Software Requirements Specification

for

RhythmiQ

Lab 2, Sections 1 and 2

Version 2

Lucas Hayes, Alanna Ivey,
Spencer Hall, Andrew Behre, Bea Eggleston,
Gaby Macatangay, Dominick Rodriguez
Team Blue

Department of Computer Science, Old Dominion University
CS 411W: Professional Workforce Development II
Professor Sumaya Sanober

November 1, 2026

Table of Contents

1. Introduction 1
1.1 Purpose 1
1.2 Scope 1
1.3 Definitions, Acronyms, and Abbreviations 2
1.4 References 4
1.5 Overview 5
2. Overall Description 6
2.1 Product Perspective 6
2.2 Product Functions 8
2.3 User Characteristics 10
2.4 Constraints 10
2.5 Assumptions and Dependencies 11
3. Specific Requirements – provided in the team's Lab 2 group SRS

1. Introduction

1.1 Purpose

This document is the Software Requirements Specification (SRS) for RhythmiQ (formerly VibeQueue), a web-based, hardware-agnostic jukebox platform for small hospitality venues. It is written for the developers who will design, implement, and test the system and serves as their technical reference. It follows the structure of IEEE Std 830-1998 [1] and refines the broader product description in Lab 1 [2] into a concise, developer-oriented statement of what is to be built. Because the CS 411W prototype is a reduced-scope version of the real-world product (RWP), this document also states, consistent with the team's current implementation plan and code, which functions the prototype fully implements, partially implements, or eliminates, and which third-party services it uses in place of RWP services.

1.2 Scope

RhythmiQ lets customers of a small cafe, coffee shop, or restaurant request songs from their own smartphones, without interrupting staff, while the owner keeps final control over what plays. Most U.S. food-service venues are small businesses that cannot absorb a large technology investment [3], and music influences how long customers stay and how much they spend [4], [5], [6], [7]. Existing options leave this gap open: TouchTunes jukeboxes depend on proprietary hardware [8], Rockbot centers on music the owner curates in advance [9], and personal streaming accounts are not licensed for commercial playback [10].

The goals of the system are to:

- give customers a low-friction way to request songs through a mobile browser, with no app download;
- give owners and staff control over which songs can be requested, by genre, mood, and explicit-content rules, and over whether requests are accepted at all;
- provide an optional Pay-to-Request revenue stream for the venue, with a 5% platform fee to sustain development;
- route all RWP playback through a commercially licensed service; and
- run on devices the venue already owns, with no new hardware purchase.

The system **will**: provide a QR-code-accessed web portal for customers, using a unique QR code generated for each venue; let customers connect to a venue, with or without an account, and search for songs using music metadata from a third-party API [11]; validate every request against the venue's request status and constraints through the Vibe Engine, with staff approving requests before they enter the queue; maintain a live song queue with optional paid priority processed through Stripe [12]; give owners and staff a Business Control Center for rules, queue overrides, request availability, and revenue tracking; and play validated songs through a playback service, which is SoundMachine [13] in the RWP and the Spotify Web API [14] in the CS 411W prototype.

The system **will not**: give customers final authority over playback, since every request must pass the venue's constraints first; require or sell proprietary hardware; play music from unlicensed sources in a commercial deployment (the prototype's Spotify playback is limited to development and classroom demonstration); store payment card data locally; or require customers to install a native application.

1.3 Definitions, Acronyms, and Abbreviations

API (Application Programming Interface) – A set of rules and protocols that allows different software applications to communicate with each other.

ASCAP (American Society of Composers, Authors, and Publishers) – A U.S. performing rights organization that licenses music for commercial use.

Blacklist / Whitelist – Venue-level lists that block specific songs or genres from the queue (blacklist) or limit requests to approved genres (whitelist).

BMI (Broadcast Music, Inc.) – A U.S. performing rights organization that manages commercial music licensing on behalf of songwriters and publishers.

Business Control Center – A dashboard interface that allows business owners and staff to manage song requests, set filters, enable or disable requests, and control the atmosphere.

Commercial License – A legal permit that allows businesses to play music publicly in a commercial setting.

Django – An open-source Python web framework used to build the RhythmiQ back end [15].

Dynamic Vibe Moderation – Integrated filters, configured through the Business Control Center, that let owners set genre limits and block explicit content automatically.

Hardware-Agnostic – Software that can run on any hardware device without specific or proprietary requirements.

Local Server – The RhythmiQ component at the venue that receives validated song selections and sends them to the playback service.

Metadata – Information, such as author, creation date, or file size, that describes a data point or data set [16].

MFCD (Major Functional Components Diagram) – A diagram showing the main software components of a system and how they interact.

MusicBrainz – An open music database whose API provides song metadata to the prototype [11].

Pay-to-Request – A feature allowing customers to pay to have their song moved ahead in the queue.

Playback Service – The external service that plays validated songs through the venue's speakers: SoundMachine in the RWP and the Spotify Web API in the CS 411W prototype.

PostgreSQL – The open-source relational database used for the Venue Database [17].

Prototype – The reduced-scope, working version of RhythmiQ built during CS 411W to demonstrate the real-world product's core capabilities.

QR Code (Quick Response Code) – A scannable barcode that provides quick access to a website or application.

Revenue Stream – A source of income for a business; here, the 5% cut taken from Pay-to-Request transactions.

RWP (Real-World Product) – The full-capability RhythmiQ product, as distinct from the CS 411W prototype.

SESAC (Society of European Stage Authors and Composers) – A U.S. performing rights organization providing commercial music licensing.

Soundcharts API – An application programming interface that allows developers to integrate Soundcharts' music catalog into their applications; the original metadata source, set aside because it is no longer free [18].

SoundMachine – A music streaming service that provides licensed commercial background music for businesses; the RWP playback service [13].

Spotify Web API – Spotify's web service for controlling playback on a Spotify-connected device; the playback service in the CS 411W prototype only, because Spotify's terms do not permit commercial playback [10], [14].

SRS (Software Requirements Specification) – A document that completely describes the external behavior and requirements of a software system, as specified by IEEE Std 830-1998.

Staff Review – The step in which staff approve pending song requests into the queue or reject them. Requests the Vibe Engine cannot decide because song data is missing are marked for review.

Stripe – The third-party payment platform planned for processing Pay-to-Request payments [12].

Vibe – The mood of a place, situation, or person, and the way it makes you feel [19].

Vibe Engine – RhythmiQ's in-house song selection and validation component; evaluates customer song requests against venue-configured constraints before allowing them into the queue.

Vibe Management – The process of controlling and curating the musical atmosphere of a business.

Web-App – A software application that runs in a web browser rather than being installed locally.

1.4 References

[1] *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998, Oct. 1998.

[2] D. Rodriguez, "Lab 1: VibeQueue product description," CS 411W, Dept. Comput. Sci., Old Dominion Univ., Norfolk, VA, USA, Version 5, Sep. 20, 2026. (VibeQueue is the original name of RhythmiQ.)

[3] National Restaurant Association, "Restaurant industry facts at a glance," 2023. [Online]. Available: https://restaurant.org/research-and-media/research/industry-statistics/national-statistics

[4] R. E. Milliman, "Using background music to affect the behavior of supermarket shoppers," *J. Marketing*, vol. 46, no. 3, pp. 86–91, 1982, doi: 10.2307/1251706.

[5] A. C. North, D. J. Hargreaves, and J. McKendrick, "The influence of in-store music on wine selections," *J. Appl. Psychol.*, vol. 84, no. 2, pp. 271–276, 1999, doi: 10.1037/0021-9010.84.2.271.

[6] Broadcast Music, Inc., "New BMI/NRG study confirms consumers eat, drink, and spend more when listening to music," Apr. 24, 2023. [Online]. Available: https://www.bmi.com/news/entry/new-bmi-nrg-study-confirms-consumers-eat-drink-spend-more-when-listening

[7] MRC Data, "The soundtrack of the modern customer experience," study commissioned by Soundtrack Your Brand, Oct. 2021.

[8] TouchTunes, "Frequently asked questions," n.d. [Online]. Available: https://business.touchtunes.com/faq

[9] Rockbot, "Business music, TV & digital signage solutions," n.d. [Online]. Available: https://rockbot.com

[10] Spotify, "Spotify terms and conditions of use," 2023. [Online]. Available: https://www.spotify.com/us/legal/end-user-agreement/

[11] MetaBrainz Foundation, "MusicBrainz API," n.d. [Online]. Available: https://musicbrainz.org/doc/MusicBrainz_API

[12] Stripe, "Stripe API reference," n.d. [Online]. Available: https://stripe.com/docs/api

[13] SoundMachine, "Background music for business made easy," n.d. [Online]. Available: https://sound-machine.com/

[14] Spotify, "Web API," Spotify for Developers, n.d. [Online]. Available: https://developer.spotify.com/documentation/web-api

[15] Django Software Foundation, "Django documentation," n.d. [Online]. Available: https://docs.djangoproject.com

[16] IBM, “Metadata,” Oct. 21, 2024. [Online]. Available: https://www.ibm.com/think/topics/metadata

[17] PostgreSQL Global Development Group, “PostgreSQL documentation,” n.d. [Online]. Available: https://www.postgresql.org/docs/

[18] Soundcharts, “Music analytics platform,” n.d. [Online]. Available: https://soundcharts.com

[19] Cambridge Dictionary, “Vibe,” Dec. 9, 2021. [Online]. Available: https://dictionary.cambridge.org/us/dictionary/english/vibe

1.5 Overview

The remainder of this document is organized as follows. Section 2, Overall Description, presents the factors that shape the product and its requirements: Section 2.1 places RhythmiQ in its operating environment and shows its major components, its external interfaces, and the third-party services the prototype uses in place of RWP services; Section 2.2 summarizes its main functions, states which are fully implemented, partially implemented, or eliminated in the CS 411W prototype under the team's current implementation plan, and notes each function's progress; Section 2.3 defines the user roles and their permissions; and Sections 2.4 and 2.5 list constraints, assumptions, and dependencies, including the licensing constraint behind the prototype's playback service. Section 3, Specific Requirements, is written by the full team in the group portion of Lab 2 and states testable functional requirements for each prototype feature: account management, login, QR code creation, venue connection, and song search with validated request submission.

2. Overall Description

2.1 Product Perspective

RhythmiQ is a self-contained, web-based system that depends on a small number of external services and on devices the venue already owns. The back end is a Python web application built with Django [15] and backed by a PostgreSQL database [17]. Customers reach the system through a mobile browser by scanning the venue's QR code, and owners and staff reach it through the Business Control Center in a web browser. A local server at the venue sends each validated song to a playback service, which plays it through the venue's existing speakers, so no proprietary hardware is required. In the RWP the playback service is SoundMachine, which holds the required commercial licenses [13]; the CS 411W prototype uses the Spotify Web API [14] instead, for the reasons given in Section 2.4. The prototype's playback path is still in development.

Figure 1 shows the Major Functional Components Diagram (MFCD), organized into presentation, application, and data layers. Color indicates each component's status in the CS 411W prototype. The Vibe Engine is partially implemented because it validates requests without mood rules or recommendations, and the playback path, made up of the Local Server and the Spotify Web API that the prototype uses in place of SoundMachine, is still in development.

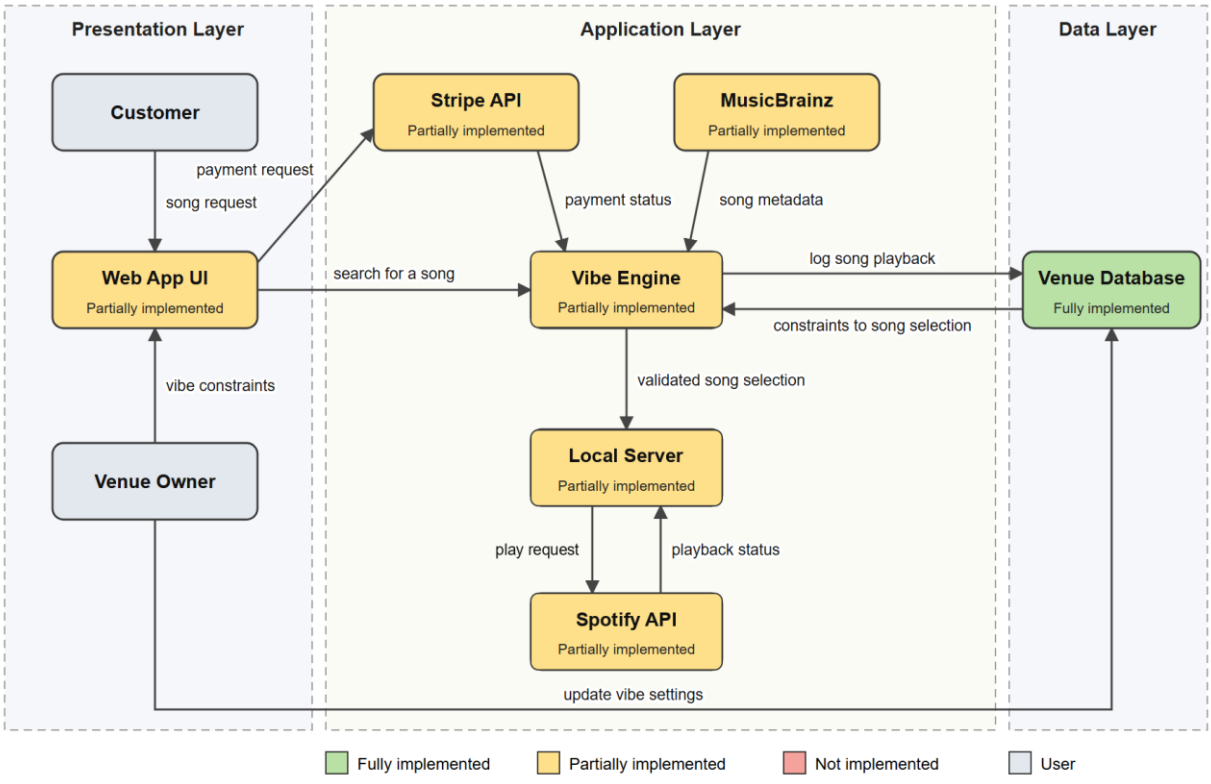


Figure 1. *RhythmiQ Major Functional Components Diagram (MFCD) with prototype status*

Subsystems and their interactions

1. **Web App UI.** The customer-facing portal and the owner/staff Business Control Center. Customers connect to a venue through its QR code, then search for and request songs; owners and staff send vibe constraints, review requests, and update vibe settings.
2. **Vibe Engine.** Receives each request from the Web App UI and checks the song's metadata against the venue's song blacklist, genre filters, explicit-content setting, and BPM range from the Venue Database. It rejects requests that break a rule, marks requests with missing song data for staff review, and records each decision with its reasons. Requests it does not reject enter the queue once staff approve them.
3. **Venue Database.** Stores user accounts, venues and their music settings, song requests and Vibe Engine decisions, the song queue, and payment records. It supplies constraints to the Vibe Engine, receives playback logs from it, and is updated directly by owner settings changes.
4. **Local Server.** Receives validated song selections from the Vibe Engine, sends a play request for each one to the playback service, and relays playback status back for logging.
5. **Playback Service.** The Spotify Web API in the prototype and SoundMachine in the RWP. It receives play requests from the Local Server, plays audio through the venue's speakers, and reports playback status back. Only SoundMachine provides the commercial licensing that the RWP requires.

External interfaces

- **User interface:** a responsive web interface in a mobile browser for customers, and a web dashboard for owners and staff.
- **Hardware interface:** the venue's existing internet-connected device and speaker system.
- **Music metadata API:** the system relies on a third-party API for song metadata. The prototype uses MusicBrainz [11] for song search, caching results locally; Soundcharts [18] was the original choice but is no longer free. No provider for genre, mood, BPM, or explicit-content data has been configured yet, so the long-term source remains under evaluation.
- **Stripe API:** the system will interact with Stripe over its API to process Pay-to-Request payments and return payment status for queue prioritization [12]. Payment and refund records are modeled in the database, but the Stripe integration is still in development.
- **Playback service:** the Local Server sends play requests to the Spotify Web API [14] in the prototype and to SoundMachine, a licensed streaming service, in the RWP [13], and receives playback status from either.

2.2 Product Functions

The main functions of RhythmiQ are summarized in Table 1, which compares the real-world product (RWP) with the CS 411W prototype. A function is *Fully implemented* when the prototype provides the RWP behavior, *Partially implemented* when it provides a reduced form, and *Eliminated* when it is excluded from the prototype. Status describes the prototype's planned scope under the team's current implementation plan and Section 3 of the team SRS; the notes record each function's progress in the team's code as of October 30, 2026.

Table 1. RWP vs. Prototype Function Comparison

Function	Real-World Product (RWP)	Prototype Status	Prototype Notes
Account Creation	Customers, staff, and owners can create user accounts.	Fully implemented	Username and password registration; new accounts receive the Customer role, and owners assign staff roles. Customers may also request songs as guests. Email verification (Section 3.1.1) is in development.
Login	Login screen and role-based authorization.	Fully implemented	Django login; each user is redirected to the dashboard for their role.
Venue QR Code and Connection	Owners generate a unique QR code for each venue; customers scan it to connect to that venue's request page.	Fully implemented	Each venue has its own public link for its QR code to open; QR code generation is in development.
Song Selection	Customers search for and select songs from a list.	Fully implemented	Title search through MusicBrainz, cached locally; results show title, artist, and a request option.
Vibe Engine	Recommends and validates songs based on the venue's music preferences.	Partially implemented	Checks the song blacklist, explicit-content setting, genre filters, and BPM range; sends requests with missing song data to staff review. No mood rules or recommendations.
Song Metadata Retrieval	Genre, mood, and explicit-content data from a commercial metadata API.	Partially implemented	MusicBrainz supplies title and artist; no genre, mood, BPM, or explicit-content provider is configured yet.
Block Songs (Queue Override)	Staff and owners prevent songs that do not fit the vibe from playing.	Fully implemented	Staff approve requests into the queue or reject them, and reorder, reprioritize, and remove queued songs.
Blacklist Song / Filter Genre	Owners block specific songs and choose which genres or moods are allowed or blocked.	Partially implemented	Song blacklist and allowed or blocked genres are enforced; set in Django admin rather than the owner dashboard. No mood filters.
Request Availability	Owners and authorized staff enable or disable song requests for the venue.	Fully implemented	Specified in Section 3.1.5; in development.
Payment Processing	Payments processed through the Stripe API.	Partially implemented	Payment and refund records are modeled; Stripe integration is in development.

Function	Real-World Product (RWP)	Prototype Status	Prototype Notes
Priority Boost	One or more paid tiers push a request higher in the queue.	Partially implemented	Staff set queue priority from −10 to 10; paid priority is in development.
View Revenue	Owners see live tracking of revenue from song requests.	Partially implemented	Owner dashboard shows request counts by status; revenue tracking is in development.
Licensed Playback	All audio plays through SoundMachine under commercial licensing.	Partially implemented	Prototype playback uses the Spotify Web API, which is not licensed for commercial use; the integration is in development. SoundMachine is used in the RWP only.
User Profile	Profiles with name, recommendations, and balance.	Partially implemented	Customers see their own requests and statuses at each venue; no profile details, recommendations, or balance.
Full Catalog Browsing, Song Details	Full music list and detailed song information.	Eliminated	Not implemented in the prototype.
Multi-Location Analytics, B2B Licensing, Loyalty Points	Cross-location dashboard, venue-management licensing, and customer rewards.	Eliminated	Future expansion only.

2.3 User Characteristics

RhythmiQ has three user roles. Each role includes the capabilities of the role before it: staff can do everything a customer can, and owners can do everything staff can. None requires technical training.

Table 2. User Roles

Role	Expected Expertise	Usage Scenario	Constraints
Customer	Everyday smartphone and mobile-browser use. No training and no app installation.	Scans the venue's QR code or opens its link, searches for a song, and requests it, with or without an account; optionally pays to prioritize it.	Short interaction window on a small screen; limited to five pending requests per venue and to songs that satisfy the venue's constraints.
Staff	Basic web-dashboard use. Little or no technical background.	Reviews requests the Vibe Engine flags, approves or rejects them, and manages the live queue by reordering, reprioritizing, and removing songs and marking them played.	Busy with service during rushes, so actions must be quick; can manage only the venues they are assigned to.
Owner	Basic web-dashboard use; understands the venue's intended atmosphere. Not necessarily technical.	Sets music rules (explicit content, genre filters, BPM range, and song blacklist), assigns staff roles, and monitors request activity; in the RWP, also tracks revenue from song requests.	Limited time, so configuration must be quick. Holds final authority over venue settings.

2.4 Constraints

- **Legal:** All audio playback in the RWP must be commercially licensed, which SoundMachine provides [13]; personal streaming terms, such as Spotify's, do not permit commercial use [10]. Because the other music providers the team contacted have not yet responded, the prototype's playback is being built on the Spotify Web API [14]. The prototype may therefore be used only for development and classroom demonstration, not in an operating venue, and must move to a licensed service before any commercial deployment.
- **Payments and privacy:** Card data must not be stored locally and must be handled by Stripe. User data collection must be kept to a minimum.
- **Hardware:** The system must not require proprietary hardware and must run on devices the venue already owns.
- **Cost:** The subscription must stay viable for a small business budget.
- **Technology:** The system is implemented in Python with Django and PostgreSQL, runs in Docker containers, and must be usable in a mobile browser without a native app. It currently runs only on development machines; GitHub Pages cannot host the back end, so a server host is required and has not yet been selected.
- **External API limits:** Request limits and pricing of third-party APIs must be handled safely. The prototype caches MusicBrainz search results locally and handles timeouts without failing, and the same approach applies to the Spotify Web API.

- **Prototype:** The prototype must be completed within the CS 411W schedule, which is why some RWP functions are reduced or eliminated (Table 1).

2.5 Assumptions and Dependencies

Dependencies

- **Music metadata API:** MusicBrainz for song search in the prototype [11]; a provider for genre, mood, BPM, and explicit-content data, such as Soundcharts [18] or another commercial source, for the RWP. Changes in pricing or policy affect availability.
- **Playback service:** the Spotify Web API [14] for the prototype, whose playback endpoints require a Spotify Premium account; SoundMachine [13] for the RWP, which must be available and hold the commercial licenses needed for playback.
- **Stripe** is planned for Pay-to-Request payments [12].
- **Software stack:** Django, psycopg (PostgreSQL driver), python-dotenv, and Requests; PostgreSQL; Docker Compose; and GitHub Actions for continuous integration (migration checks and tests).

Assumptions

- The venue has a reliable internet connection and a device connected to its speaker system.
- Customers have a smartphone with a camera and a current mobile browser.
- In the RWP, the venue holds the SoundMachine subscription needed to play licensed audio; for prototype demonstrations, a Spotify Premium account is signed in on the playback device.
- An administrator assigns each venue's owner and staff, and owners configure venue constraints and the venue QR code before customers begin making requests.