# Software Requirements Specification

for

# Rhythmiq

"Your Vibes, Managed"

Sections 1 and 2

Prepared by Andrew Behre
Team Blue

CS 411 Lab 2

October 18, 2026

# Contents

**1 Introduction** **2**

1.1 Purpose . . . . . 2

1.2 Scope . . . . . 2

1.3 Definitions, Acronyms, and Abbreviations . . . . . 3

1.4 References . . . . . 4

1.5 Overview . . . . . 5

**2 Overall Description** **5**

2.1 Product Perspective . . . . . 5

2.2 Product Functions . . . . . 6

2.3 User Characteristics . . . . . 7

2.4 Constraints . . . . . 8

2.5 Assumptions and Dependencies . . . . . 8

# 1. Introduction

## 1.1 Purpose

This Software Requirements Specification (SRS) defines what Team Blue's developers must build for Rhythmiq, a web-based jukebox platform for small cafes. It follows IEEE Std 830-1998 [1] and serves as a technical reference for developers, testers, and maintainers during implementation. It does not repeat the broader product case made in Lab 1 [2]. Sections 1 and 2 give context, scope, and a product description; Section 3 will state the individual requirements.

## 1.2 Scope

Rhythmiq is a software-only web application that lets cafe customers request songs from their own phones while the venue owner keeps control of what plays. It aims to remove staff-handled song requests, give small venues an affordable way to use music for customer engagement [3, 4], and return a majority share of request revenue to the venue: 67%, compared with roughly 33–39% reported for TouchTunes [5] and 0% under Rockbot's subscriber terms [6]. This section abbreviates Lab 1, Section 1 [2].

Rhythmiq will:

- Give customers QR-code access to a venue-specific web portal with no app download and no account.
- Let customers search for songs and submit requests that are filtered against the venue's vibe constraints.
- Validate every request in the Vibe Engine before it reaches the queue.
- Give owners and staff a Business Control Center to set constraints, manage the live queue, and view revenue.
- Offer an optional pay-to-request priority boost processed through Stripe.
- Play queued songs through a playback provider on equipment the venue already owns.
- Keep each venue's settings, queue, and data separate from every other venue's.

Rhythmiq will not:

- Require proprietary hardware or a customer-installed app.
- Generate custom songs for customers.
- Store payment card data.
- Let customers bypass owner-defined vibe constraints, or let staff change venue settings.
- Claim commercial music licensing in the prototype. The Spotify integration is a proof of concept only.

## 1.3 Definitions, Acronyms, and Abbreviations

This is the team glossary from Lab 1, updated for the current architecture. Entries for Soundcharts and SoundMachine were replaced by MusicBrainz, Spotify Web API, and Adapter Function.

**Adapter Function**
A function written by the team that wraps one external API call, so the rest of the code never calls a provider directly and a provider can be replaced by changing only the adapter.

**API (Application Programming Interface)**
A set of protocols and tools that allow different software applications to communicate with each other.

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

**BMI (Broadcast Music, Inc.)**
A major U.S. performing rights organization that manages commercial music licensing for songwriters and publishers.

**Business Control Center**
Rhythmiq's dashboard that lets owners and staff manage song requests, set filters, track revenue, and control the atmosphere.

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

**Django**
A Python web framework used to build Rhythmiq's web application.

**Hardware Agnostic**
Software that runs on existing devices without requiring specific or proprietary hardware.

**Metadata**
Information, such as author, creation date, or file size, that describes a data point or data set [7]. For a song, this includes details such as genre, tempo, and mood.

**MFCD (Major Functional Components Diagram)**
A diagram showing the main software components of a system and how they interact, as defined in CS 410.

**MusicBrainz**
An open music encyclopedia and its web API, used by Rhythmiq to look up song metadata.

**NRG (National Research Group)**
A global insights and strategy firm specializing in technology and entertainment.

**Pay-to-Request**
A feature that lets customers pay to have their song moved higher in the queue.

**PoC (Proof of Concept)**
A limited implementation that shows an approach is feasible without being production-ready.

**PostgreSQL**
An open-source database system that stores Rhythmiq's venue, song, and transaction data.

**QR Code (Quick Response Code)**
A scannable two-dimensional barcode that opens a website or application when read by a smartphone camera.

**Revenue Stream**
A source of income for a business; here, the owner's 67% share of pay-to-request transactions.

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

**Spotify Web API**
Spotify's API, used in the prototype as a proof-of-concept playback provider behind an Adapter Function.

**SRS (Software Requirements Specification)**
A document that states the requirements of a software system, structured per IEEE Std 830-1998.

**Stripe API**
A third-party payment service that Rhythmiq uses to process customer payments without storing card data itself.

**Vibe** The mood of a place, situation, or person and the way it makes people feel [8].

**Vibe Engine**
Rhythmiq's in-house song validation algorithm, which evaluates customer requests against a venue's configured constraints before they enter 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

Entries follow IEEE citation format and are numbered in order of first citation. Entry [2] is the formal reference to Lab 1.

## References

[1] "IEEE recommended practice for software requirements specifications," IEEE Std 830-1998, Jun. 1998.

[2] Team Blue, "Rhythmiq: Lab 1 outline," CS 411, course document, Aug. 30, 2026.

[3] Broadcast Music, Inc., "New BMI/NRG study confirms consumers eat, drink, and spend more when listening to music," BMI, Apr.

24, 2023. [Online]. Available: https://www.bmi.com/news/entry/new-bmi-nrg-study-confirms-consumers-eat-drink-spend-more-when-listening

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

[5] r/BarOwners, "Percent taken out for TouchTunes," Reddit, n.d. [Online]. Available: https://www.reddit.com/r/BarOwners/comments/1dkmvft/percent_taken_out_for_touchtunes/

[6] Rockbot, "Subscriber terms," n.d. [Online]. Available: https://rockbot.com/subscriber-terms/

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

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

[9] TouchTunes, "About us: Our story," n.d. [Online]. Available: https://www.touchtunes.com/about/our-story

[10] Rockbot, "Smarter music for your business," n.d. [Online]. Available: https://rockbot.com/go

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

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

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

### 1.5 Overview

Section 2 describes Rhythmiq as a whole: its place among external services and the MFCD (2.1), its features and their prototype status (2.2), its users (2.3), its constraints (2.4), and its assumptions and dependencies (2.5). Section 3 will state the specific, testable requirements, organized by feature. Terms are used as defined in Section 1.3, and numbered citations refer to Section 1.4.

## 2. Overall Description

### 2.1 Product Perspective

Rhythmiq is a new, self-contained web application. It replaces hardware-based jukeboxes such as TouchTunes [9] and platforms such as Rockbot [10] that depend on dedicated machines or apps. Customers and owners reach it through a mobile-first web UI, and it runs on devices a venue already owns. The Vibe Engine is the core: it checks each request against the venue's constraints before the request is queued.

**External interfaces.** Rhythmiq calls three third-party web APIs: MusicBrainz for song metadata [11], the Spotify Web API for playback in the prototype [12], and Stripe for payments [13]. The team writes one adapter function for each external API call, and the rest of the code calls only those functions. A provider is replaced by rewriting its adapter, which matters because the metadata and playback providers have already changed since Lab 1 [2]. Spotify is a proof-of-concept playback provider only; a commercially licensed provider remains a production requirement.

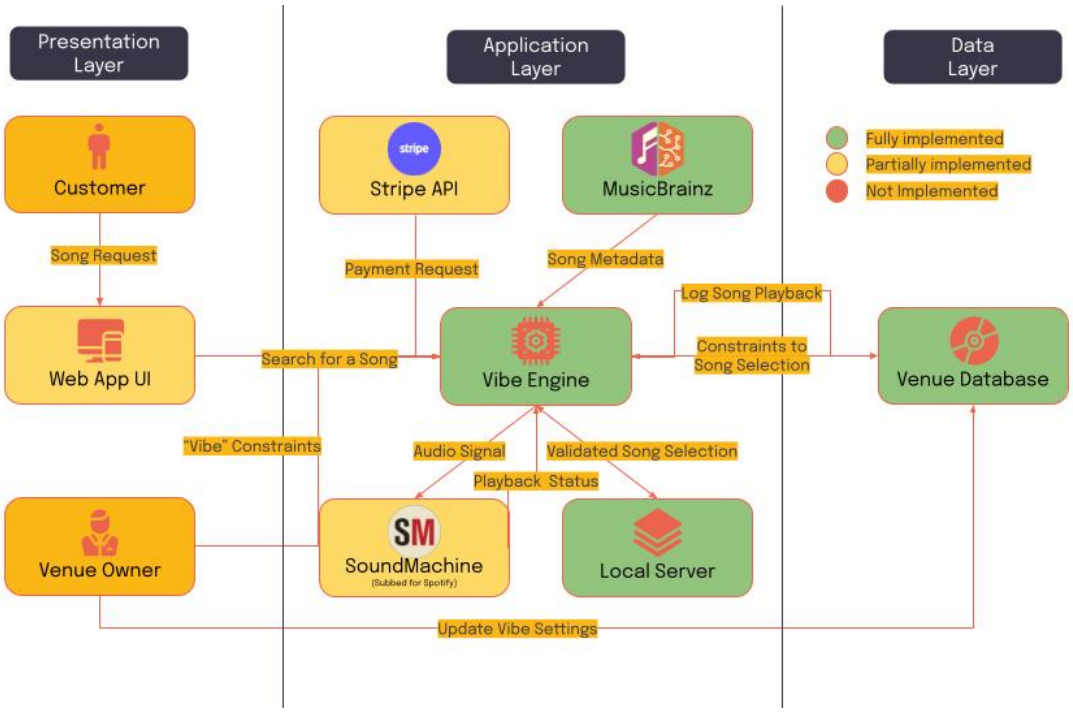



Figure 1: Rhythmiq major functional components diagram (MFCD). Arrows point in the direction of the initiating call.

Figure 1 shows how the subsystems interact:

- **Web App UI:** responsive front end for customer song search and requests, and for the Business Control Center.
- **Vibe Engine:** validates requests against venue constraints, maintains the queue, logs each song played, and is the only component that calls the adapters.
- **Adapters:** team-written functions that wrap MusicBrainz (metadata lookup), Spotify (playback), and Stripe (payment requests).
- **Local Server:** receives validated song selections at the venue.
- **Venue Database:** PostgreSQL store for each venue's vibe constraints, queue state, play log, and transaction and revenue data.

## 2.2 Product Functions

The prototype fully implements five of Rhythmiq's eight features and partially implements three; none is eliminated. Table 1 compares the real-world product (RWP) with the prototype.

Table 1: Rhythmiq features and prototype status.

| **Feature** | **What it does** | **Prototype status** | **Prototype notes** |
|---|---|---|---|
| QR Code Access | Opens a venue-specific portal in a mobile browser from a table QR code, with no app and no account | Fully implemented | None |
| Song Browsing and Requests | Lets customers search songs by genre and vibe and submit requests, filtered by the venue's constraints | Fully implemented | Metadata comes from MusicBrainz, so filters use the attributes it provides |
| Vibe Engine | Checks each request against the venue's constraints and rejects non-matching songs before they are queued | Fully implemented | Constraint types are limited to attributes available from the metadata provider |
| Dynamic Vibe Moderation | Enforces the owner's rules automatically on every request, with no staff review | Fully implemented | None |
| Business Control Center | Gives owners and staff constraint settings, allow and block lists, live queue management, revenue views, and role-based access | Partially implemented | Delivered through the partial Web App UI; revenue views depend on the partial payment flow |
| Pay to Request | Offers an optional paid priority boost, with 67% of revenue going to the venue | Partially implemented | Stripe integration only; no bank account is attached to the prototype |
| Song Playback | Plays queued songs through the venue's existing speakers | Partially implemented | Spotify Web API stands in as a proof of concept behind the playback adapter; a commercially licensed provider is not integrated |
| Hardware-Agnostic Deployment | Runs on phones, tablets, or laptops the venue already owns | Fully implemented | None |

## 2.3 User Characteristics

Rhythmiq has three user roles, and none requires technical expertise. Each is a non-specialist using the system in short sessions on a phone, tablet, or counter device.

Table 2: Rhythmiq user roles.

| Role | Expected expertise | Usage scenario | Constraints |
|---|---|---|---|
| Customer | Everyday smartphone use; no knowledge of the system | Scans the table QR code, searches for a song, requests it, and may pay for a priority boost | No app download and no login beyond a simple session; minimal personal data; mobile browser only |
| Staff (for example, a barista) | Basic device use; trained informally by the owner | Reviews the live queue during a shift and reorders, removes, or blocks requests | Cannot change venue settings; sessions are short and often interrupted during rushes |
| Owner | Small-business owner with basic device use; no development or music-licensing background | Creates the venue account, sets genre, mood, tempo, and explicit-content rules, adjusts them by time of day, and reviews revenue | Cannot bypass licensing rules; configures rules in occasional sessions during slow periods |

## 2.4 Constraints

Four sets of constraints limit the design: technical, legal, environmental, and project.

- **Technical:** The backend is written in Python with Django, Django REST Framework, and Django Channels, stores data in PostgreSQL, and is packaged with Docker. The front end must be mobile-first and responsive. All external API calls must go through adapter functions. MusicBrainz limits request rates, so the metadata adapter must throttle and cache lookups. MusicBrainz is a metadata catalog and does not supply audio-analysis attributes such as tempo or mood, so constraints on those attributes depend on what the metadata provider can supply.
- **Legal:** Production playback must be covered by commercial licensing from ASCAP, BMI, and SESAC. Spotify's developer terms restrict commercial use, so the prototype's Spotify integration is a proof of concept and must not be deployed in a paying venue. Rhythmiq must not store card data and must collect only minimal personal data from customers.
- **Environmental:** Each venue needs an internet connection and a device wired or paired to its existing speakers. Customers need a smartphone with a camera and a mobile browser.
- **Project:** The prototype has no bank account, so payments cannot be completed end to end, and all work must fit the team's planned sprints.

## 2.5 Assumptions and Dependencies

Rhythmiq depends on three external APIs and assumes their terms and availability hold long enough to finish the prototype. Both providers have already changed since Lab 1, which is why adapter functions are required.

Dependencies:

- **MusicBrainz API:** song search and metadata.
- **Spotify Web API:** proof-of-concept playback, which needs developer credentials and a Spotify account permitted to control playback.
- **Stripe API and Python library:** payment requests; card data stays with Stripe.
- **Software stack:** Python, Django, Django REST Framework, Django Channels, the Requests library, PostgreSQL, and Docker, with GitHub Actions for testing and deployment.

Assumptions:

- Each venue has an internet connection and one device connected to its speakers.
- Customers have a smartphone with a camera and a mobile browser.
- Songs customers request have usable metadata in MusicBrainz.
- Each adapter keeps a stable interface, so replacing a provider changes only that adapter and not the Vibe Engine.