Learnability:** The selected messing solution must provide developer-friendly documentation, and be easily integrated with the selected front-end framework.

## 6.2 Top Three Candidates

### 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 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 [6.1].

### 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].

### 6.2.3 Chat Engine

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].

# 

# 7 Music Streaming Platform

## 7.1 Application and Evaluation Criteria

The Virtual Jukebox application will 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 track playback history
* View track queue
* Add tracks to the queue

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

* Create playlists
* Retrieve playlists
* View playlist tracks
* Search for tracks in the host’s library
* Search for tracks in the streaming platform’s catalogue
* Add tracks to playlists
* Remove playlist tracks
* Sort playlist tracks by artist
* Sort playlist tracks by genre
* Sort playlist tracks by year

## 7.2 Top Three Candidates

### 7.2.1 Spotify

The Spotify SDK is a suitable music streaming SDK that meets all the requirements except for having an API with the ability to sort tracks by genre or year. The level of engagement with the Spotify Platform services, both from Spotify itself and Developers, is its most desirable factor. The documentation for both the Web Playback SDK [7.1] and Web API [7.2] is extensive–including code examples, troubleshooting information and guides. As for Community engagement, there are many tutorials on YouTube about many aspects of development with the SDK–user authentication, making requests, and front-end integration.  
 An important consideration, however, is the rate limits of the API itself. There is no clear information about rate limits. In Rate Limits, Spotify mentions that the rate limit of an app’s calls to the API is based on the number of requests made in a 30 second window, and that if the app exceeds the limit further requests will be denied until a cooldown period has passed [7.3]. Considering the minimal usage requirements for Virtual Jukebox, rate limits will not be a concern. However, if any issues were to arise, Spotify offers strategies for developing apps with rate limits in mind.  
 Furthermore, any usage of the Spotify platform must be strictly non-commercial. Spotify allows Developers to develop and distribute streaming applications which are for only personal, non-commercial use [7.4].

### 7.2.2 Apple Music

The Apple Music SDK is a suitable music streaming SDK, satisfying all requirements apart from featuring an API with the ability to sort tracks by genre or year. Unlike the Spotify SDK, the Apple Music SDK allows Developers to use the Apple Music API through the playback controller directly, rather than having to interact with both elements separately [7.5]. This will allow for cleaner, more concise code which will allow the time to spend less time on core application development. This will allow for more time to be spent enhancing user experience through feature-adding, testing and bug-fixing. However, unlike the Spotify SDK, the level of engagement with the Apple Music SDK is much lower. The Documentation for the SDK is similar in quality to Spotify, but there is less information (video tutorials, blog posts, guides etc.) available.

Once again, it is important to consider request rate limits. The Apple Music API documentation doesn’t offer any concrete information about rate limits. API documentation states that it limits the number of requests an app makes over a certain time period and that if this limit is exceeded, the app will not be able to make additional requests until the request rate has reduced [7.6]. However, as previously mentioned, the requirements for Virtual Jukebox don’t indicate this will be an issue.

Additionally, usage of the Apple Music platform must also be non-commercial. The documentation informs that Developers using the SDK cannot monetise access to Apple Music through their application in any way [7.7].

### 7.2.3 Deezer

The Deezer SDK is a suitable music streaming SDK that meets all the requirements of the Virtual Jukebox application except for having an API with the ability to sort tracks by genre or year. The Deezer SDK lets Developers access Deezer API through a playback controller running on their website [7.8]. Once more, this will allow for more maintainable code, offering the same ongoing development benefits mentioned about Apple Music. However, the Deezer SDK has a low engagement level. The documentation is more user-friendly than that of the Apple Music SDK but not as detailed as the Spotify SDK documentation. Also, there is more information available about using the SDK–example projects on Deezer website–but the level of third-party support is not like Spotify’s [7.9].  
 For rate limits, Deezer states that there is a limit to the amount of requests an application can make [7.10]. However, there is no more information provided. This is not preferred as the team needs to have an idea of the behaviour of the API at higher loads to keep in mind during development.  
 Much like both the Spotify and Apple Music Platforms, the Deezer SDK must, too, be non-commercial only. Developers can only use the Deezer platform for non-commercial purposes in non-commercial environments. Also, Developers must tell users that their application’s use must be strictly private–for use only within a ‘family scope’ [7.11].

# 

# 8 Conclusion

## 8.1 Front-end Framework

Angular, React and Vue were found to be perfectly suitable for the Virtual Jukebox application. Through extensive research, it was found that the strengths of Vue were most suited to the requirements of the project. The Virtual Jukebox application has a low number of front-end requirements, and therefore a simple and effective framework such as Vue was found to be the most appropriate. Vue will be able to satisfy the requirements of the Virtual Jukebox application.

## 8.2 Back-end Framework

Django is the most suitable back-end framework for use in the development of the Virtual Jukebox application. Although all the researched back-end frameworks are compatible with the requirements of the Virtual Jukebox application, Django was found to be preferable in certain areas. Django was viewed positively due to its protection against common web application security vulnerabilities such as cross-site scripting and SQL injection, allowing for more focus on the development of the required features of the Virtual Jukebox application. Django also has extensive and detailed documentation, and provides support for use with a PostgreSQL database.

## 8.3 Database System

After prototyping and a comparative analysis between MySQL and PostgreSQL, Virtual Jukebox will utilize the PostgreSQL database system to enable persistent data storage of aspects including user data, jukebox data including their locations, and playlist data. This is primarily due to PostgreSQL’s extended support in geographical location support (with the PostGIS library), as well as its pure open-source licensing, and compatibility with a wider range of data types compared to MySQL. Although PostgreSQL is often considered more difficult to learn, that will be combated with the usage of the Django ORM which will pair well with the Django back-end due to its native integration. Django ORM enables an easier process of development in database interaction due to the lack of needing to write SQL queries, and the linkage of objects to database tables and entities. Additionally, Virtual Jukebox will incorporate Redis caching to be used alongside PostgreSQL in order to provide quicker access to frequently used data.

## 8.4 Messaging Solution

Chat Engine was found to be 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 front-end. Due to SendBird and TalkJS charging much higher prices which increase as the scale of the application increases, these are deemed unsuitable as they also offer many features which are unneeded in the current specifications. An instant messaging system will be manually implemented utilising the WebSocket API on the client side to provide updates through the Vue front-end, and message events will be handled using Django in the back-end of the Virtual Jukebox application.

## 8.5 Music Streaming Platform

With all aspects of the candidate streaming SDKs considered, it was found that the most appropriate SDK is Spotify. The SDK has a superior level of engagement–Developers utilising the SDK benefit from many sources of information (video tutorials, guides, blog posts etc.) for development. Because of this, developing the core application will be much more straight-forward and the time saved by working with a more familiar SDK will give the team more time to spend on other areas of development–feature-adding, testing, and bug-fixing. Spending more time in these areas will contribute to providing users with a more enjoyable experience.

# 

# References

## 3.x Front-end Framework

[3.1] M. Jablecnik. ”How large is Flutter vs React Native community in 2021” dev.to. <https://dev.to/mjablecnik/how-large-is-flutter-vs-react-native-community-in-2021-2df5> (accessed Mar. 28, 2022).

[3.2] javaTpoint. ”Pros and Cons of ReactJS” javatpoint.com. <https://www.javatpoint.com/pros-and-cons-of-react> (accessed Mar. 28, 2022).

[3.3] React, “React” Reactjs.org, <https://reactjs.org/> (accessed Mar. 28, 2022)

[3.4] React Naive.”Who’s using React Native?”.reactnative.dev. <https://reactnative.dev/showcase> (accessed Mar. 28, 2022).

[3.5] Niel Patel, “Top 10 Most Popular Frontend Frameworks For Web Development 2022” makeanapplike.com <https://makeanapplike.com/most-popular-front-end-web-frameworks-for-developments/> (accessed Mar. 28, 2022)

[3.6] Altexsoft,“The good and the bad of angular development,” altexsoft.com. <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/>. (accessed Mar. 28, 2022).

[3.7] A. Gazta, “Good and bad of angular development,” greycampus.com. <https://www.greycampus.com/blog/programming/good-and-bad-of-angular-development>. (accessed Mar. 28, 2022)

[3.8] S. Merenych and A. Popovych, “Best examples of angular websites and applications: Top websites built with angular - clockwise software,” clockwise.software <https://clockwise.software/blog/best-angular-applications/>. (accessed Mar. 28, 2022).

[3.9] J. Dakowicz, “React vs Vue: Which One to choose in 2022?,” pagepro.co <https://pagepro.co/blog/react-vs-vue-comparison/>. (accessed Mar. 29, 2022).

[3.10] Maja Nowak, “Vue vs React in 2022: Which Framework to Choose and When” Monterail.com https://www.monterail.com/blog/vue-vs-react-2021 (accessed Mar 30, 2022)

[3.11] Altexsoft, “The Good and the Bad of Vue.js Framework Programming” altexsoft.com. <https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/>. (accessed Mar. 30, 2022). .

[3.12] E. Asiuwhu, “Comparing Vue.js mobile app development frameworks,” blog.logrocket.com. <https://blog.logrocket.com/comparing-vue-js-mobile-app-development-frameworks/>. (accessed Mar. 29, 2022).

[3.13] W. Mista Oct 8, “12 companies that have utilized vue.js in their applications,” naturaily.com. <https://naturaily.com/blog/companies-vue-js-applications>. (accessed Mar. 29, 2022).

## 4.x Back-end Framework

[4.1] Mozilla, "The WebSocket API (WebSockets) - Web APIs | MDN'', <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>. (accessed: Mar. 24, 2022).

[4.2] S. Bhatt “Pros and Cons of Django Framework for App Development” Dzone.com <https://dzone.com/articles/pros-and-cons-of-django-framework-for-app-developm> (accessed Mar. 30, 2022)

[4.3] V. Sidorenko “The Advantages And Disadvantages of Using Django” Datafloq.com <https://datafloq.com/read/advantages-and-disadvantages-of-using-django/> (accessed Mar. 30, 2022)

[4.4] FullScale, “Ruby on Rails Pros and Cons” Fullscale.io <https://fullscale.io/blog/ruby-on-rails-pros-and-cons/> (accessed Mar. 30, 2022)

[4.5] Altexsoft. “The Good and the Bad of Node.js Web App Development” Altexsoft.com. <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/> (accessed Mar. 30, 2022).

[4.6] Expressjs “Database Integration” expressjs.com <https://expressjs.com/en/guide/database-integration.html> (accessed Mar. 30, 2022)

[4.7] Oleg Romanyuk, “Express.js Security Tips: How You Can Save and Secure Your App  
Freecodecamp.org <https://www.freecodecamp.org/news/express-js-security-tips/> (acessed Mar. 30, 2022)

## 5.x Database System

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

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

[5.3]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).

[5.4] 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).

[5.5] 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).

[5.6] A. Duggal. “SQLite vs MySQL: 5 Critical Differences.” Hevodata.com. <https://hevodata.com/learn/sqlite-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).

[5.12] EDB Team. “PostgreSQL vs. MySQL: A 360-degree Comparison [Syntax, Performance, Scalability and Features].” EnterpriseDB.com. <https://www.enterprisedb.com/blog/postgresql-vs-mysql-360-degree-comparison-syntax-performance-scalability-and-features> (accessed Apr. 7, 2022).

[5.13] D. Bolton. “Why I Choose PostgreSQL Over MySQL/MariaDB.” Dice.com. https://insights.dice.com/2015/03/19/why-i-choose-postgresql-over-mysqlmariadb/ (accessed Apr. 7, 2022).

[5.14] M. Hoyos. “What is an ORM and Why You Should Use it.” Medium.com <https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a> (accessed Apr. 7, 2022).

[5.15] Sequelize. “Sequelize.” Sequelize.org. <https://sequelize.org/> (accessed Apr. 7, 2022).

[5.16] Prisma. “Supported databases.” Prisma.io. <https://www.prisma.io/docs/reference/database-reference/supported-databases> (accessed Apr. 7, 2022).

[5.17] TypeORM. “Features.” TypeORM.io. <https://typeorm.io/> (accessed Apr. 7, 2022).

[5.18] Timwis. “Other postgres column types.” GitHub.com. <https://github.com/prisma/prisma1/issues/2466> (accessed Apr. 7, 2022).

[5.19] Openbase. “Sequelize reviews.” Openbase.com. <https://openbase.com/js/sequelize/reviews> (accessed Apr. 7, 2022).

[5.10] TypeORM. “Caching queries.” <https://orkhan.gitbook.io/typeorm/docs/caching> (accessed Apr. 7, 2022).

[5.21] TypeORM. “Entity Manager API.” TypeORM.io. <https://typeorm.io/entity-manager-api> (accessed Apr. 7, 2022).

[5.22] T. Shay. “Django vs SQLAlchemy – Which Python ORM is better?” EverSQL.com. <https://www.eversql.com/django-vs-sqlalchemy-which-python-orm-is-better> (accessed Apr. 7, 2022).

[5.23] A. Adegbenro. “Using GeoDjango and PostGIS in Django.” pganalyze.com. <https://pganalyze.com/blog/geodjango-postgis> (accessed Apr. 7, 2022).

[5.24] MySQL. “MySQL Products”. MySQL.com. <https://www.mysql.com/products/> (accessed Apr. 9, 2022).

## 6.x Messaging Solution

[6.1] Talkjs, "Features - TalkJS", <https://talkjs.com/features/>. (accessed 24. Mar, 2022).

[6.2] Sendbird, "Chat Built for Online Communities | Grow Quality Engagement in Your Community", <https://sendbird.com/solutions/chat-for-online-communities>. (accessed 24. Mar, 2022).

[6.3] ChatEngine, "Chat Engine | Simple APIs and UI Kit", <https://chatengine.io/>. (accessed 26. Mar 2022).

## 7.x Streaming SDK

[7.1] Spotify, “Web Playback SDK Reference”,  
 https://developer.spotify.com/documentation/web-playback-sdk/reference/. (accessed   
 Mar. 24, 2022).

[7.2] Spotify, “Spotify Web API”,  
 https://developer.spotify.com/documentation/web-api/reference/#/. (accessed  
 Mar. 24, 2022).

[7.3] Spotify, “Rate Limits”,  
 https://developer.spotify.com/documentation/web-api/guides/rate-limits/. (accessed   
 Mar. 30, 2022).

[7.4] Spotify, “Spotify Developer Terms”, https://developer.spotify.com/terms/. (accessed   
 Mar. 26, 2022).

[7.5] Apple, “MusicKit: This class represents the Apple Music API. https://developer.apple.com/documentation/musickitjs/musickit/api. (accessed Mar. 30, 2022).

[7.6] Apple, “Generating Developer Tokens: Generate a developer token needed to make requests to Apple Music API”. https://developer.apple.com/documentation/applemusicapi/generating\_developer\_tokens?changes=\_10&language=objc. (accessed Mar. 29, 2022).

[7.7] Apple, “Apple Developer Program License Agreement”. https://developer.apple.com/support/downloads/terms/apple-developer-program/Apple-Developer-Program-License-Agreement-20211213-English.pdf. (accessed Mar. 29, 2022).

[7.8] Deezer, “Javascript SDK: The JavaScript SDK allows you to stream music, authenticate users and lets you access all the Deezer API endpoints”. https://developers.deezer.com/sdk/javascript. (accessed Mar. 30, 2022).

[7.9] Deezer, “Custom player: This sample of code shows how the SDK JavaScript interacts with an invisible widget to create your own UI. https://developers.deezer.com/sdk/javascript/example-invisible. (accessed Mar. 30, 2022).

[7.10] Deezer, “Deezer FAQs for developers”. https://support.deezer.com/hc/en-gb/articles/360011538897-Deezer-FAQs-for-developers. (accessed Mar. 30, 2022).

[7.11] Deezer, “Terms of Use”. https://www.deezer.com/legal/cgu. (accessed Mar. 30, 2022).