

# Change History

| Version | Notes |
| --- | --- |
| 0.1 | * First Iteration |
| 0.2 | * Added diagrams to section 3 * Added diagram descriptions to section 3 * Reformatted use cases * Reformatted System Specification Summary and Configuration Requirements * Completed walk-throughs * Reformatted information model * Added new Network Diagram for section 5.1 * Completed section 6 * Completed section 8 * Reformatted Quality Attributes * Completed Phasing, Roadmap and Implementation Plan |

# 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. |
| FIFO | A process for ordering data, where the initial data that is added to a queue will be the data that is taken first from the queue. |

Contents

\*\*We are aware some links do not work, we will fix in next iteration

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

[**Glossary**](#_heading=h.30j0zll)1

[**1 Introduction**](#_heading=h.1fob9te)5

[1.1 Purpose](#_heading=h.3znysh7) 5

[1.2 Audience](#_heading=h.2et92p0) 5

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

[**2 Business Requirements and Goals**](#_heading=h.1t3h5sf)6

[2.1 Business Drivers](#_heading=h.4d34og8) 6

[2.2 Solution Requirements](#_heading=h.2s8eyo1) 6

[2.3 High Level Metrics for Success](#_heading=h.17dp8vu) 6

[2.4 Architecture Boundary](#_heading=h.3rdcrjn) 6

[**3. Business Model**](#_heading=h.26in1rg)7

[3.1 Organisational Context](#_heading=h.4h042r0) 57

3.2 Current Situation 57

3.3 Solution Definition 57

3.4 Actors 57

3.5 Interactions of Actors 57

[3.6 Use Cases](#_heading=h.lnxbz9) 10

[3.6.1 Logging In to a Virtual Jukebox Account](#_heading=h.35nkun2) 10

[3.6.2 Hosting a Virtual Jukebox Session](#_heading=h.1ksv4uv) 12

[3.6.3 Joining a Virtual Jukebox Session](#_heading=h.44sinio) 14

[3.6.4 Adding a Song to the Song Queue](#_heading=h.2jxsxqh) 15

[3.6.5 Voting for a Song in the Queue](#_heading=h.z337ya) 17

[3.6.6 Using the Chat Room](#_heading=h.3j2qqm3) 19

[3.6.7 Closing a Virtual Jukebox Session](#_heading=h.1y810tw) 21

[**4 Solution Architecture Model**](#_heading=h.4i7ojhp)23

[4.1 Solution Components](#_heading=h.4h042r0) 57

[4.1.1 Internal Components](#_heading=h.2xcytpi) 23

[4.1.2 External Components](#_heading=h.1ci93xb) 24

[4.2 Integration Specification](#_heading=h.4h042r0) 57

4.3 Walk-Throughs 57

[4.3.1 Logging In to a Virtual Jukebox Account](#_heading=h.3whwml4) 27

[4.3.2 Hosting a Virtual Jukebox Session](#_heading=h.2bn6wsx) 28

[4.3.3 Joining a Virtual Jukebox Session](#_heading=h.qsh70q) 29

[4.3.4 Adding a Song to the Song Queue](#_heading=h.3as4poj) 30

[4.3.5 Voting for a Song in the Queue](#_heading=h.1pxezwc) 31

[4.3.6 Using the Chat Room](#_heading=h.49x2ik5) 31

[4.3.7 Closing a Virtual Jukebox Session](#_heading=h.2p2csry) 32

[4.4 Information Model](#_heading=h.147n2zr) 33

[4.4.1 Virtual Jukebox Database](#_heading=h.3o7alnk) 33

[4.4.2 Redis cache](#_heading=h.23ckvvd) 34

[**5 Detailed Physical Architecture**](#_heading=h.ihv636)35

[5.1 Network Diagram](#_heading=h.4h042r0) 57

5.2 System Specification Summary 57

[5.3 Configuration Requirements](#_heading=h.32hioqz) 36

[**6 Solution Impact**](#_heading=h.41mghml)37

[6.1 Impact on Processes](#_heading=h.4h042r0) 57

[6.2 Impact on People](#_heading=h.2grqrue) 37

[6.3 Technical Impact](#_heading=h.vx1227) 37

[**7 Quality Attributes**](#_heading=h.1v1yuxt)38

[7.1 Reliability](#_heading=h.4f1mdlm) 38

[7.2.1 Fault Tolerance](#_heading=h.2u6wntf) 38

[7.2.2 Fault Avoidance](#_heading=h.19c6y18) 38

[7.2.3 Capacity](#_heading=h.3tbugp1) 38

[7.2 Availability](#_heading=h.28h4qwu) 39

[7.2.1 Malicious Attack](#_heading=h.nmf14n) 39

[7.2.2 Unresponsiveness](#_heading=h.37m2jsg) 39

[7.3 Serviceability](#_heading=h.1mrcu09) 39

[7.3.1 Documentation](#_heading=h.46r0co2) 39

[7.3.2 Coding Standards and Best Practices](#_heading=h.2lwamvv) 39

[7.4 Best Practice Commentary](#_heading=h.4h042r0) 57

[7.4.1 Deliverable Scheduling](#_heading=h.111kx3o) 40

[7.5 Architecture Commentary](#_heading=h.4h042r0) 57

[7.5.1 Monetisation](#_heading=h.3l18frh) 40

[**8 Success Criteria and Test Plan**](#_heading=h.4h042r0) 57

8.1 Use Case Validation 57

[8.2 Validation of Solution metrics](#_heading=h.4k668n3) 42

[8.3 Test Plan](#_heading=h.2zbgiuw) 43

[**9. Phasing, Roadmap, and Implementation Plan**](#_heading=h.4bvk7pj)53

[9.1 Technical Assumptions](#_heading=h.2r0uhxc) 53

[9.1.1 User Interfaces](#_heading=h.1664s55) 53

[9.1.2 Hardware Interfaces](#_heading=h.3q5sasy) 53

[9.1.4 Communications Interfaces](#_heading=h.25b2l0r) 53

[9.2 Solution Phases and Technical Milestones](#_heading=h.4h042r0) 57

[9.2.1 Phase 1: Package 3](#_heading=h.kgcv8k) 53

[9.2.2 Phase 2: Package 4](#_heading=h.34g0dwd) 54

[9.2.3 Phase 3: Package 5](#_heading=h.1opuj5n) 54

[9.2.4 Phase 4: Package 6](#_heading=h.1jlao46) 54

[9.3 Technical Work Breakdown Structure](#_heading=h.4h042r0) 57

9.4 Implementation Instructions 57

[9.4.1 Coding Standards and Guidelines](#_heading=h.43ky6rz) 55

[9.4.2 Documentation Standards](#_heading=h.2iq8gzs) 56

[9.4.3 Repository Standards](#_heading=h.xvir7l) 56

[**10 Appendix**](#_heading=h.3hv69ve)57

[**11 References**](#_heading=h.1x0gk37)57

# 1 Introduction

## 1.1 Purpose

This report provides a detailed design in regards to the implementation of the application–featuring use cases, walkthroughs, architectural diagrams, an ER diagram, test plans and component descriptions.

## 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 Business Requirements and Goals

## 2.1 Business Drivers

A gap in the market has been identified, with guests in attendance of venues currently having very little control over the music that is being played. Businesses are commonly playing music from predefined playlists in a random order or broadcasting a radio station with no control over what is being played. If guests of a venue would prefer to listen to different music, the only option is to find the person responsible for the music and to request that the song be skipped, or a different song be played. The Virtual Jukebox application aims to solve this issue by providing guests with the ability to control the music that is being played at a venue.

## 2.2 Solution Requirements

The Virtual Jukebox application will allow hosts to create a Virtual Jukebox account and login using their credentials. The application will also allow guests to generate and use a guest account to join a session, which will be generated with a random username. Guests will be able to connect to a Virtual Jukebox Session either by scanning a session’s QR code (or directly visiting the link), or by selecting a session on the Virtual Jukebox map. Each Virtual Jukebox session will have a playlist acting as a pool of songs which users can choose from to add to the song queue, which will be imported from the host’s Spotify account.

The Virtual Jukebox application will utilise a song queue which is viewable by all guests as well as the host. The queue will control the ordering of song playback on the host device. The user-queued songs will be ordered in a first-in-first-out (FIFO) manner by default, but songs with votes will be prioritised in playback order. Songs with the same number of votes will remain in a FIFO order. When no songs are queued by users, the host device will randomly play songs in the host defined playlist. Every session will allow all users to exchange instant messages through the live chat room, with these messages being filtered for profanity.

## 2.3 High Level Metrics for Success

For the project to be considered a success, the Virtual Jukebox application must allow hosts to register and sign into their own personal accounts and import a playlist from their Spotify account. Hosts must be able to create a session and allow them to specify the location of their session on a map, as well as being provided with a QR code containing a direct link to the session. Guests must be able to connect to this session by selecting the session on the map, or by scanning the QR code. When guests have joined a session, they must have the ability to add songs to the music queue, be able to vote for songs in the current music queue, with the queue being ordered by number of votes placed for each song. For the host, the Virtual Jukebox application must play the song through the Spotify SDK with the highest number of votes when the currently playing song has finished.

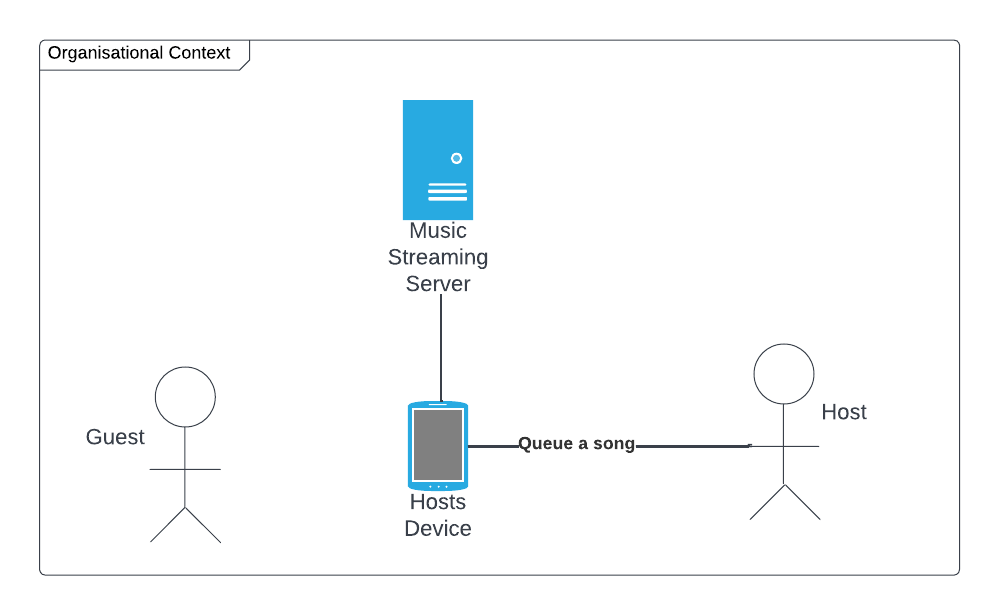
## 2.4 Architecture Boundary

“What is in-scope, and what is not in-scope.” \*\* I am not sure what is being asked here.

# 3. Business Model

## 3.1 Organisational Context

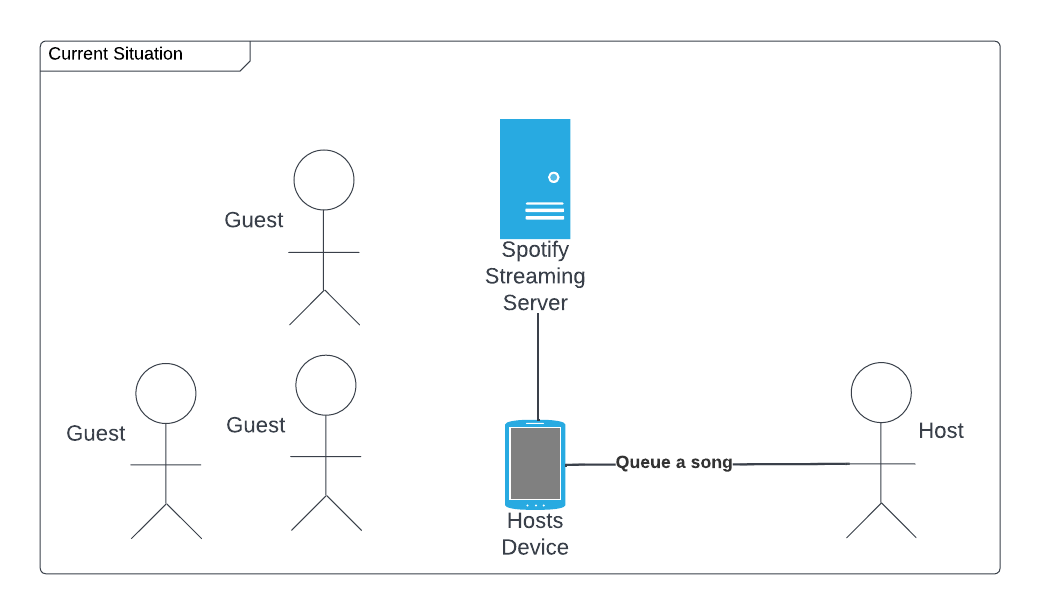
The current state of music streaming exists primarily for an individual. Music can be instantly streamed for an individual through various music streaming services such as Spotify, which can instantly playback any song out of a database of millions. When it comes to music streaming for a group of people, options are extremely limited. The current situation of group music streaming for gatherings, social events, or businesses is that a single device will control the music playback and queue. Users who do not have access to this device will not have any control over the played music.



The above diagram is a representation of this situation. The guest actor in this instance does not have any control over the music being played, and they cannot directly queue a song. The host actor is able to queue songs and control music playback, however they are the only user with these capabilities.

## 3.2 Current Situation

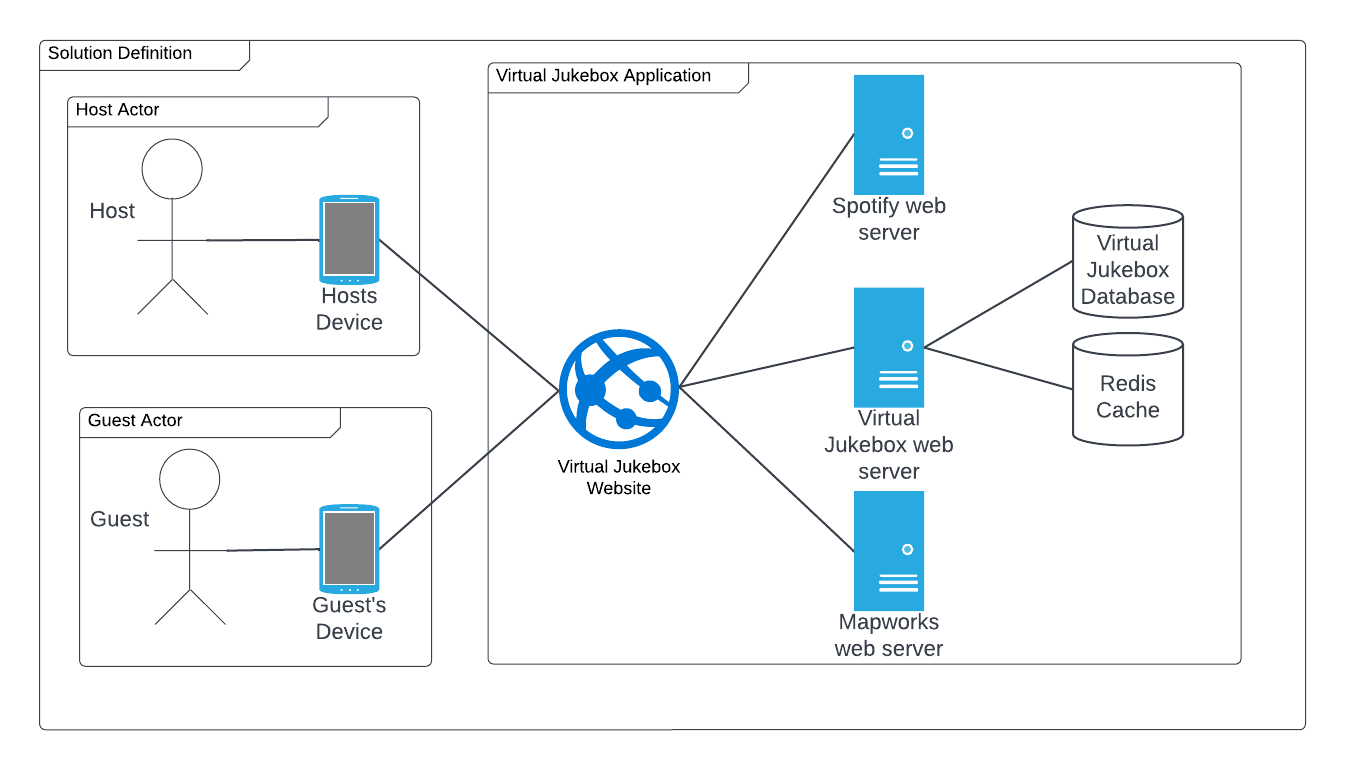
A more specific example of the above situation would be house parties, as depicted in the diagram below.



In this example, the host actor is the host of the house party, and they have direct control over the music queue and playback. A house party may feature many guest actors, however, they have no direct control over the music being played, and cannot modify the song queue. The guest actors can interact with the host actor to request songs, but ultimately only the host has control over the music being played.

## 3.3 Solution Definition

The main issue derived from the current situation is the fact that only one actor has complete control over the music queue and playback. Our solution definition aims to give guest actors control over the music being played through the use of a “Virtual Jukebox Session”.



As represented in the above diagram, both host users and guest users interact with the Virtual Jukebox application. The Virtual Jukebox application allows a host actor to create a Virtual Jukebox Session, and the details of this session will be stored inside of the Redis cache. Guest actors can join this session through the Virtual Jukebox Map, a QR code or a URL. Once actors are connected to a Virtual Jukebox Session, they can request songs from the Virtual Jukebox Session’s playlist to be added to the queue, and vote on songs currently in the queue. This allows all actors, both guests and hosts, to have control over the music queue.

## 3.4 Actors

| **Actor** | **Description** | **Current use of the System** | **Expected use of the New Solution** |
| --- | --- | --- | --- |
| **Virtual Jukebox Application** | The application which acts as a Jukebox. | Does not exist | The Virtual Jukebox application is responsible for providing users the ability to set up and join Virtual Jukebox sessions, managing these sessions’ song queues, and in turn, handling music playback via Spotify |
| **Hosts** | A type of user responsible for | Hosts will control playback through their device, and will physically pass their device to guests users for the guest users to queue songs. Hosts can additionally queue songs on their own device. | Hosts may create a Virtual Jukebox session, and can control the music playback. Hosts can vote for songs in the song queue, and add songs from the session playlist into the song queue. |
| **Guests** | A type of user who does not have control over the song queue | Guests will not have any direct control over the song queue unless they are able to interact with a host user’s device, when they can then queue songs on. | Guests can create and log into an account, join a session, queue songs in a session, send messages, and vote on the current song queue. Guests are not required to log into an account to join sessions. |

## 3.5 Interactions of Actors

## 

## 3.6 Use Cases

### 3.6.1 Logging In to a Virtual Jukebox Account

| **Goal** | Logging In to a Virtual Jukebox Account |
| --- | --- |
| **Code** | VJ-UC-01 |
| **Actors** | Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The host enters their username and password of their Virtual Jukebox Account 2. The host clicks the submit button 3. The system checks that the hashed password is identical to the one stored in the database 4. The system notifies the user that they have successfully logged in 5. The system closes the login screen and opens the home page.   **Extensions:**  1A - The host has not created a Virtual Jukebox Account   1. The user will select the make an account button 2. The user will enter a username, email and password to assign to the new Virtual Jukebox Account 3. The user will authenticate their Virtual Jukebox Account to an existing Spotify account 4. The use case resumes at step 4   1B - The host does not remember the password for an existing Virtual Jukebox Account   1. The user will select the reset password button 2. The user will enter the email of their Virtual Jukebox Account 3. The system will send an email containing a password reset link to the provided email. 4. The use case resumes at step 1   3A - The system cannot find the account in the database   1. The system notifies the user that there is an incorrect username or password 2. The use case resumes at step 1   3B - The system finds the hashed password provided and the one stored are different   1. The system notifies the user that there is an incorrect username or password 2. The use case resumes at step 1 |
| **Frequency** | The host selects the “login” button |
| **Pre-processing** | The host is connected to the internet |
| **Post-processing** | The host is logged into the Virtual Jukebox Account |
| **Included use case** | None |
| **Extended use case** | None |

### 3.6.2 Hosting a Virtual Jukebox Session

| **Goal** | To allow hosts to create a Virtual Jukebox Session |
| --- | --- |
| **Code** | VJ-UC-02 |
| **Actors** | Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The system checks the database to make sure the current host user does not have any active sessions in the database 2. The system retrieves the current location of the host device 3. The system prompts the user to select a location on the Virtual Jukebox Map to host the Virtual Jukebox Session. The initial location of the map is the location retrieved at step 1. 4. The system prompts the user to enter a name for their Virtual Jukebox Session 5. The system prompts the user to enter a description for their Virtual Jukebox Session 6. The system interacts with the Spotify SDK to fetch all of the host’s Spotify Playlists 7. The system prompts the user to select a playlist for their Virtual Jukebox Session 8. The system adds the Virtual Jukebox Session information to the database 9. The system closes the create session popup and opens the Virtual Jukebox Session page   **Extensions:**  1A - The host’s authenticated Spotify account does not have Spotify Premium   1. The system notifies the user they cannot create a session without Spotify Premium 2. The use case ends   1B - The host has an active session in the database   1. The system notifies the user they have an active session in progress. The system asks the user if they want to close that session and start a new one 2. The system closes the Virtual Jukebox Session in the database 3. The use case resumes at step 2   2A - The host’s device does not have location services enabled   1. The use case resumes at step 3   6A - The host’s connected Spotify account does not contain any playlists   1. The system notifies the user that they have no active playlists and asks the user to create a playlist inside of Spotify 2. The use case ends |
| **Frequency** | The host selects the start a session button |
| **Pre-processing** | The actor is connected to the internet and logged into their account |
| **Post-processing** | A Virtual Jukebox Session is created and added to the database |
| **Included use case** | VJ-UC-07 |
| **Extended use case** | VJ-UC-01 |

### 3.6.3 Joining a Virtual Jukebox Session

| **Goal** | To allow guests to join a Virtual Jukebox Session |
| --- | --- |
| **Code** | VJ-UC-03 |
| **Actors** | Guests, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The system will display the title and description of the Virtual Jukebox Session to the guest 2. The guest will click the join button to join the Virtual Jukebox Session 3. The system will connect the user to the Virtual Jukebox Session   **Extensions:**  2A) TheVirtual Jukebox Session is out of range of the guest   1. The system notifies the guest that they are out of range and cannot join the Virtual Jukebox Session 2. The use case resumes at step 1   2B) TheVirtual Jukebox Session is passworded   1. The guest will be prompted to enter the password for the Virtual Jukebox Session 2. The system checks that the hashed password is identical to the one stored in the database 3. If the password is correct, the use case resumes at step 3. 4. If the password is incorrect, the use case resumes at step 2A.1 |
| **Frequency** | The guest selects a Virtual Jukebox Session from the Virtual Jukebox Map or scans a room’s QR code |
| **Pre-processing** | The guest is connected to the internet |
| **Post-processing** | The guest is connected to the session and their id is added to the session’s database table |
| **Included use case** | None |
| **Extended use case** | VJ-UC-02 |

### 3.6.4 Adding a Song to the Song Queue

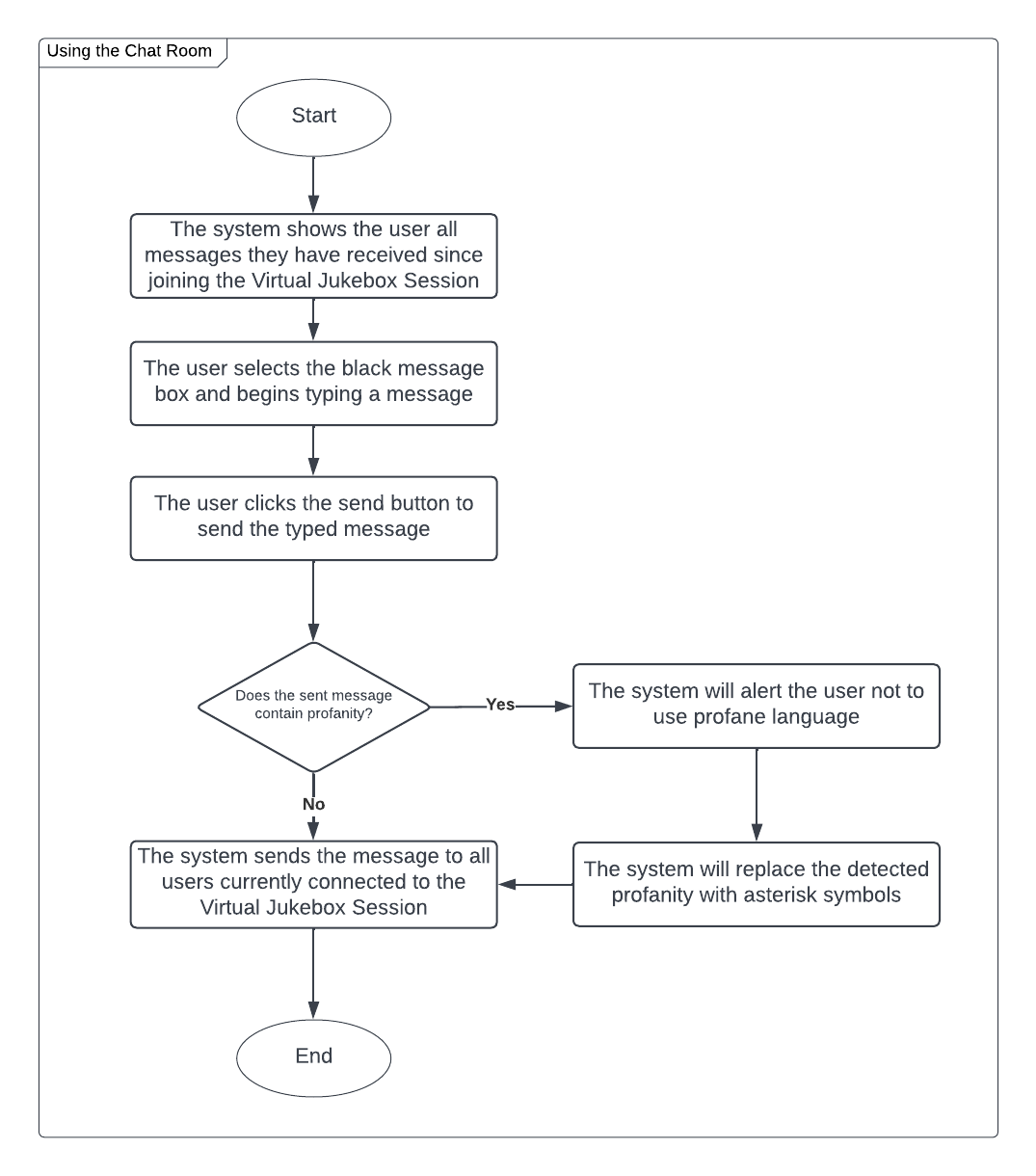
| **Goal** | To allow hosts and guests to add songs from a Virtual Jukebox Session’s playlist to the session’s song queue. |
| --- | --- |
| **Code** | VJ-UC-04 |
| **Actors** | Guests, Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The system shows the guest or host all songs currently in the Virtual Jukebox Session’s playlist 2. The user selects a song from the playlist to add to the queue 3. The system checks the number of credits the user has in the database. 4. The system decreases the number of credits the user has by one 5. The system adds the selected song to the end of the Virtual Jukebox Session’s song queue   **Extensions:**  2A) The selected song is already in the song queue   1. The system notifies the user that the current song is already in the queue 2. The use case resumes at step 1   3A) The guest or user does not have any credits   1. The system notifies the user that they do not currently have any credits to add songs to the song queue 2. The use case resumes at step 1 |
| **Frequency** | A guest or host clicks the view playlist button |
| **Pre-processing** | The guest or host is connected to the internet, and is currently connected to an active Virtual Jukebox Session |
| **Post-processing** | A song is added to the Virtual Jukebox Session’s queue |
| **Included use case** | None |
| **Extended use case** | VJ-UC-02, VJ-UC-03 |

### 3.6.5 Voting for a Song in the Queue

| **Goal** | To allow hosts and guests to upvote songs currently in a Virtual Jukebox Sessions’s song queue. |
| --- | --- |
| **Code** | VJ-UC-05 |
| **Actors** | Guests, Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The system shows the guest or host all songs currently in the Virtual Jukebox Session’s song queue 2. The user selects a song from the list to upvote 3. The system will add the vote to the cache 4. The system will reorder the current song queue by the total number of votes on each song 5. The system will push the updated song queue to all connected users.   **Extensions:**  1A) There are no songs currently in the song queue   1. The system will notify the user there are currently no songs in the song queue 2. The use case ends   2A) The user has already upvoted the selected song   1. The system will remove the vote from the cache 2. The use case resumes at step 4 |
| **Frequency** | A guest or host clicks the view queue button |
| **Pre-processing** | The guest or host is connected to the internet, and is currently connected to an active Virtual Jukebox Session |
| **Post-processing** | The number of votes on a song in a Virtual Jukebox Session’s song queue is changed |
| **Included use case** | None |
| **Extended use case** | VJ-UC-02, VJ-UC-03 |

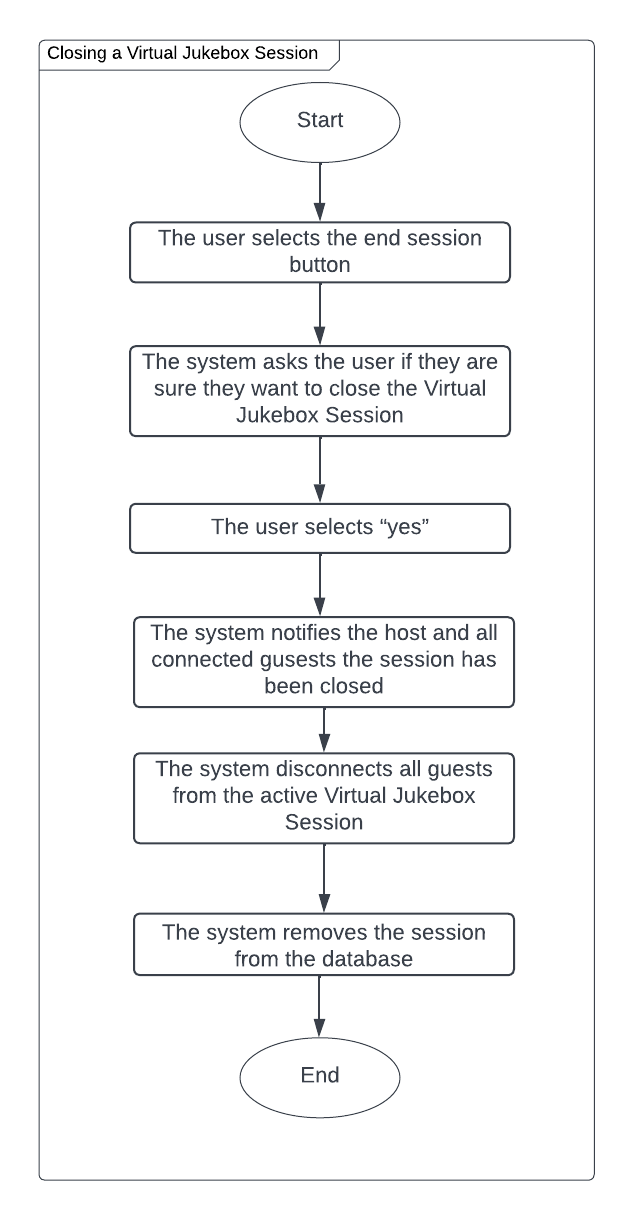
### 3.6.6 Using the Chat Room

| **Goal** | To allow hosts and guests to send messages within a Virtual Jukebox Session. |
| --- | --- |
| **Code** | VJ-UC-06 |
| **Actors** | Guests, Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The system shows the user all messages they have received since joining the Virtual Jukebox Session 2. The user selects the blank message box to start typing a new message 3. The user finishes typing their message and clicks the send message button 4. The system checks the message typed by the user for profanity 5. The system sends the message to all users currently connected to the Virtual Jukebox Session   **Extensions:**  4A) The system detects profanity in the typed message   1. The system will alert the user not to use profane language 2. The system will replace the detected profanity with asterisk symbols 3. The use case resumes at step 5 |
| **Frequency** | A guest or host clicks the view chat button |
| **Pre-processing** | The guest or host is connected to the internet, and is currently connected to an active Virtual Jukebox Session |
| **Post-processing** | A message is sent in a Virtual Jukebox Session and all connected users will receive the message |
| **Included use case** | None |
| **Extended use case** | VJ-UC-02, VJ-UC-03 |



### 3.6.7 Closing a Virtual Jukebox Session

| **Goal** | To allow hosts to close a Virtual Jukebox Session |
| --- | --- |
| **Code** | VJ-UC-07 |
| **Actors** | Guests, Hosts, Virtual Jukebox Application |
| **Description** | **Steps:**   1. The user selects the “end session” button 2. The system asks the user if they are sure they want to close the Virtual Jukebox Session 3. The user selects “yes” 4. The system notifies the host the session has been closed 5. The system disconnects all guests from the active Virtual Jukebox Session 6. The system removes the session from the database |
| **Frequency** | The host selects the session settings button |
| **Pre-processing** | The host is connected to the internet, and is currently connected to an active Virtual Jukebox Session |
| **Post-processing** | The Virtual Jukebox Session is closed, and all connected users will be disconnected. The session will be removed from the database. |
| **Included use case** | None |
| **Extended use case** | VJ-UC-02 |



# 4 Solution Architecture Model

## 4.1 Solution Components

### 4.1.1 Internal Components

**Virtual Jukebox session:** A Virtual Jukebox session is an instance of a song queue with an associated playlist and host, which guest users can join. Each session has one song playing at any given time (unless stopped or paused by the host), and the song queue will dictate which song will play next. Virtual Jukebox Sessions may be public, meaning any guest user may connect to them without a password, or may be private, meaning they may only be connected to by guest users with access to a password or QR code. All Virtual Jukebox Sessions have an associated connection range in metres, in which guest users are unable to join public or private sessions without being within that session’s connection range if they are joining via the Virtual Jukebox map.

**Song queue:** A song queue is a data structure abstracted to all users as an ordered list of songs. The song queue acts as a priority queue, where a higher priority moves elements to the front of the queue. Each element in the song queue is a song, and ordering shifts in accordance to user and Virtual Jukebox session actions such as queuing songs (appends a song), upvoting a song (bumps the song up the queue), and playing a song (removes the front song from the queue). Given no user actions other than queuing songs, the song queue operates in a first-come-first-serve (FIFO) manner. A song queue cannot contain two of the same song at any given time. One song queue is linked to a Virtual Jukebox session, but song queues come in two separate types as follows:

* **User-declared queue**: A user-declared queue exists when any user in a Virtual Jukebox session queues a song. This song queue allows all users within the Virtual Jukebox session to upvote songs, whereas the ambient queue does not.
* **Ambient queue**: An ambient queue exists when and only when there does not exist a user-declared queue. In other words, the ambient queue plays when no songs are queued by users. This queue consists of the entire playlist of the Virtual Jukebox session shuffled.

**Song:** A song is a Spotify-based track, identified by its Spotify ID. Songs can be in a playing state (i.e. currently streaming) or not (queued or part of a playlist) at any given time. Every song stored in the Virtual Jukebox application is part of a playlist, but can be present in song queues and the currently playing song within a Virtual Jukebox session.

**Playlist:** A playlist is a collection of songs. One playlist is associated with a Virtual Jukebox session at any given time which forms the pool of songs that all users are able to choose from to queue. Hosts are able to import Spotify playlists linked to their Spotify Premium account to be used as a playlist for a Virtual Jukebox session.

**Chat room:** The chat room is the Virtual Jukebox application’s component referring to its messaging system. A separate chat room instance exists for each Virtual Jukebox session, which enables all users connected to a session to view chat history (of the session) from the point at which they have joined the session onwards. All users can post textual messages in a chat room, and all users may also view those messages. The chat room includes the following moderation functionality:

* **Message rate limiting**
* **Profane language filtration**

**Virtual Jukebox** **database (PostgreSQL):** The primary database for persistent storage of the Virtual Jukebox application. The database is PostgreSQL-based and situated on the Virtual Jukebox web server. The database is responsible for securely storing Virtual Jukebox account data including user passwords, usernames and emails, as well as Virtual Jukebox sessions data including geographical locations, and connection ranges.

**Virtual Jukebox web server:** The Virtual Jukebox web server is a web server responsible for storing the PostgreSQL database. It will also store the application itself which is deployed via Docker.

**Redis cache:** A server-side cache storing commonly accessed, non-confidential data items for the Virtual Jukebox application.

**Web page view:** The web page view refers to the visible front-end portion of the Virtual Jukebox application. As the application is a single-page web application, the web page view will be responsible for displaying all Virtual Jukebox related information to all users. This includes the:

* **Virtual Jukebox session screen**, which includes the following views:
  + Chat room view
  + Song queue view
  + Playlist selection page view
* **Virtual Jukebox session settings screen**
* **Virtual Jukebox session join screen**
* **Virtual Jukebox dashboard screen**, which has options to find a Virtual Jukebox session or create one (if logged in).
* **Virtual Jukebox map selection screen,** which has different views depending on guest or host use case:
  + Guest users use this screen to select a Virtual Jukebox session for joining
  + Host users use this screen to select a location for the Virtual Jukebox session.
* **Virtual Jukebox login and registration screens**

### 4.1.2 External Components

**Music streaming system:** Also referred to as the Spotify SDK. The music streaming system refers to Spotify’s playback system which is utilised by the Virtual Jukebox application to enable host music playback. The Spotify SDK is responsible for enabling this user playback via API calls to Spotify’s web server.

**Spotify:** The web server used by Spotify to store user account information and track information. The Virtual Jukebox application isn’t responsible for this web server, and assumes it exists within Spotify. The Virtual Jukebox application isn’t responsible for this web server/service, and assumes it exists.

**Virtual Jukebox map system:** Also referred to as the Mapworks SDK. The Virtual Jukebox map system refers to the mapping software development kit i.e. the Mapworks SDK, which is used for sourcing geographical data used for displaying Virtual Jukebox sessions, via API calls to the Mapworks web server.

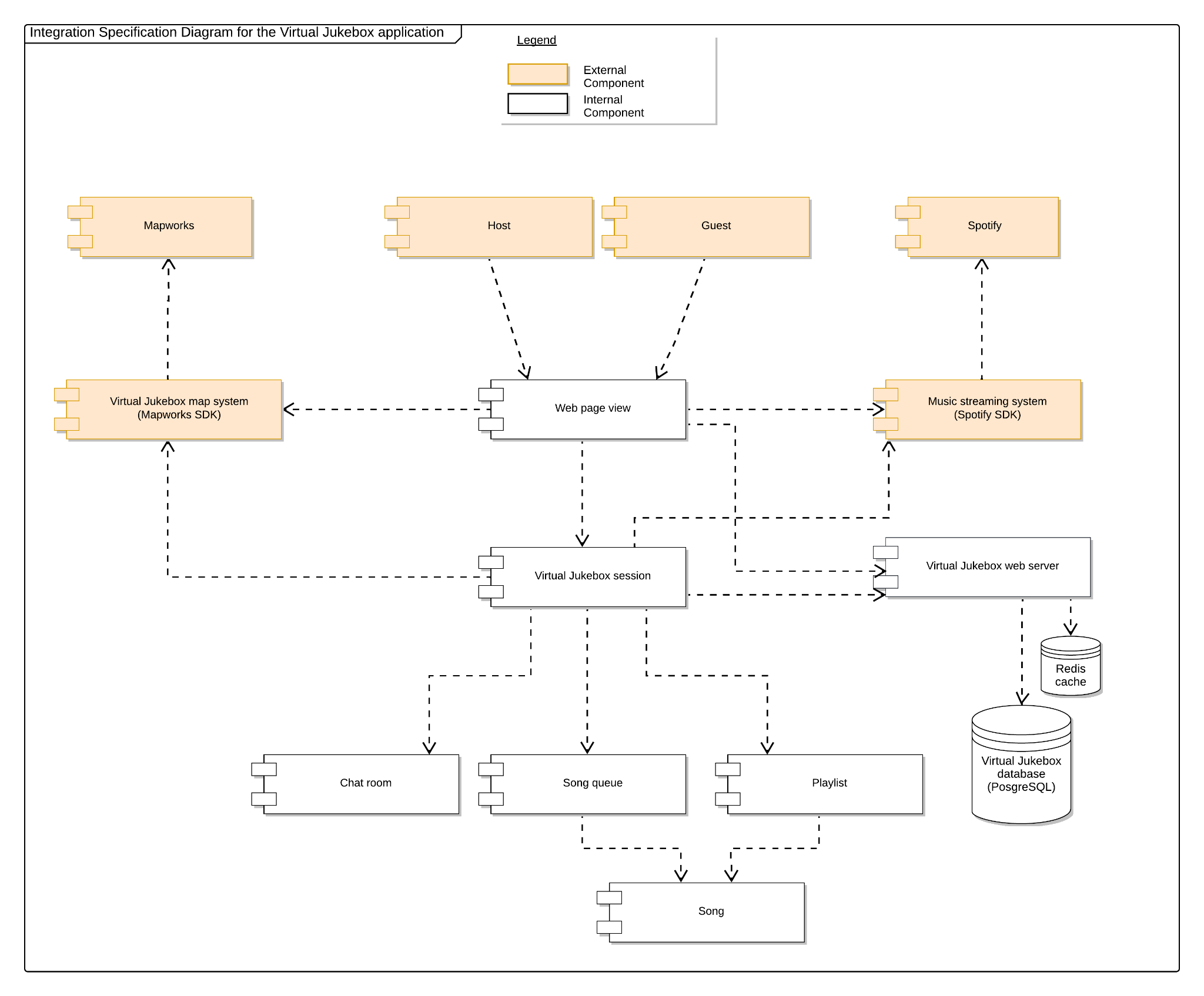
**Mapworks:** The web server used by Mapworks to store geographical information and provide mapping services. The Virtual Jukebox application isn’t responsible for this web server, and assumes it exists within Mapworks. The database resides on the Mapworks web server storing geographical information and mapping services. The Virtual Jukebox application isn’t responsible for this database, and assumes it exists within Mapworks.

**Host:** (See Actors sections for a detailed description). A type of user hosting a Virtual Jukebox session. Hosts require a Virtual Jukebox account as well as a Spotify Premium account to enable Spotify SDK usage. Also has all the functionality of guests while in a Virtual Jukebox session.

**Guest:** (See Actors sections for a detailed description). A type of user that can queue songs in a Virtual Jukebox session using credits, and upvote songs in the song queue.

## 4.2 Integration Specification

The following diagram showcases how each component interacts with and depends on each other.



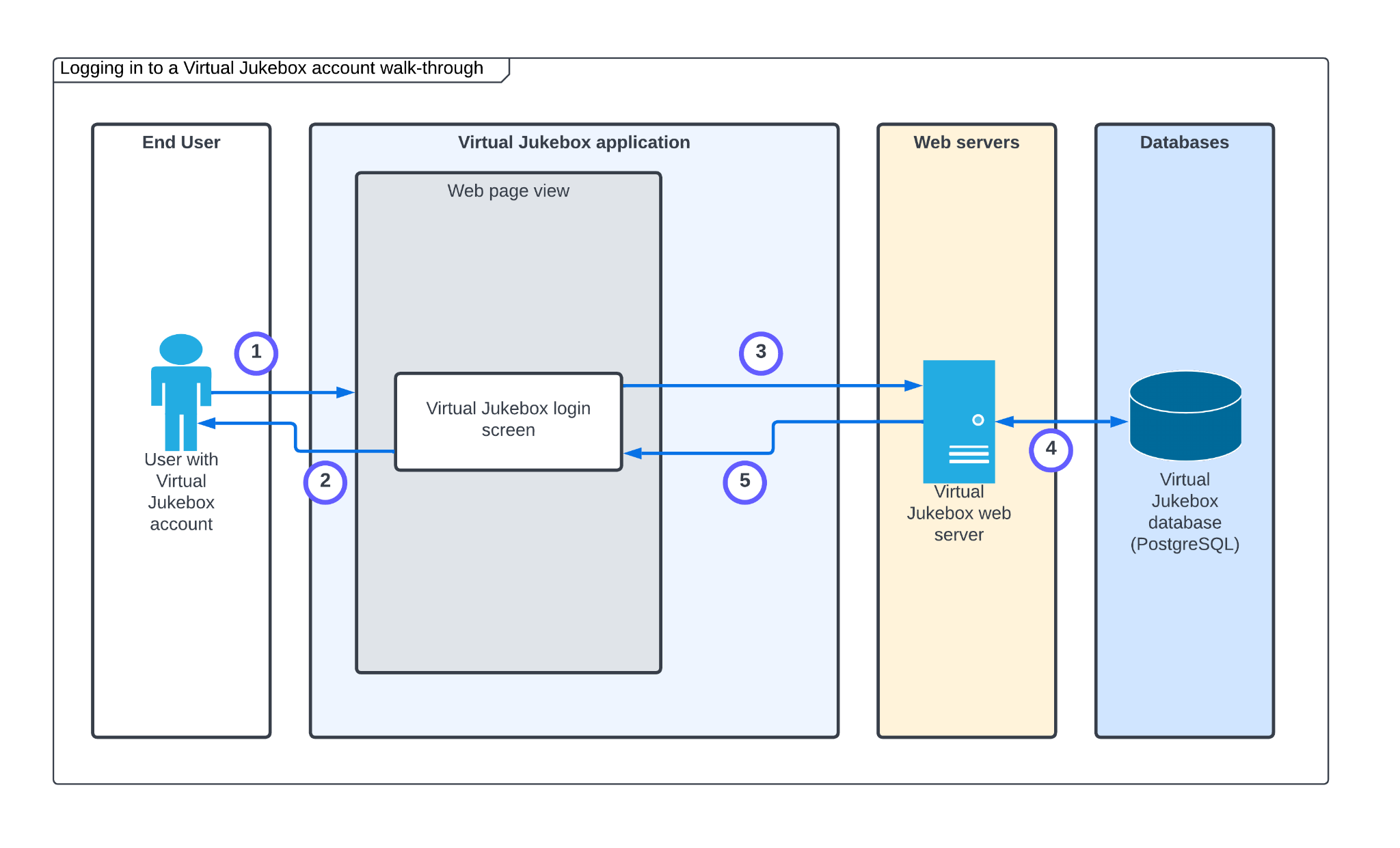
Host and Guest functionality depends on the web page view, as host and guest users are unable to use the application without a front-end web page. Hosts use the web page view to log into their accounts and create Virtual Jukebox sessions, whereas guests use the web page view to search for and join Virtual Jukebox sessions. The web page view depends on Virtual Jukebox sessions as it serves as the interface for collecting and showcasing Virtual Jukebox sessions data and components (such as the chat room and song queue). The web page view also depends on (1) the Spotify SDK to enable host Spotify account verification, (2) the Virtual Jukebox map system to show a map of Virtual Jukebox sessions to users, and (3) the Virtual Jukebox web server to fetch the Virtual Jukebox session data itself in addition to verifying host account login credentials. The Virtual Jukebox map system and the music streaming system depend on their respective services (Mapworks and Spotify) to provide the Virtual Jukebox application with mapping functionally and Spotify integration respectively. The Virtual Jukebox web server is dependent on the database and the Redis cache to act as a data source for Virtual Jukebox application functions such as user login validation.

Virtual Jukebox sessions are dependent on (1) the Virtual Jukebox map system for enforcing session connection ranges, (2) a chat room to provide messaging functionality for connected users, (3) the song queue to provide functionality in song orders, (4) playlists to enable guests to request songs to queue out of a pool of songs specified by the host, (5) The Virtual Jukebox web server to store and retrieve session data (such as session passwords and session connection ranges), and (6) the music streaming system to enable music playback of the current playing song. Both the song queue and playlist depend on songs to enable their respective functionality (as both act as song containers).

## 4.3 Walk-Throughs

### 4.3.1 Logging In to a Virtual Jukebox Account

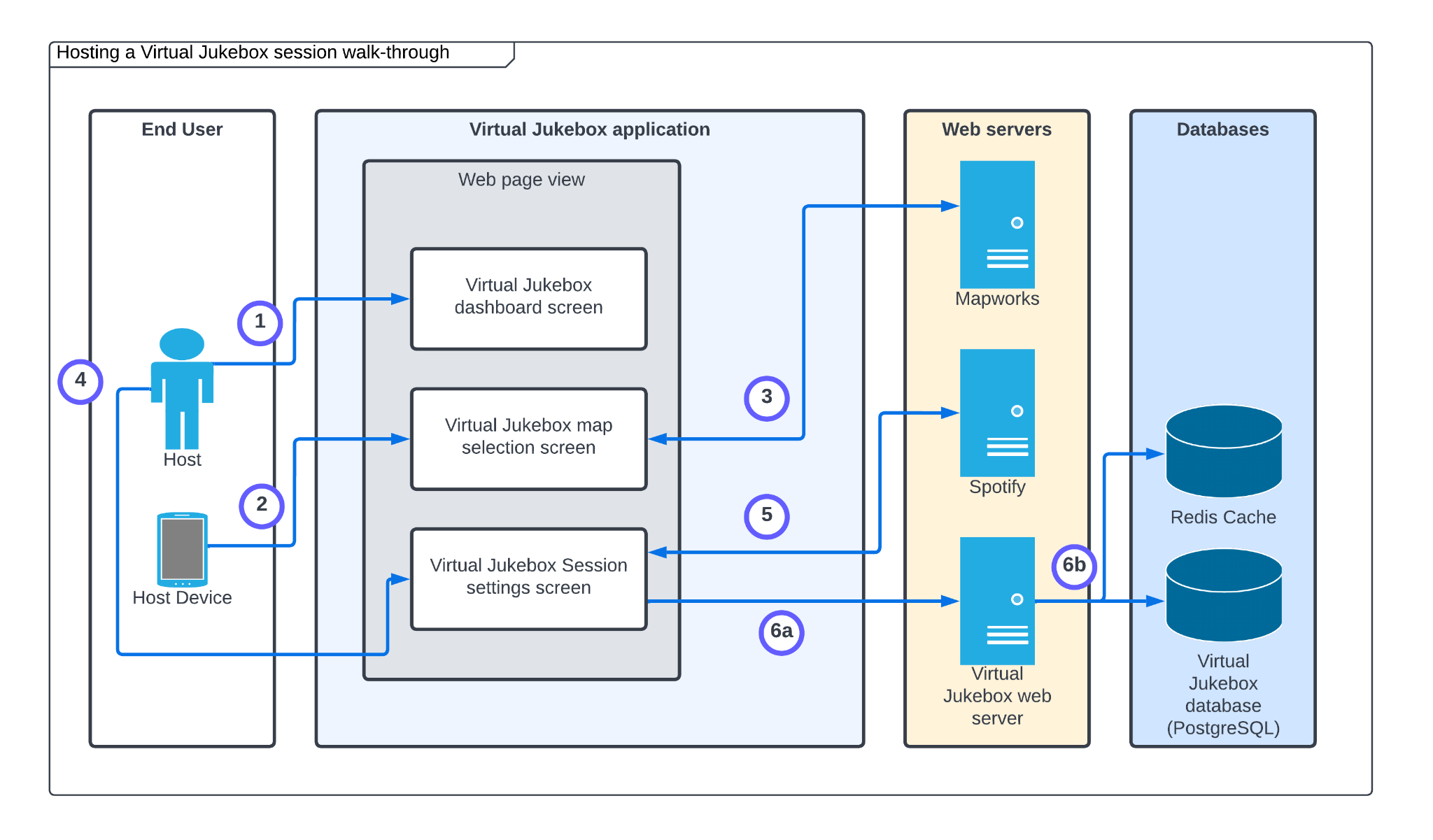
This activity involves a user of the Virtual Jukebox application logging into their account given they have previously registered an account.



| **Step** | **Description** |
| --- | --- |
| 1. | The user accesses the Virtual Jukebox application through a mobile or desktop browser. |
| 2. | The web application is launched and displays login fields including username and password fields for the user. |
| 3. | After providing login credentials, a request is sent to the Virtual Jukebox web server for credential validation. |
| 4. | A query is made to the Virtual Jukebox database to check whether the usernames provided matches any of the usernames stored. If so, the provided password is checked with the corresponding password linked to the stored email matched. |
| 5. | On success, the Virtual Jukebox dashboard screen is displayed on the web page. If the password did not match, or the username was not registered, an error message is displayed to the web page. |

### 4.3.2 Hosting a Virtual Jukebox Session

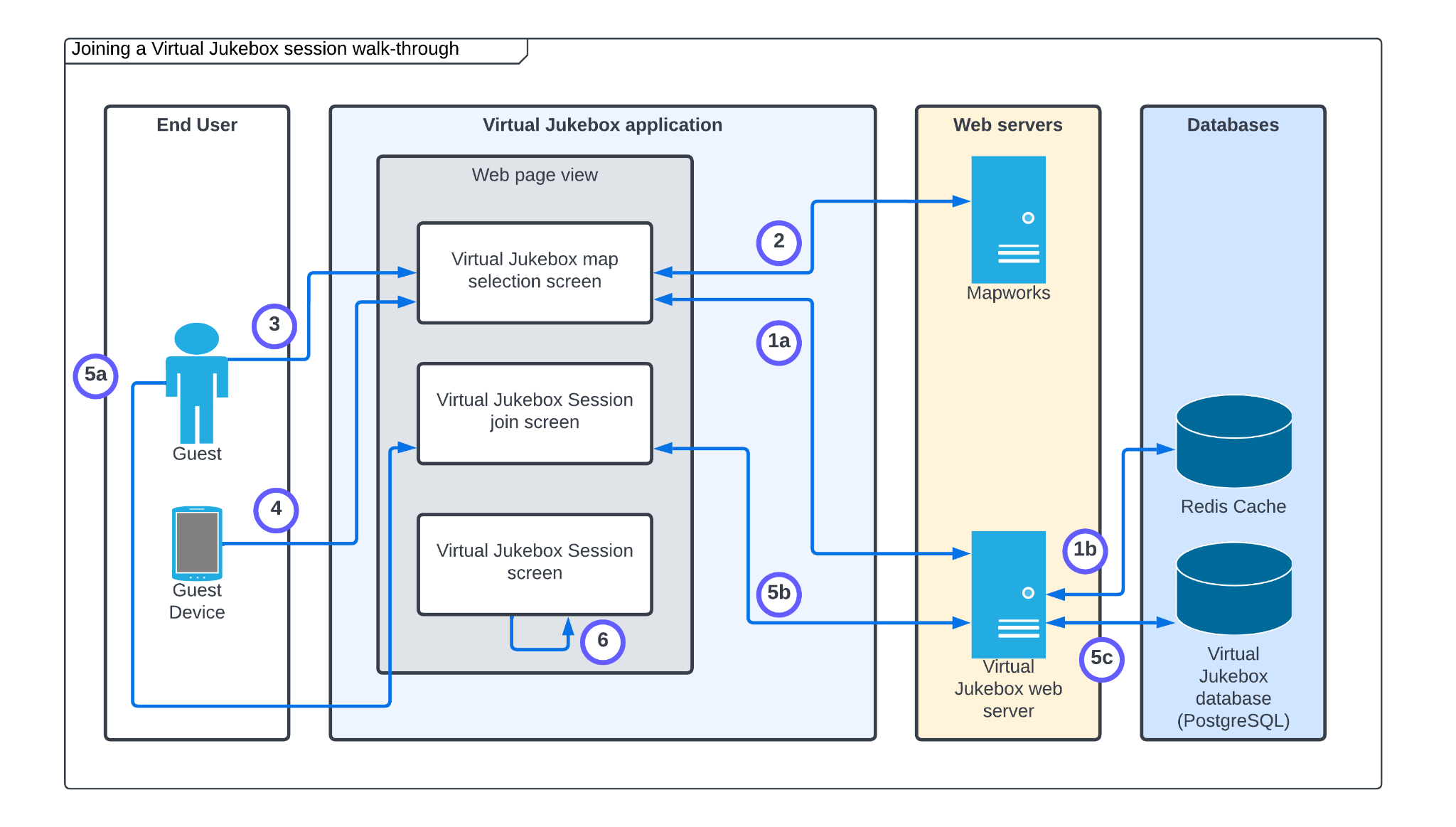
This activity involves a user of the Virtual Jukebox application with an account in setting up a Virtual Jukebox session to become a host.



| **Step** | **Description** |
| --- | --- |
| 1. | The logged in user selects the start a session button |
| 2. | The system retrieves the current location of the host device via GPS if available. |
| 3. | The system makes API calls via the Mapworks SDK to display the Virtual Jukebox map with the current location selected in which users can select a different location if wanted or needed (e.g. no GPS). |
| 4. | The system prompts the user to enter the Virtual Jukebox session’s name and description (to be provided by the user). |
| 5 | The system displays the user’s Spotify playlists by making API via the Spotify SDK, to be selected by the user. The access token and refresh tokens are acquired by a httponly cookie from the end user. |
| 6a + 6b. | The system adds a record to the Virtual Jukebox database, storing the newly created Virtual Jukebox session. For the duration of the Virtual Jukebox session being active, the playlist data (including songs) will be stored on the Redis cache.  (6a - Virtual Jukebox web server access, 6b - writing to the database and Redis cache) |

### 4.3.3 Joining a Virtual Jukebox Session

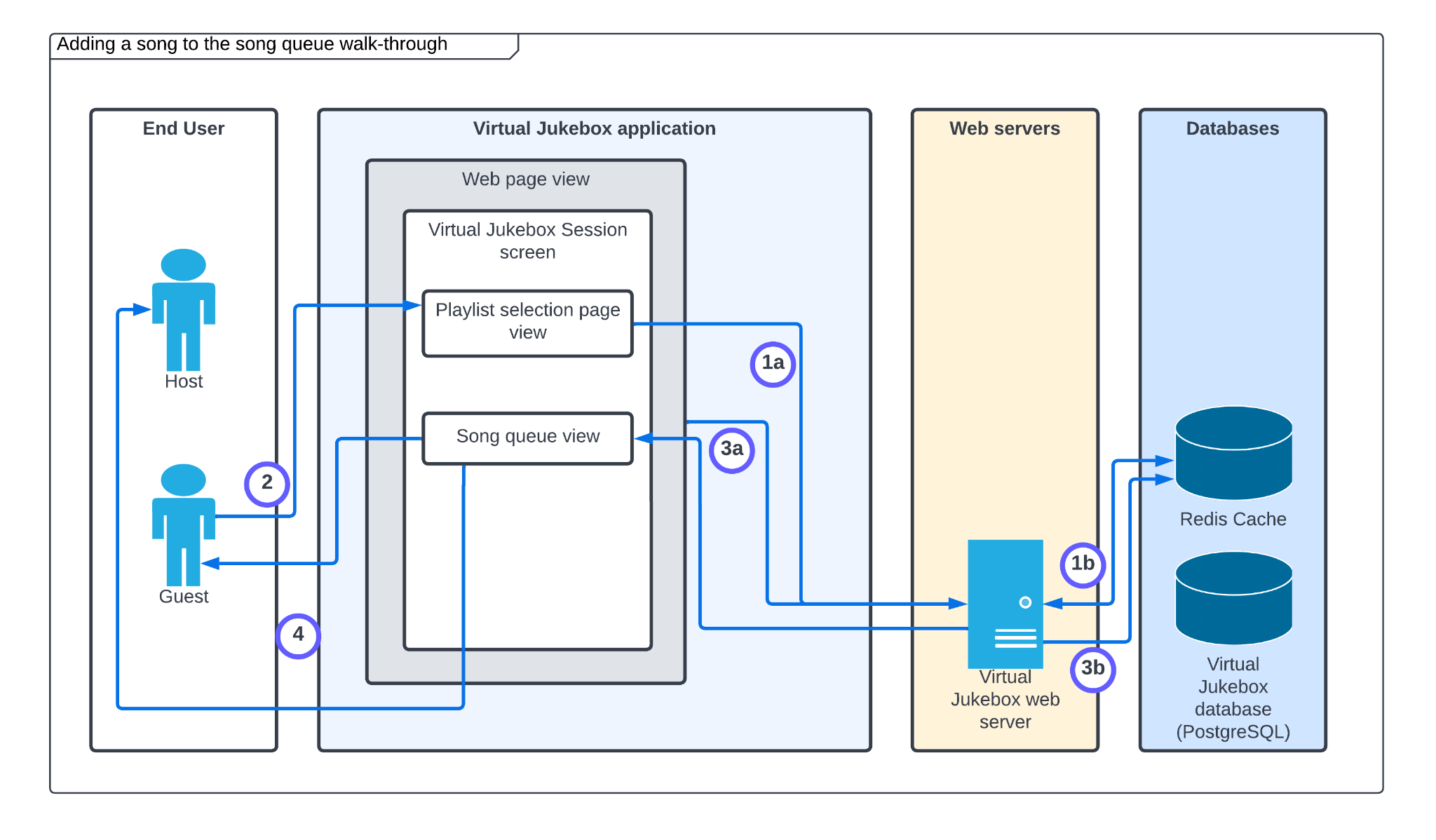
This activity involves a guest of the Virtual Jukebox application joining a Virtual Jukebox session via the Virtual Jukebox map. Note that it is assumed the guest user is using a device with a GPS, and that the user has selected the join a session button.



| **Step** | **Description** |
| --- | --- |
| 1a + 1b. | The system queries the Redis cache via the Virtual Jukebox web server to retrieve all active Virtual Jukebox session data. If it does not exist in the cache, it will instead query the Virtual Jukebox database.  (1a - Virtual Jukebox web server access , 1b - reading from the data stores) |
| 2. | The Virtual Jukebox map system is used (via API calls to Mapworks) to display all active Virtual Jukebox sessions on a map for the user. |
| 3. | The user selects the Virtual Jukebox session on the map selection screen (displayed using the Mapworks API). |
| 4. | On Virtual Jukebox session selection, the system uses the guest device’s GPS to check whether the selected session is within connection range of the device (queries database of stored session geographical location data). |
| 5a + 5b + 5c. | The guest enters a password if necessary. This password is hashed and checked with the hash of the corresponding session’s password stored on the Virtual Jukebox database.  (5a - guest provides the password, 5b/5c - Virtual Jukebox web server and database access with password authentication). |
| 6. | The Virtual Jukebox session’s data is loaded on the Virtual Jukebox session screen. |

### 4.3.4 Adding a Song to the Song Queue

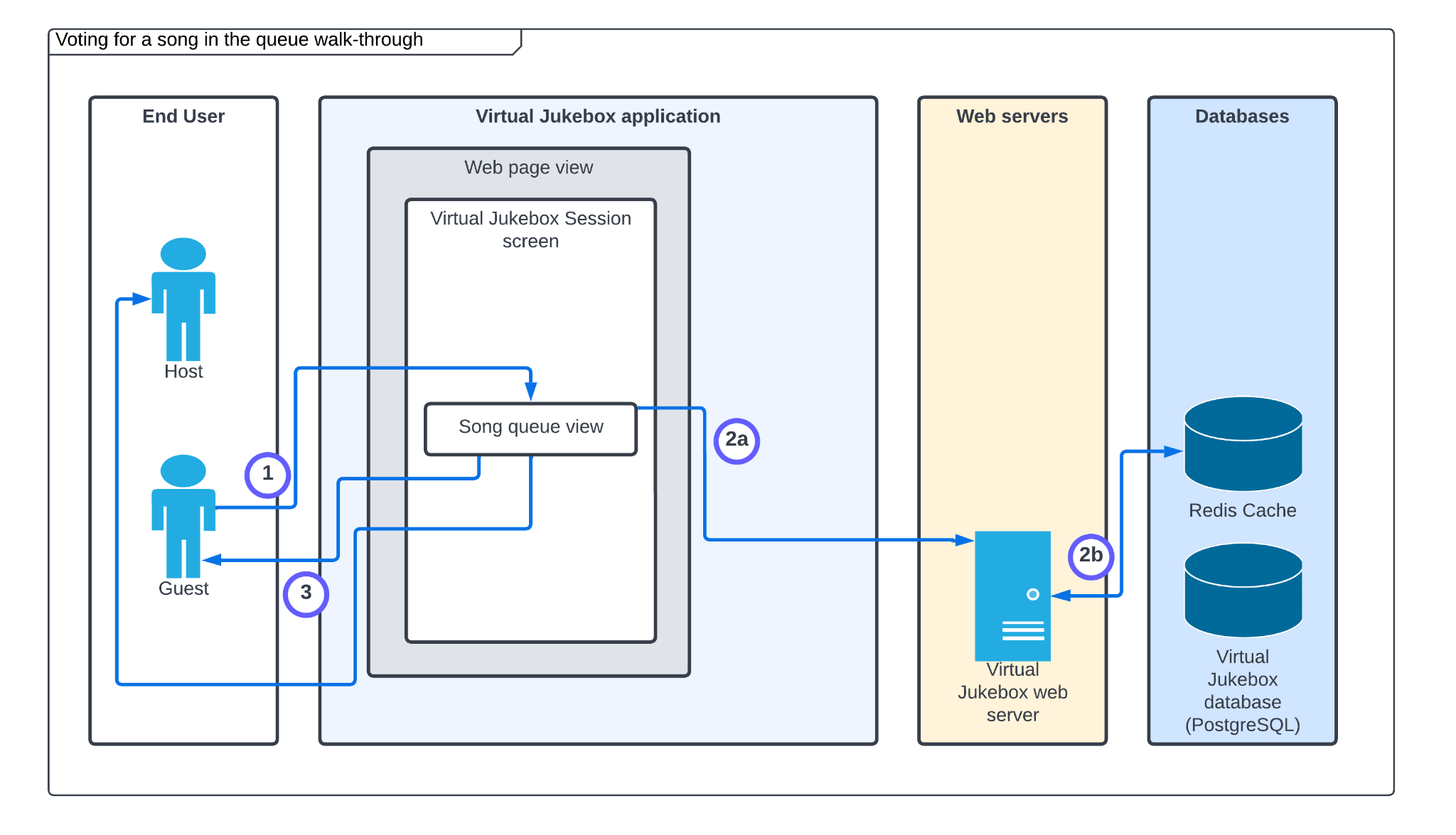
This activity involves a guest or host user queuing a song within a Virtual Jukebox session. It assumes guests have already joined the session.



| **Step** | **Description** |
| --- | --- |
| 1a + 1b. | The system displays a list of all songs in the playlist of the Virtual Jukebox session (playlist selection view) by accessing the playlist via the Redis cache.  (1a - Virtual Jukebox web server access , 1b - reading Virtual Jukebox session playlist from the Redis cache) |
| 2. | The user selects a song in the playlist. |
| 3a. + 3b. | The system queries the amount of credits the user has on their account for the Virtual Jukebox session by querying the Redis cache. If the amount of credits assigned to the user is > 0, the system decrements that credit counter and writes the new song to the end of the song queue.  (3a - Virtual Jukebox web server access , 3b - reading/writing to the Redis cache) |
| 4. | The system pushes the updated song queue to all user’s web page views in the Virtual Jukebox session. |

### 4.3.5 Voting for a Song in the Queue

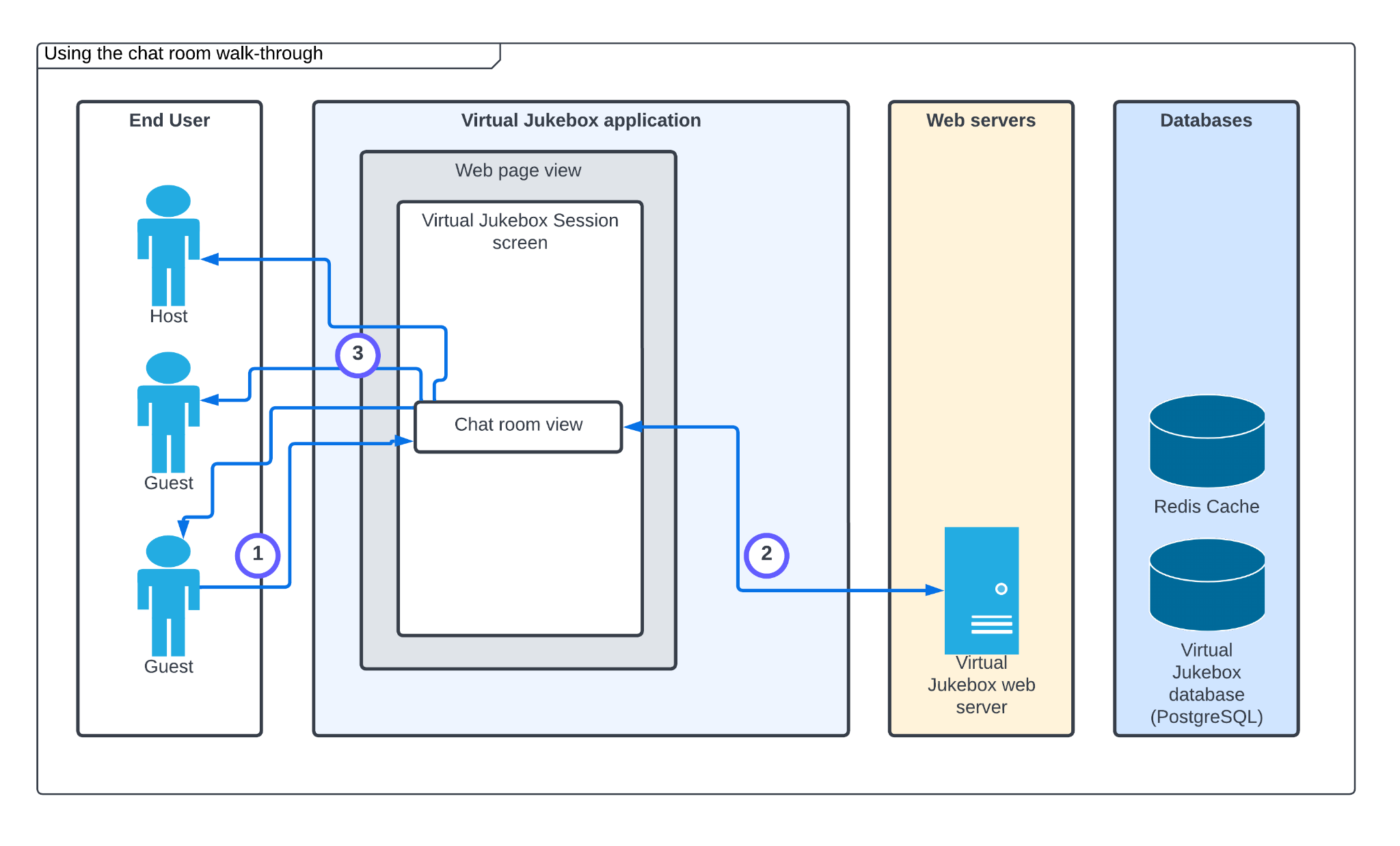
This activity involves a guest or host user voting for a song within the song queue given it is a user-declared song queue, in order to increase the priority of that song in the queue.



| **Step** | **Description** |
| --- | --- |
| 1. | A user selects the upvote button next to a song in the song queue. |
| 2a + 2b. | An update Redis operation is undergone to increase the priority of the selected song in the song queue. |
| 3. | The system pushes the updated song queue to all user’s web page views in the Virtual Jukebox session. |

### 4.3.6 Using the Chat Room

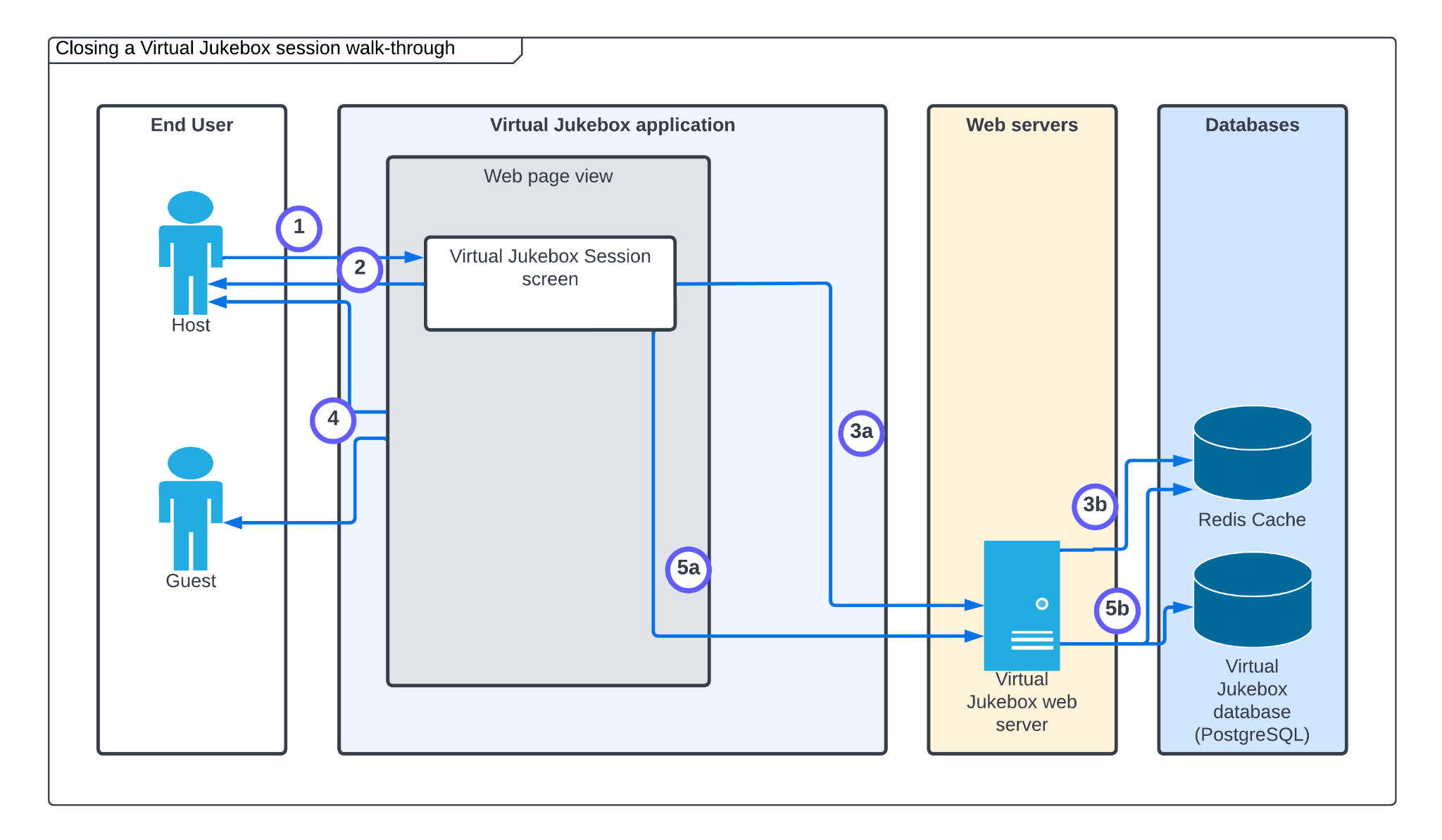
This activity involves a guest or host user posting and reading messages in the chat room while in a Virtual Jukebox session.



| **Step** | **Description** |
| --- | --- |
| 1. | The user clicks the empty text field/box, enters a message and clicks the send message button on the chat room view. |
| 2. | The message is passed through a profanity filter. |
| 3. | If the message has no profanity, the message is added to the chat room view, and the view is updated for each user connected to the session via WebSockets. |

### 4.3.7 Closing a Virtual Jukebox Session

This activity involves a host closing an active Virtual Jukebox session, disabling guests from seeing or accessing the session which will also remove all contents from the song queue.

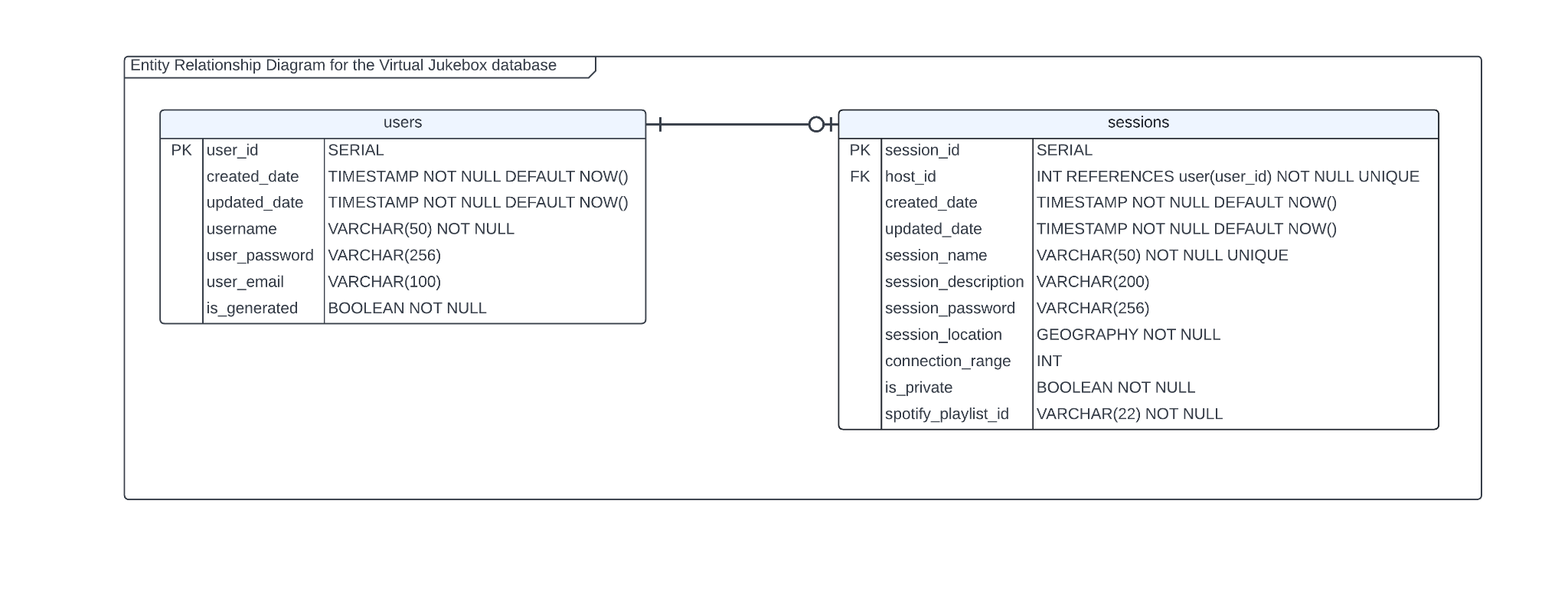


| **Step** | **Description** |
| --- | --- |
| 1. | The host user clicks the “end session” button and confirms their selection. |
| 2. | The system notifies the host that the Virtual Jukebox session is closed. |
| 3a + 3b. | The system disconnects all guest users and the host from the Virtual Jukebox session, and removes user information from the Redis cache.  (3a - Virtual Jukebox web server access , 3b - removing user information from the Redis cache) |
| 4. | The web view of each previously connected user is updated to indicate the Virtual Jukebox session has closed. |
| 5a + 5b. | The corresponding Virtual Jukebox session record is removed from the database, and the playlist information is removed from the Redis cache.  (5a - Virtual Jukebox web server access , 5b - removing session information from the Redis cache and Virtual Jukebox database) |

## 4.4 Information Model

### 4.4.1 Virtual Jukebox Database

The following Entity Relationship Diagram (ERD) outlines the information model of the Virtual Jukebox database, including its tables, attributes and relationships.

Note that “Virtual Jukebox” has been omitted from these table names that represent components that would usually include it. For example, “Virtual Jukebox sessions” are represented by the “session” table.

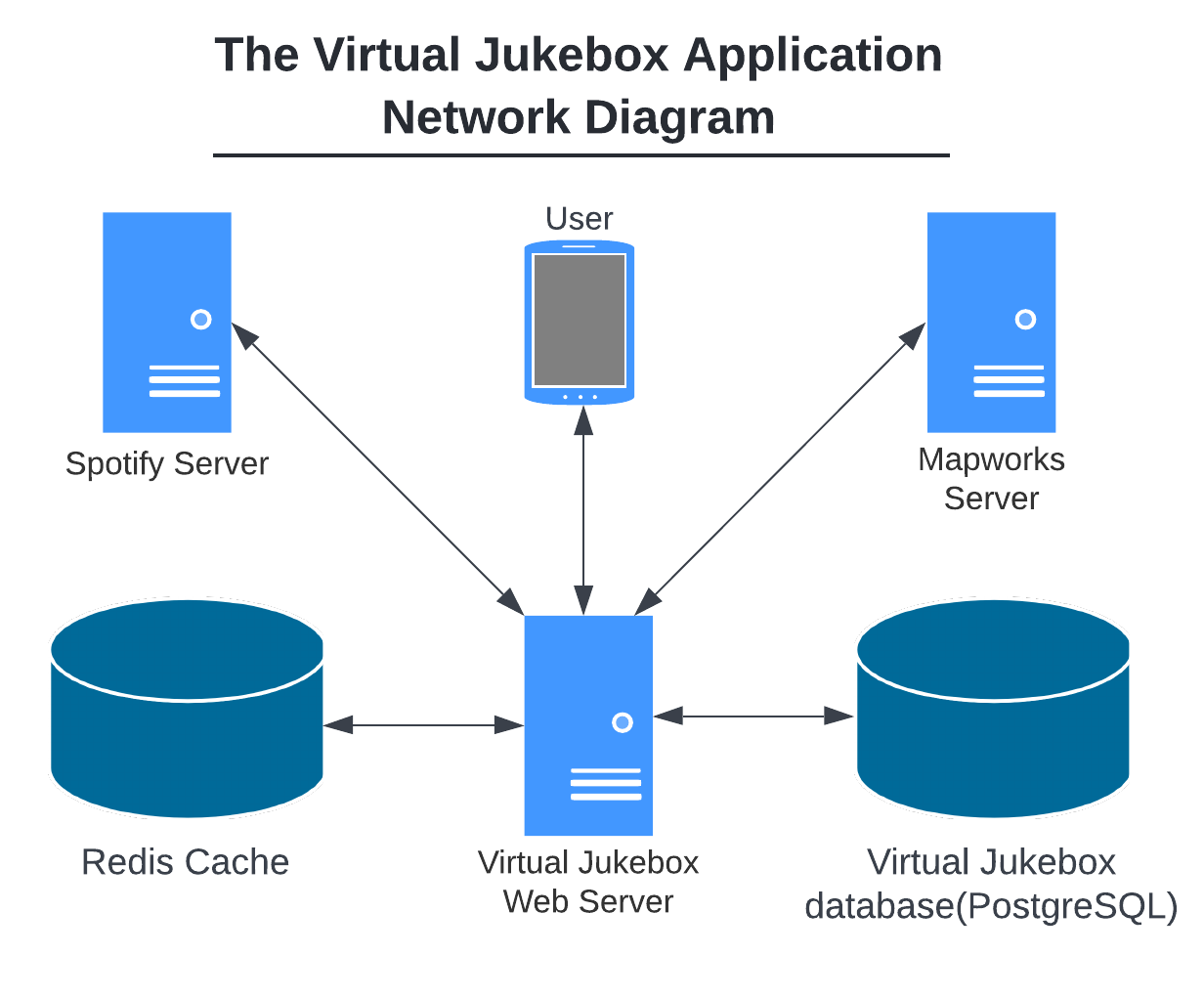
The passwords stored in the users/sessions tables are stored using the PBKDF2 algorithm and hashed using SHA-256 with a salt by Django. Playlist architectures such as songs, albums and artists aren’t stored in the database, and are instead stored as a spotify ID, which can be used with the Spotify SDK to retrieve this data if the cache expires. Each table is given a created and updated date field to track times that changes made to records. Both generated users (guests without Virtual Jukebox accounts) and registered users are stored in the users table, which are distinguishable via the is\_generated boolean field.

### 4.4.2 Redis cache

Each song queue is stored on the Redis cache/on-memory, as there is no need for long term persistence. Some session data is stored in the Redis cache including session names, locations, connection ranges, state of privacy, and descriptions. Playlist and song data is stored on the Redis cache also, as they will not be persistent, and can simply be queried via the Spotify SDK in the event that they need to be obtained and do not exist on the cache. Storing this on the cache will reduce the need for recurring calls to Spotify (via their API) whenever a song is queued. Credits for each user will reset when they leave a Virtual Jukebox session, therefore there is no reason to make them persistent, meaning their counters will reside on the Redis cache/on-memory. The playlist of a Virtual Jukebox session is stored on the Redis cache due to its frequently accessed nature, and the fact that it wouldn’t be changed often (synchronisation not a concern).

# 5 Detailed Physical Architecture

## 5.1 Network Diagram



Communication between the Spotify, Mapworks and Virtual Jukebox Servers will use the HTTPS protocol. This will be over the 443 port number.

To communicate from the Virtual Jukebox Web Server to the Virtual Jukebox database it will use the TCP/IP protocol over the 5432 port number.

Redis will connect to the Virtual Jukebox Web Server using TCP protocol as well, over the 6379 port number.

## 5.2 System Specification Summary

The Virtual Jukebox application will be set up as a multi-container Docker application, with each Docker container being responsible for one task. The Docker images used to create the containers will be the official Docker images for the specified technology. The server that the application is to be deployed must meet the minimum requirements to run at least one of each mentioned container concurrently.

| **Docker Container** | **Docker Image** |
| --- | --- |
| Vue | Node - Official Image |
| Django | Python - Official Image |
| PostgreSQL | Postgres - Official Image |
| Redis | Redis - Official Image |

For the technologies, the version used will be the most recent stable release at the time of development beginning.

On the client side of the application, users will require access to a mobile device or computer that can run a modern web browser (Microsoft Edge, Safari, Google Chrome, and Mozilla Firefox), as well as an Internet connection.

**Question:**

* Not sure about the specifications of the server - Should we just mention the minimum requirements, or does this depend on how many users/traffic we expect to have?

## 5.3 Configuration Requirements

To allow for communication between the Docker containers, they will all be hosted on the same server and communication will occur using the Docker bridge network. The PostgreSQL database will expose port 5432, which will allow the Django web server to connect to the database. To allow for the Vue front-end on the client side of the application to retrieve data from the Django web server, port 8000 will be open on the web-server. To allow for the Django web server to read and modify the Redis cache, port 6379 will be used.

# 

# 6 Solution Impact

This section should describe the impact which the solution has on processes, people and existing technology.

## 6.1 Impact on Processes

Cafes are places that usually use speakers to play music for their customers. The Virtual Jukebox application will change this process, by removing the need for speakers. The customer will be able to listen to the music that is hosted by the cafe, through their mobile phone, changing the usual process of playing music through a speaker.

## 6.2 Impact on People

When a customer goes to a cafe or a music event they are only able to listen to the music and they have no interaction with what is being played. The Virtual Jukebox application gives the users that join the session the ability to vote on what song is played next and it also gives them the ability to add a song to the queue This functionality makes the user that is listening not only a consumer, but someone that can interact with a group.

## 6.3 Technical Impact

To connect to the Virtual Jukebox application, a user will need to use a mobile or a laptop to be a host or a guest/listener. While connecting to the Virtual Jukebox application this will use a small percentage of the connected devices battery.

If the connection is made using a cellular network, then there will also be data usage over the cellular network. The device can also be connected over WI-FI which means the venue will need a modem which has a router connected to it that is capable of WI-FI.

To play music on the host user’s device, it will need to use the device's speakers. There are also possibilities for the user to connect external audio devices to the device connected to the Virtual Jukebox application.

When a guest/host user wants to join or host a session, if the device has a built-in GPS, it will use that to find the current location of the user. This is done to see where the user is located and show the Virtual Jukebox sessions in the area and It is also to select a location to host the session.

For the Virtual Jukebox application to operate a Virtual Jukebox session, the device will need to be connected to the internet which will use the Spotify API. This means that the Spotify server must be operational for the Virtual Jukebox session to work.

# 

# 7 Quality Attributes

In discussion of the quality attributes applicable to the solution, specific scenarios relating to each attribute–namely, reliability, availability and serviceability–and in turn, the solution, will be discussed with use of *quality attribute scenarios*. Consider a quality attribute scenario as a brief description of how the application is required to respond to some stimulus.

As per (Carnegie Mellon University 2015) there are six parts which comprise a quality attribute scenario. In order, they are:

1. *Source*: some entity (user, developer, the application etc) which generates a stimulus
2. *Stimulus*: a condition which affects the application
3. *Artefact*: the part of the application which was stimulated by the stimulus
4. *Environment*:the condition under which the stimulus occurred
5. *Response*: the activity to be completed as a result of the stimulus
6. *Response measure*: the measure by which the response will be evaluated

## 7.1 Reliability

### 7.2.1 Fault Tolerance

**Scenario**: *A fault in the application leads to a system-wide failure*

**Source**: The application

**Stimulus**: A fault in the application

**Artefact**: The component in which the fault occurred

**Environment**: Normal operation

**Response**: The application handles the resultant failure, allowing it to fail gracefully

**Response Measure**: A description detailing the nature and location of the failure is entered into an error log

### 7.2.2 Fault Avoidance

**Scenario**: *The User attempts a task that could result in a fault in the application*

**Source**: External to the application

**Stimulus**: User attempts to perform an invalid action

**Artefact**: Client

**Environment**: Normal operation

**Response**: The client provides visual feedback stating whether the User’s action was successful. If it was unsuccessful, the client will provide details in regards to what went wrong and how to rectify the issue, if possible.

**Response Measure**: User satisfaction

### 7.2.3 Capacity

**Scenario**: *The application is unable to handle a large number of concurrent user requests*

**Source**: External to the application

**Stimulus**: Application receives more requests per second than it can process

**Artefact**: Client

**Environment**: Normal operation

**Response**: The client provides visual feedback stating whether the user’s action was successful. If it was unsuccessful, the client will provide details in regards to what went wrong and how to rectify the issue, if possible.

**Response Measure**: The client takes no longer than 10 seconds to load any given page

## 7.2 Availability

### 7.2.1 Malicious Attack

**Scenario**: *The application is compromised by a Denial of Service (DoS) attack*

**Source**: External to the application

**Stimulus**: Application receives more requests per second than it can process

**Artefact**: External to the application

**Environment**: Normal operation

**Response**: The client blocks concurrent user requests, displaying a message to the user informing them to wait momentarily before attempting that particular task again.

**Response Measure**: The DoS attack does not persist for longer than one minute.

### 7.2.2 Unresponsiveness

**Scenario**: *The Spotify platform becomes unresponsive, causing the application to fail*

**Source**: Internal to the application

**Stimulus**: A fault in the Spotify platform

**Artefact**: The Spotify platform

**Environment**: Normal operation

**Response**: The application handles the resultant failure, and shuts down until the Spotify platform returns to normal operation

**Response Measure**: The uptime of the application is at least 95%

## 7.3 Serviceability

### 7.3.1 Documentation

**Scenario**: *A lack of technical documentation makes it difficult to use the application, manage the codebase and add new features*

**Source**: Developers

**Stimulus**: The writing of high quality documentation was not prioritised throughout development, and as a result, does not provide an accurate understanding of the functionality of the application

**Artefact**: Technical documentation

**Environment**: Development, runtime

**Response**: Developers dedicate more time towards improving documentation

### 7.3.2 Coding Standards and Best Practices

**Scenario**: *The codebase has become too large and complex, making it difficult to use the application, manage the codebase and add new features*

**Source**: Developers

**Stimulus**: Application source code was written without strict adherence to a predefined set of standards, style guides, and Software Engineering concepts.

**Artefact**: The application

**Environment**: Development, runtime

**Response**: Developers dedicate more time towards implementing components in a manner which coincides with Software Development best practices–consistently following standards and employing various Software Engineering design patterns

**Scenario**: *The components of application codebase and their associated layers have become excessively dependent and indistinguishable from one another*

**Source**: Developers

**Stimulus**: Application source code was written without strict adherence to a predefined set of standards, style guides, and Software Engineering concepts.

**Artefact**: Interrelated components of the application

**Environment**: Development

**Response**: Developers design and implement components with reverence to Software Engineering principles–namely, coupling and cohesion.

## 7.4 Best Practice Commentary

### 7.4.1 Deliverable Scheduling

Delivering the TIS prior to the SRS may lead to greater complications during development.

Had delivery of the SRS been scheduled before the TIS, the Developers would have been granted the ability to consider a concrete set of requirements during the technology investigation (and proposal) process. This would have allowed for more in-depth analysis centred solely on the aspects of the technological components applicable to the application. In turn, it would have encouraged better judgement in deciding which set of technologies to propose.

However, under the prescribed scheduling the Developers could only infer requirements in reference to the “capabilities” outlined in the Project Brief. This could have perhaps resulted in less appropriate–or perhaps, outright inappropriate–technologies being proposed for use during development. Ultimately, the technologies used to implement the application may differ from those proposed in the TIS if a change in technologies is deemed necessary in order to more adequately meet the requirements prescribed in the SRS.

## 7.5 Architecture Commentary

### 7.5.1 Monetisation

Changes in regards to business requirements–namely, monetisation–will disallow further usage of the Spotify Platform. Spotify only allows developers to develop and distribute streaming applications which are solely for personal, non-commercial use. Consequently, Amristar would need to either source a streaming platform which enables monetisation or a media playback solution using local files. In either case, a considerable redesign of the application may be necessary. The associated resource consumption–time, money, personnel–may drastically outweigh the potential business value of monetisation.

# 8 Success Criteria and Test Plan

## 8.1 Use Case Validation

| **Use Case ID** | **Use Case Name** | **Validation** |
| --- | --- | --- |
| VJ-UC-01 | Logging into a Virtual Jukebox Account | When the user clicks on the login button on the Virtual Jukebox application it will bring them to the Login/Sign-up Page where the user can enter their username and password.  If the password for the given username is correct it will notify the user that they have successfully logged in and bring the user to the Home Page. |
| VJ-UC-02 | To allow hosts to create a Virtual Jukebox Session | When the user is logged in to the homepage and clicks the ‘Host Session’ button, it will check if the current user is already hosting a session in the database. It will then bring them to the Host Session Page.  The user will be able to choose a location from the map that will use the user's location data to prefill the current host location.  The user can add a session title and description for the session and click the ‘Next’ button.  The ‘Next’ button will bring the user to another page where it calls the Spotify API to display the user's Spotify playlist in a scrollable layout and the user can select which playlist to use for the session.  The user can click on the ‘Host Session’ button to create the session which will bring them to the Virtual Jukebox Session page. |
| VJ-UC-03 | To allow guests to join a Virtual Jukebox Session | When the user is logged in to the homepage.  A map using the user's current location will be loaded where it will display all active Virtual Jukebox sessions. The user will be able to click on the landmark icon on the map to join the session.  After clicking the ‘Join Session’ button the system will bring the user to the Virtual Jukebox Session page. |
| VJ-UC-04 | To allow hosts and guests to add songs from a Virtual Jukebox Session’s playlist to the session’s song queue. | When the user is logged in to the homepage and has selected to join a Virtual Jukebox session.  The system will show the cover art of the currently played song. It will also show all the songs in the queue.  The user can click on the ‘Add Song’ button and the system will go to the Add Song to Queue page.  The user will then be able to select what song they want to add from the playlist to the queue. After adding the song the system will bring the user back to the Virtual Jukebox Session page. |
| VJ-UC-05 | To allow hosts and guests to upvote songs currently in a Virtual Jukebox Sessions’s song queue. | When the user is logged in to the homepage and has selected to join a Virtual Jukebox session.  The system will show the user all the songs in the queue and the user can upvote any song in that queue by clicking the thumbs up icon.  The queue will rearrange itself based on the number of upvotes. The higher the upvotes the higher the song priority in the queue will be. |
| VJ-UC-06 | To allow hosts and guests to send messages within a Virtual Jukebox Session. | When the user is logged in to the homepage and has selected to join a Virtual Jukebox session and clicked on the Message button.  The system will show all the messages since the user joined the Virtual Jukebox session. The user will be able to enter their message in the message box below and send their message. |
| VJ-UC-07 | To allow hosts to close a Virtual Jukebox Session | When the host user is logged in to the homepage and has selected to go on their hosted Virtual Jukebox session.  The user can click the ‘End Session’ button which the system will then bring up a pop up to confirm the ending of the session. The user can then select ‘Yes’ to end the session which prompts the system to end the Virtual Jukebox session and bring the user back to the home page. |

## 8.2 Validation of Solution metrics

| **Application load time (first load)** | 1 – 3 seconds |
| --- | --- |
| **Application load time (subsequent loads, cached)** | 1 – 2 seconds |
| **Data refresh times** | 1 – 2 seconds |
| **Application load time to another page (no connection to other servers)** | 0.5 - 1.5 seconds |
| **Application load time to another page (with connection to another server)** | 2 - 4 seconds |
| **Submission of form to Virtual Jukebox database** | 1 - 3 seconds |
| **Submission of form to Spotify Server** | 2 - 4 seconds |
| **Retrieve information from cache** | 0.5 - 1 seconds |

## 8.3 Test Plan

# 

# 

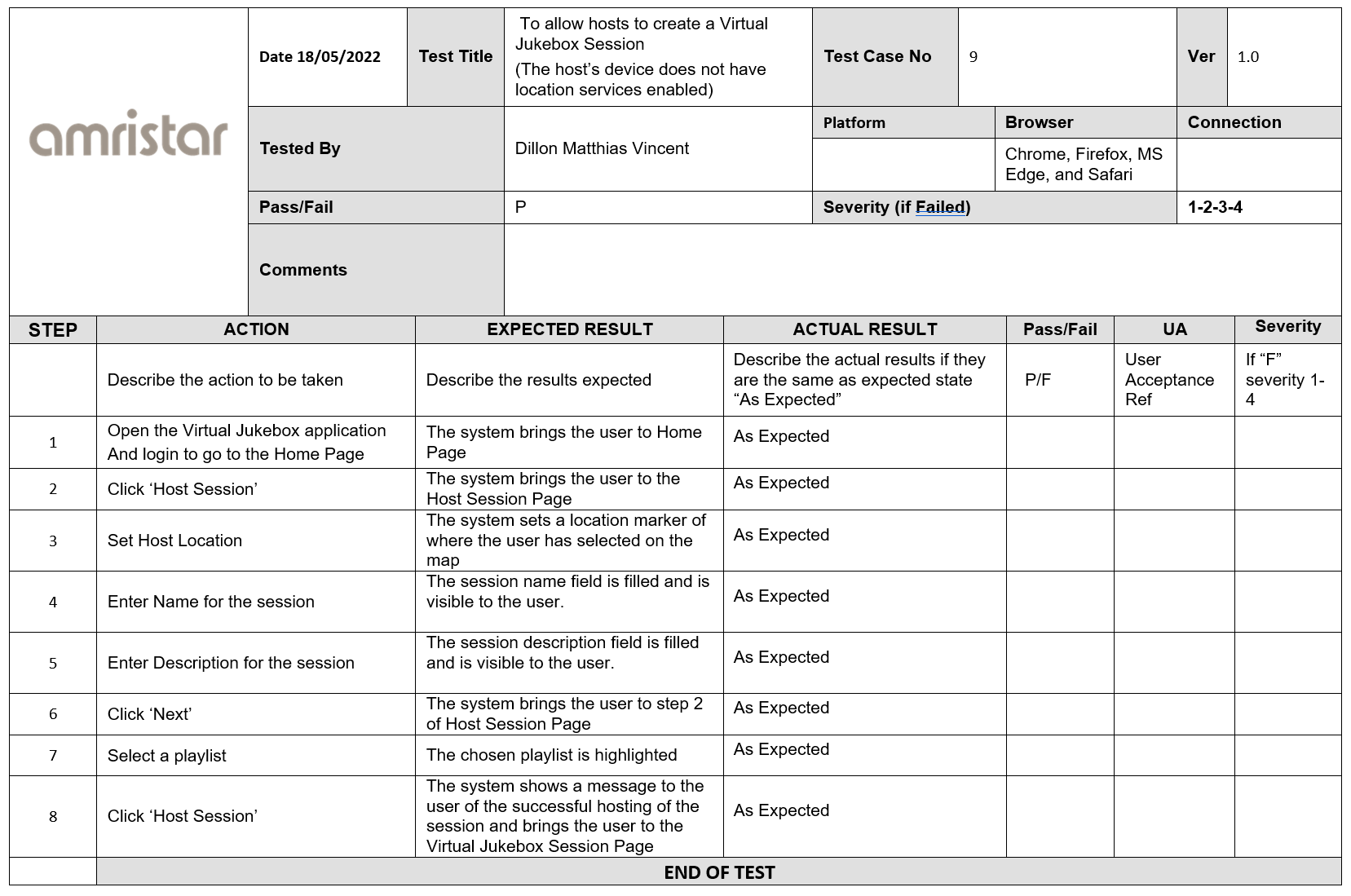
# 

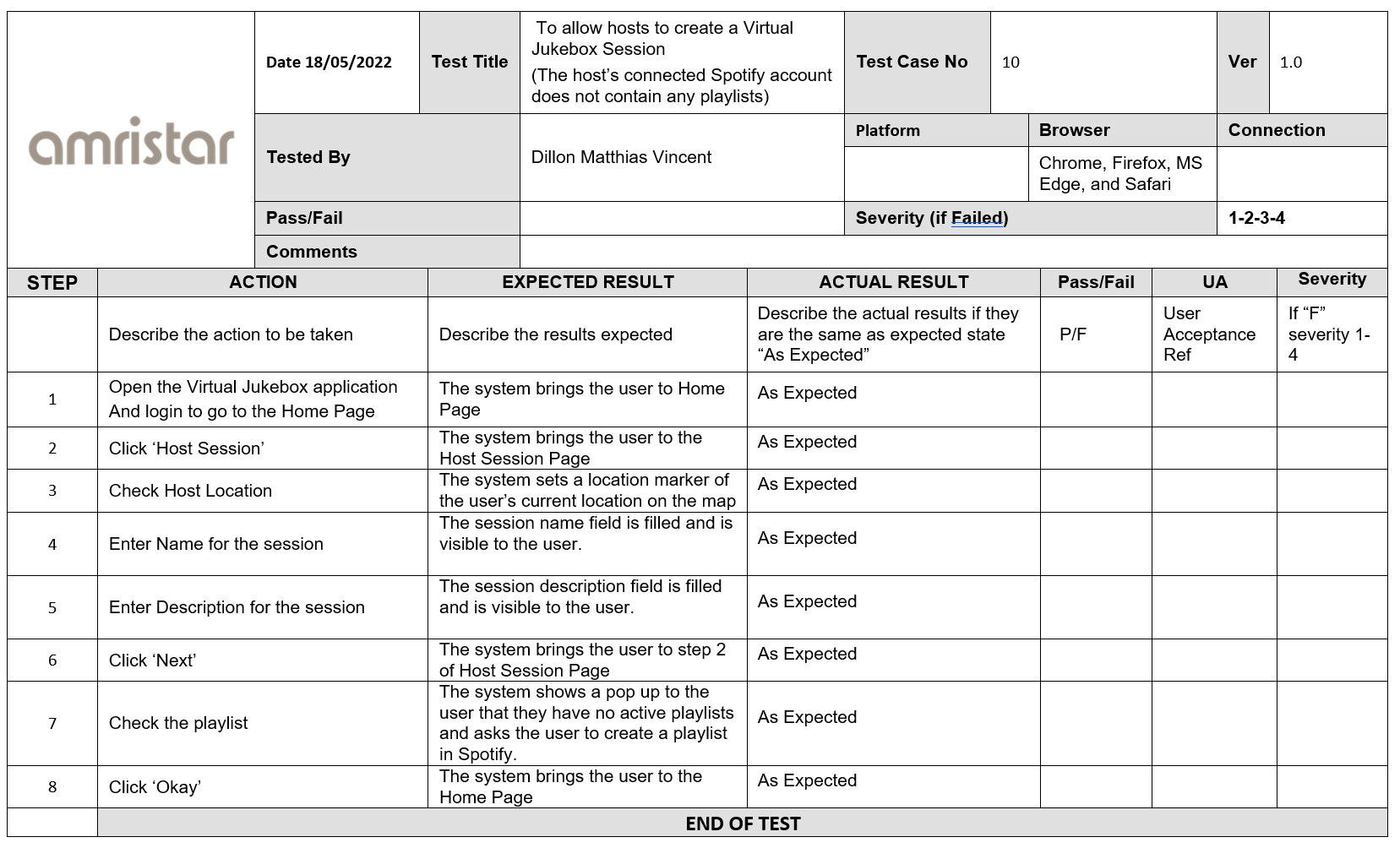
# 

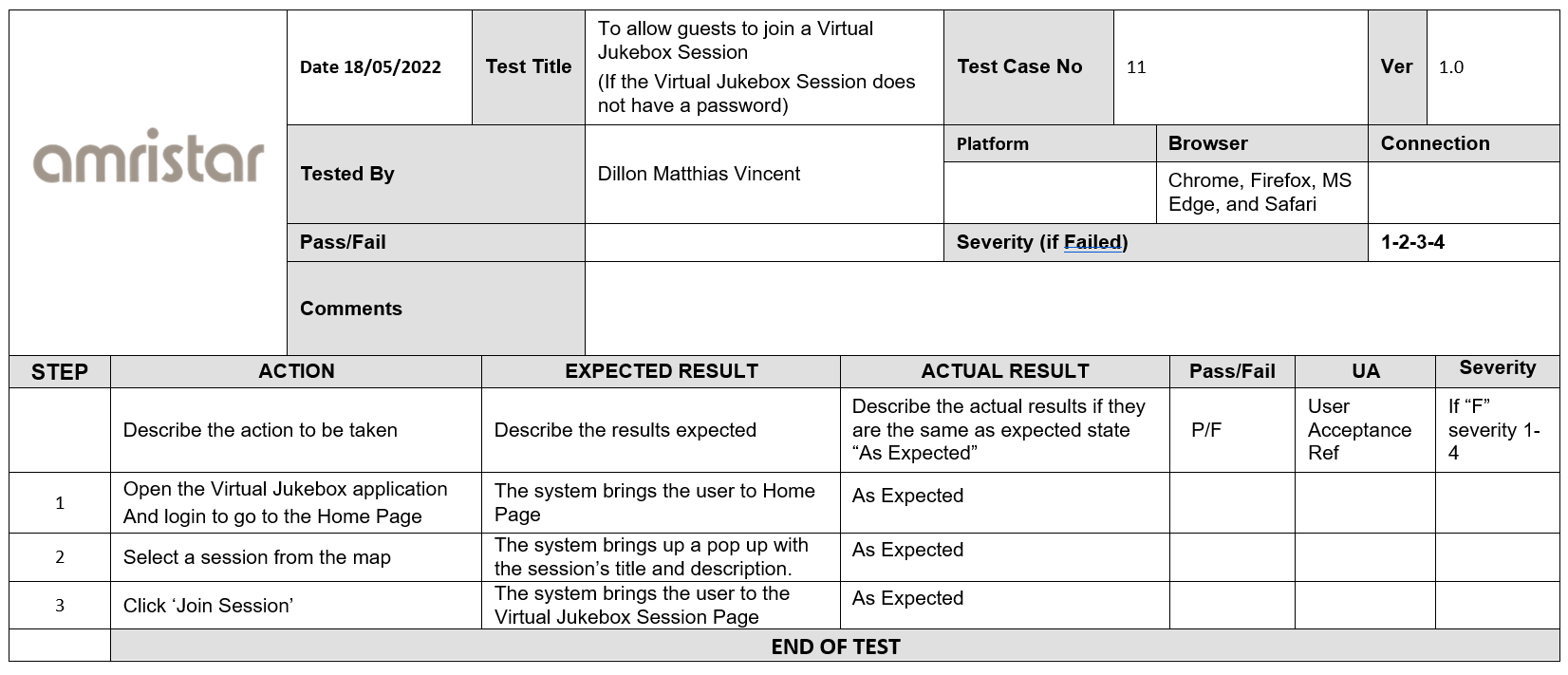
# 

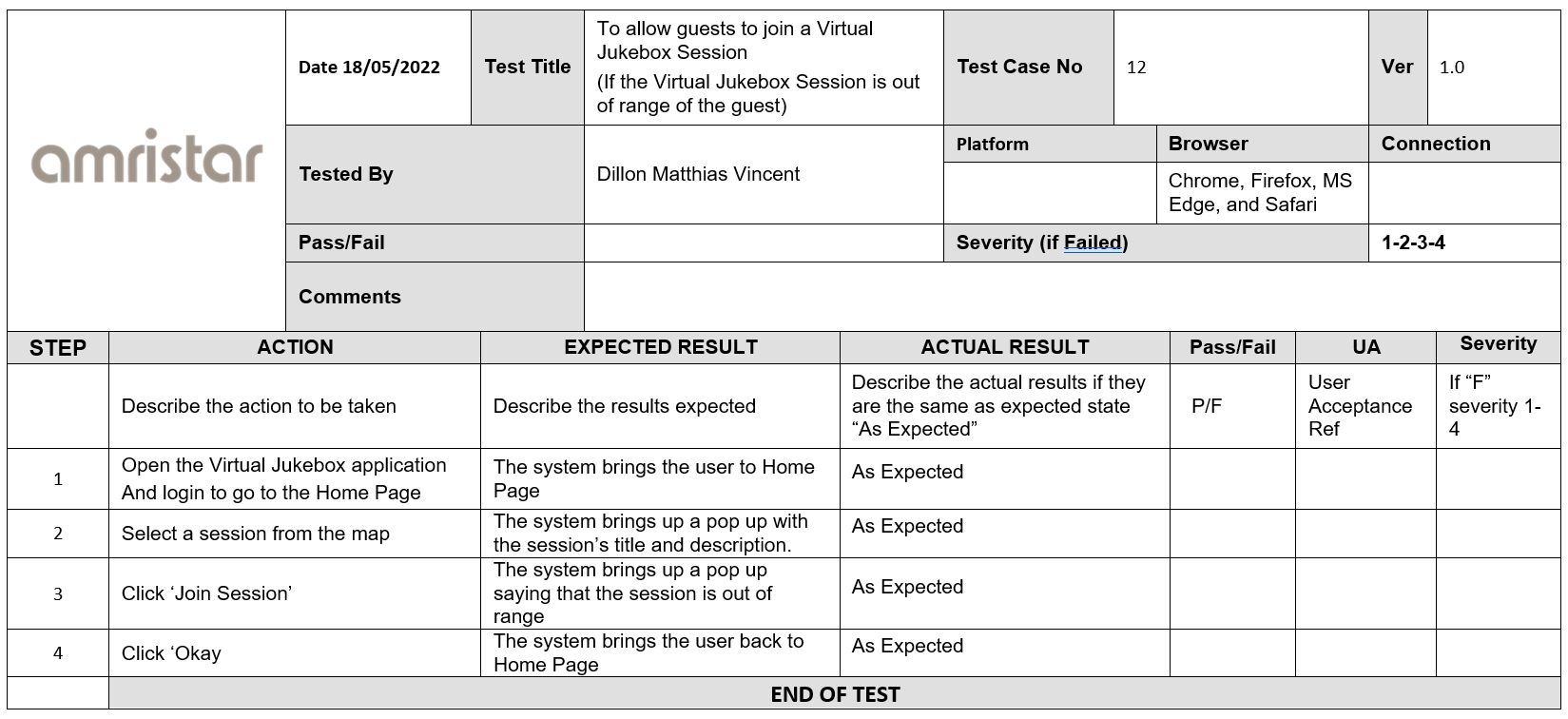
# 

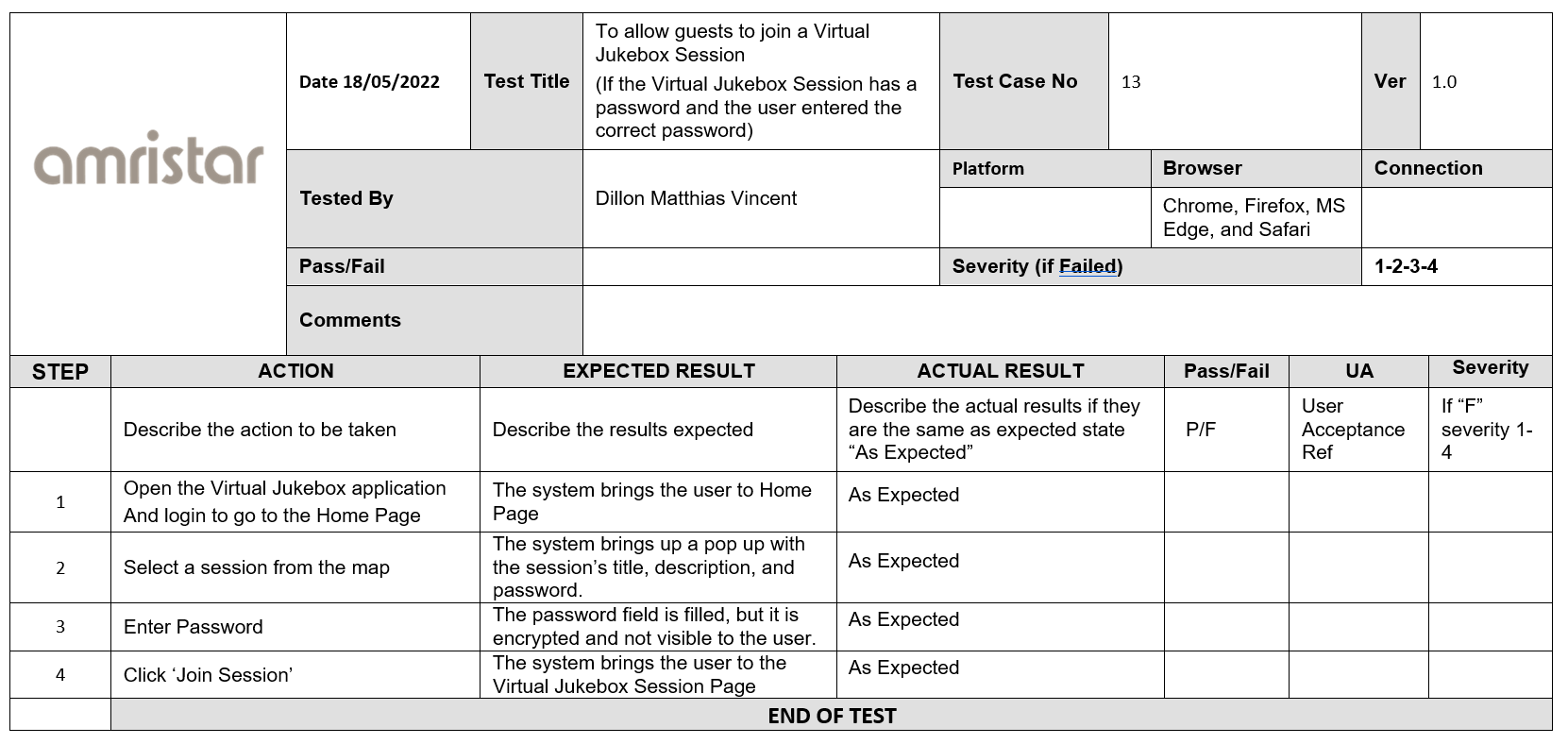
# 

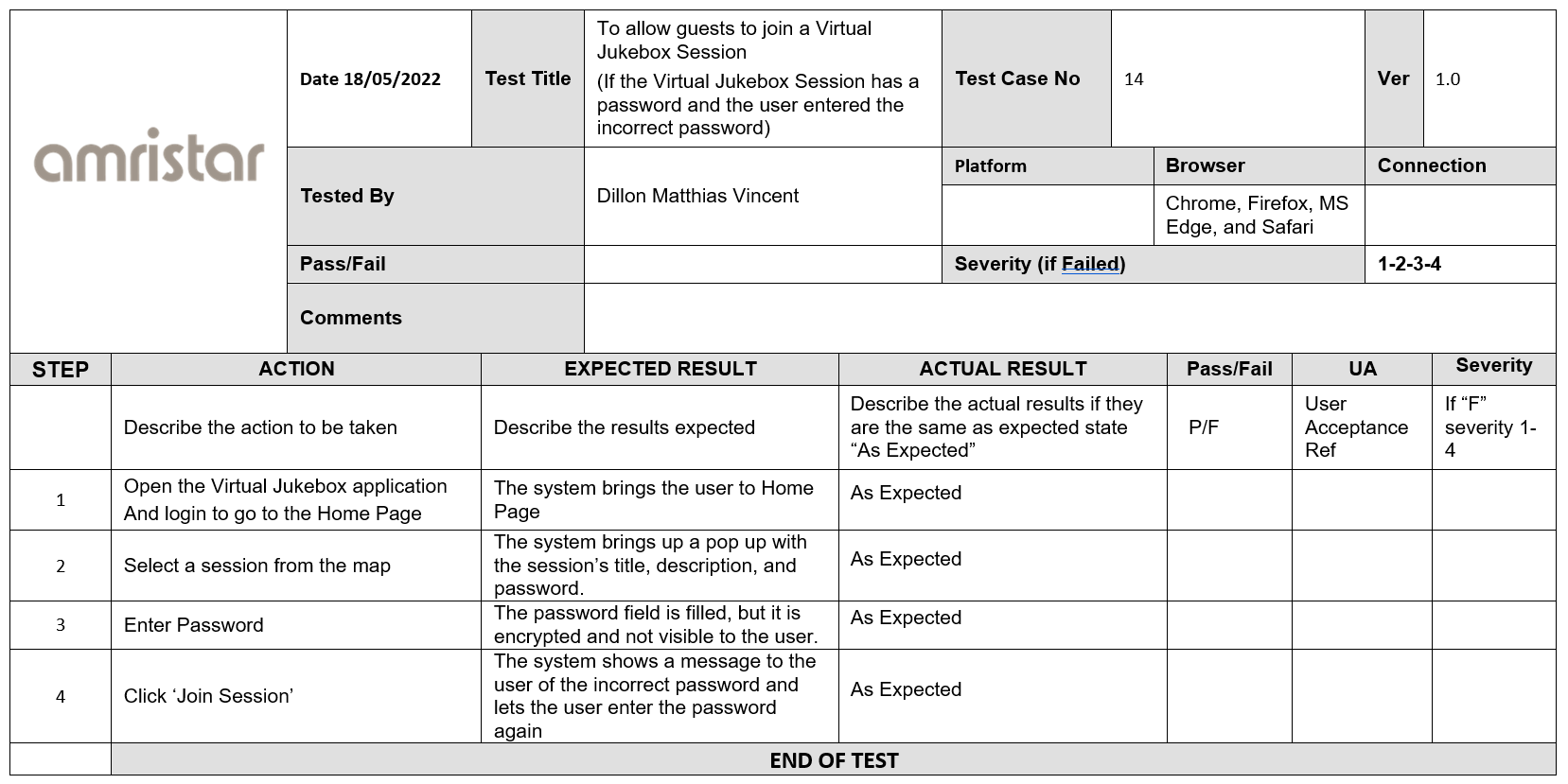


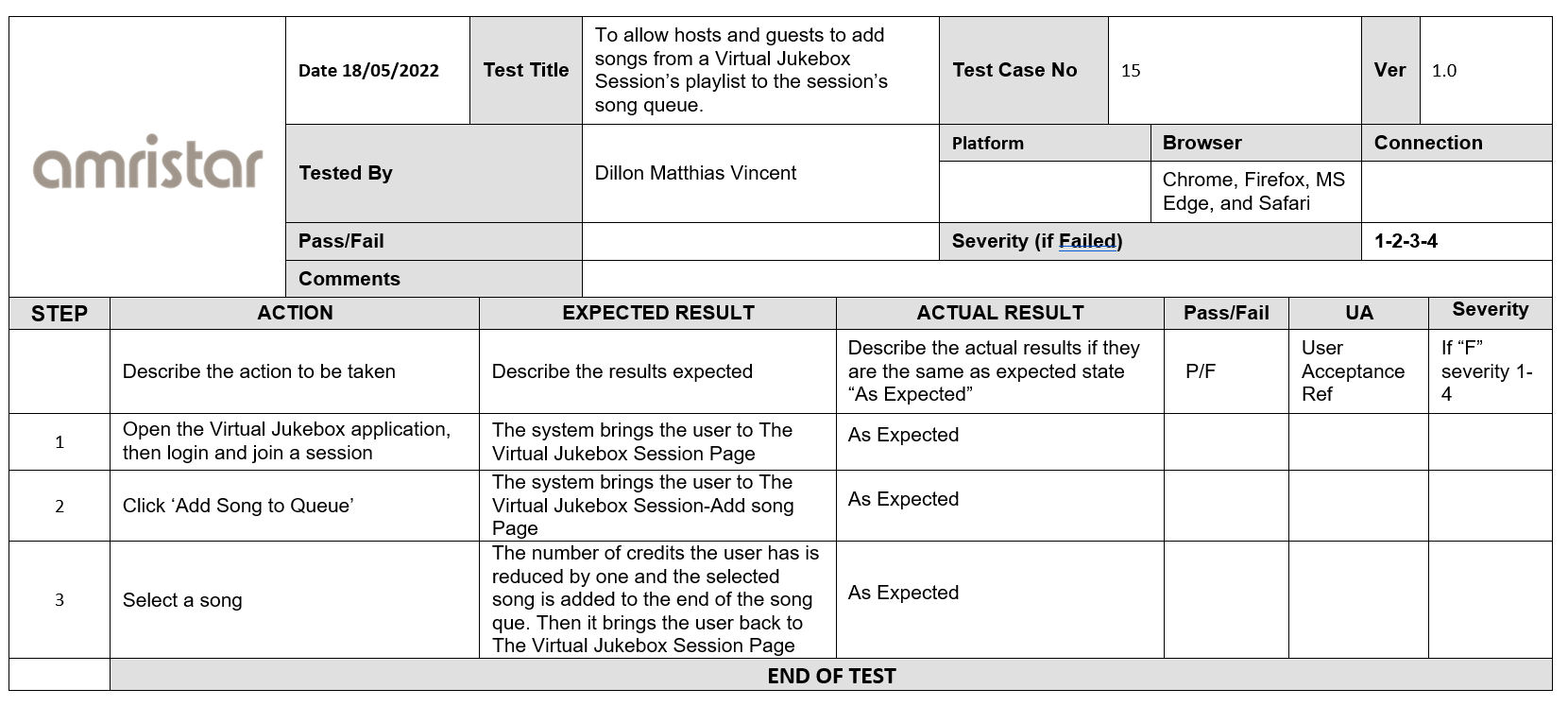


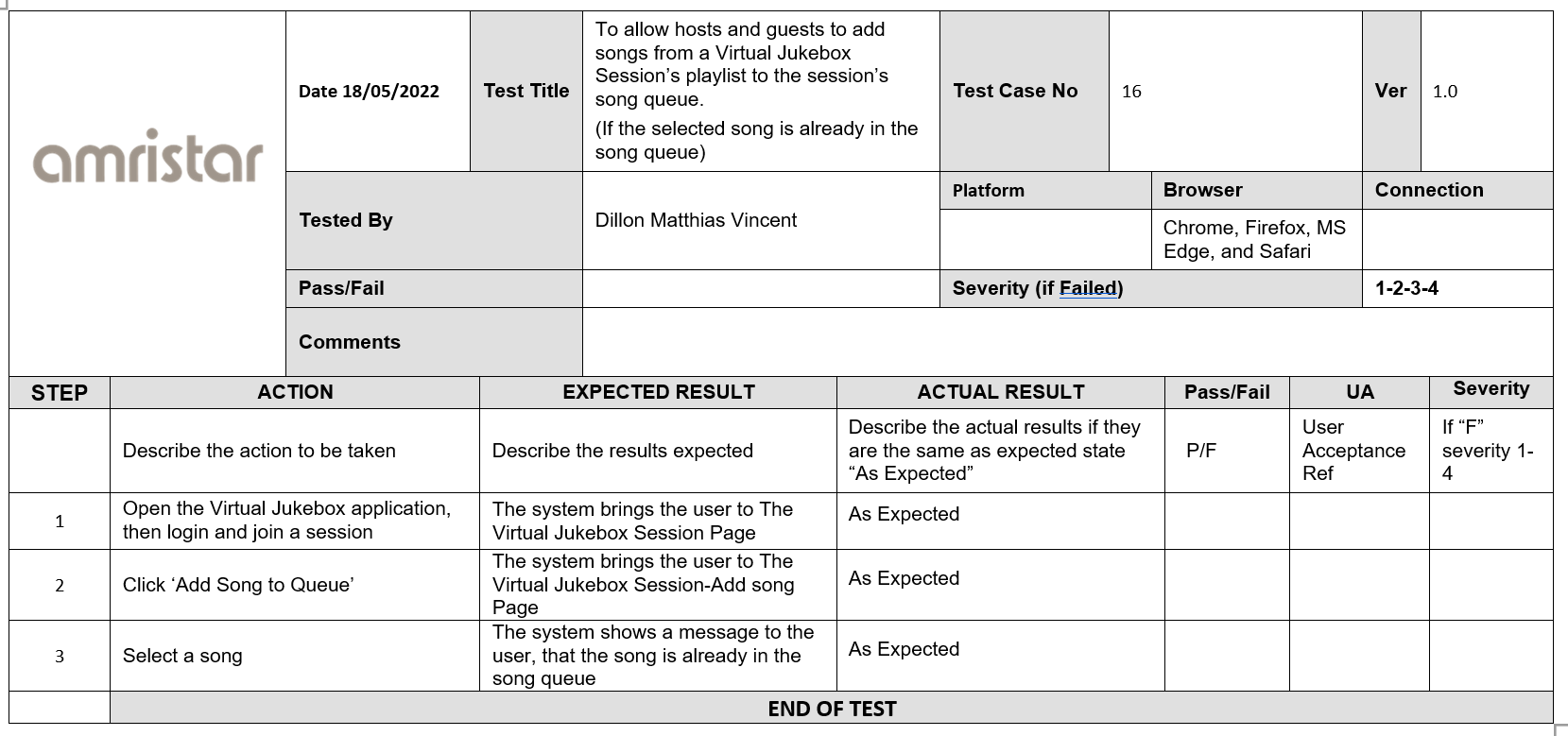


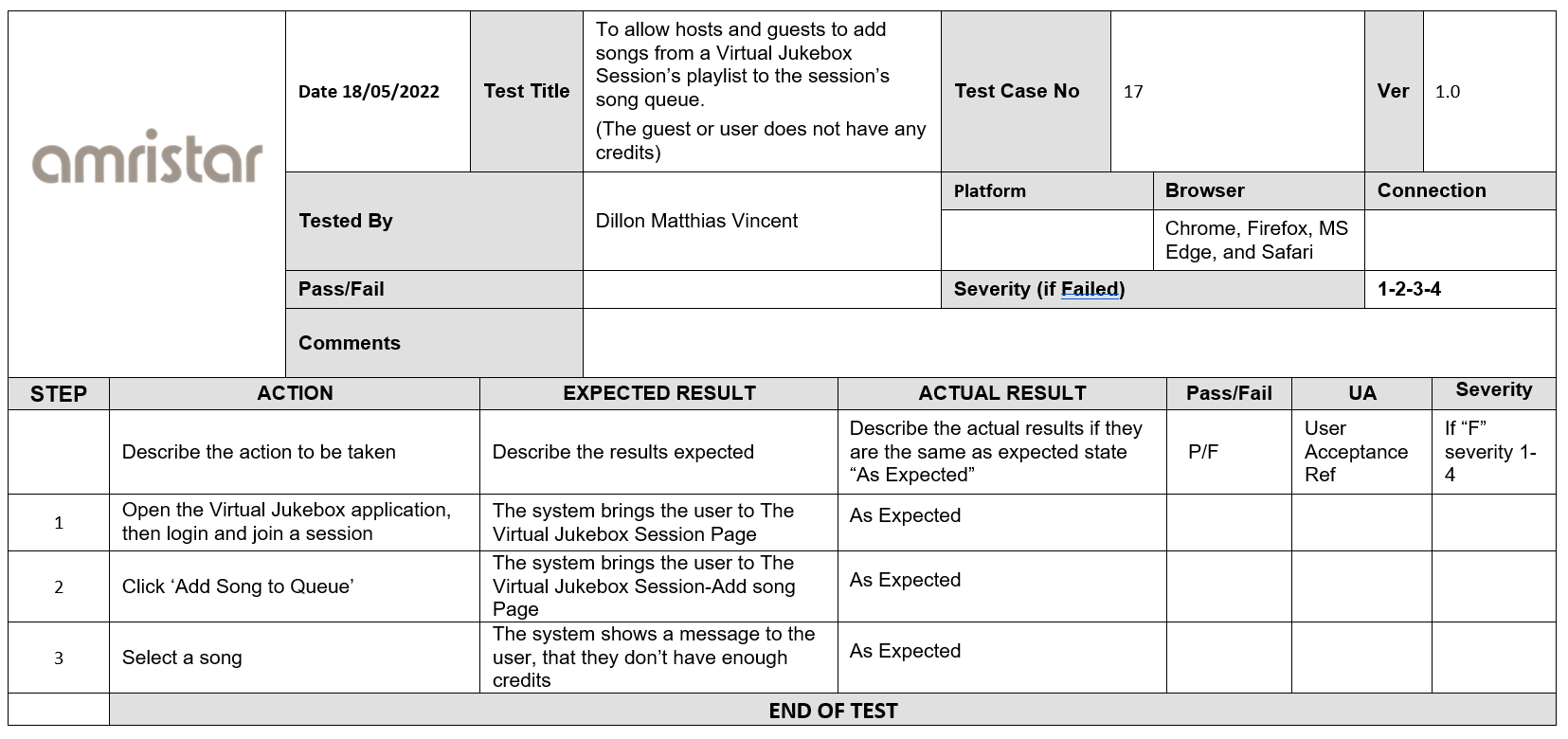


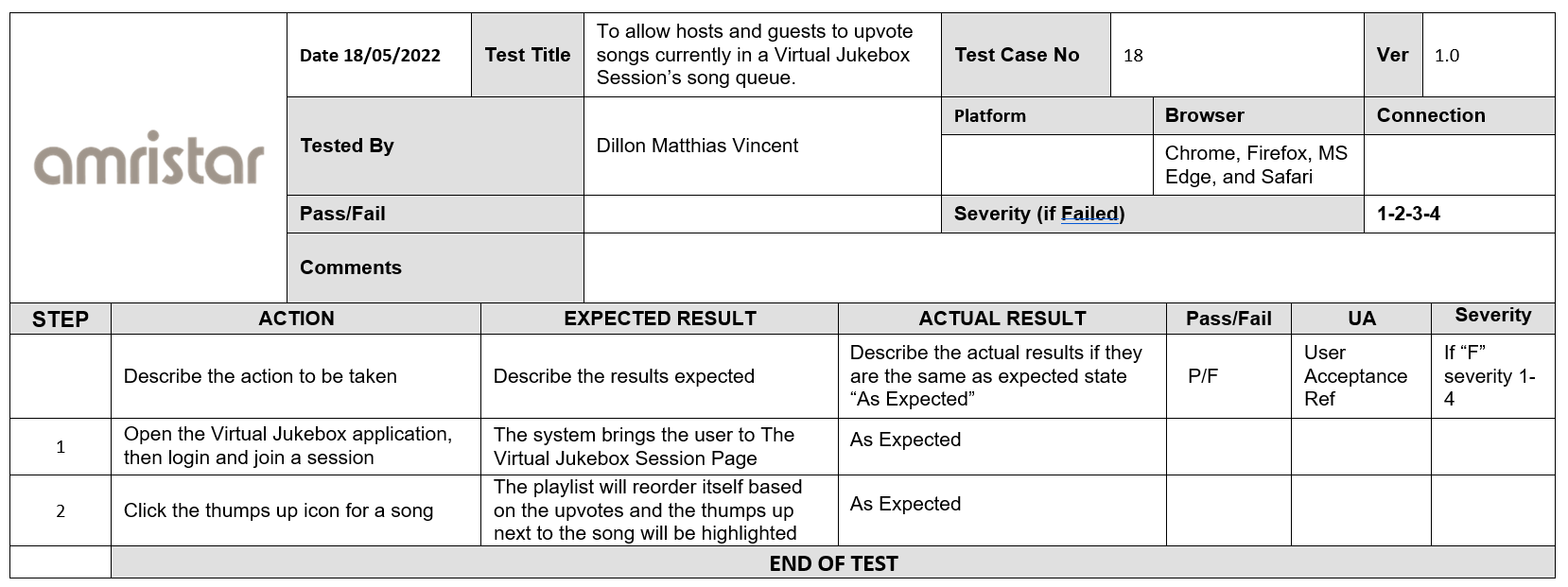


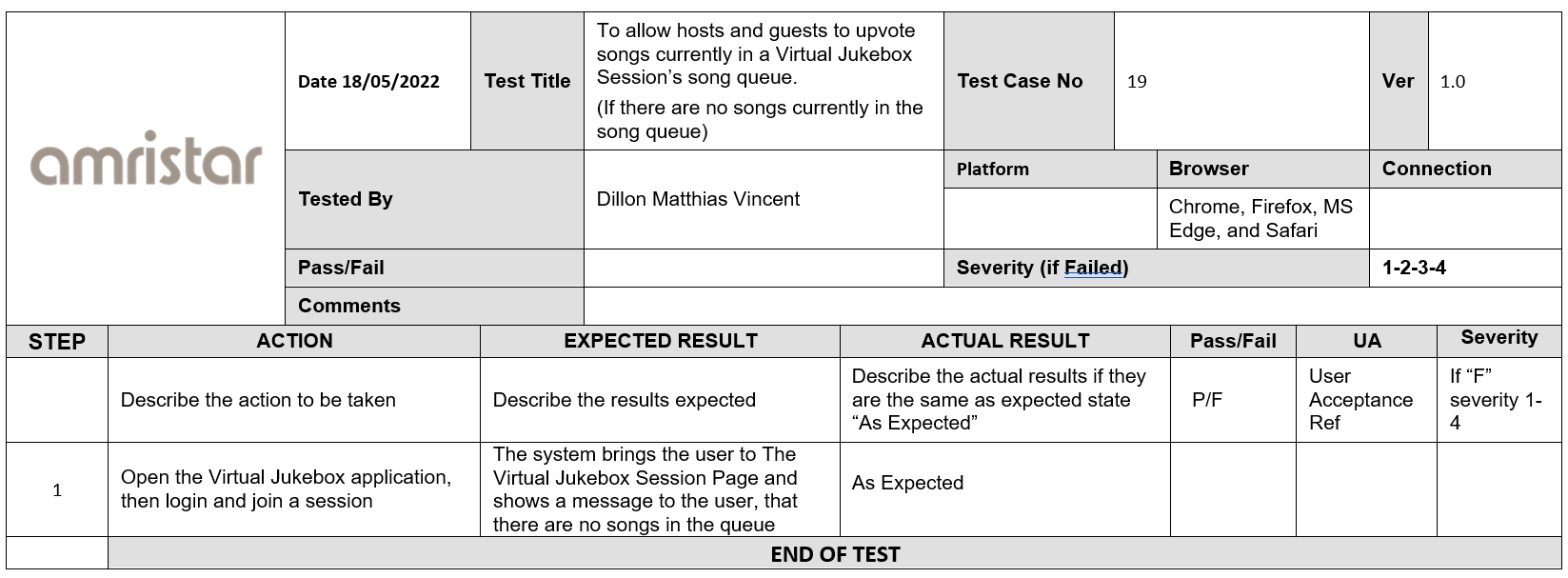


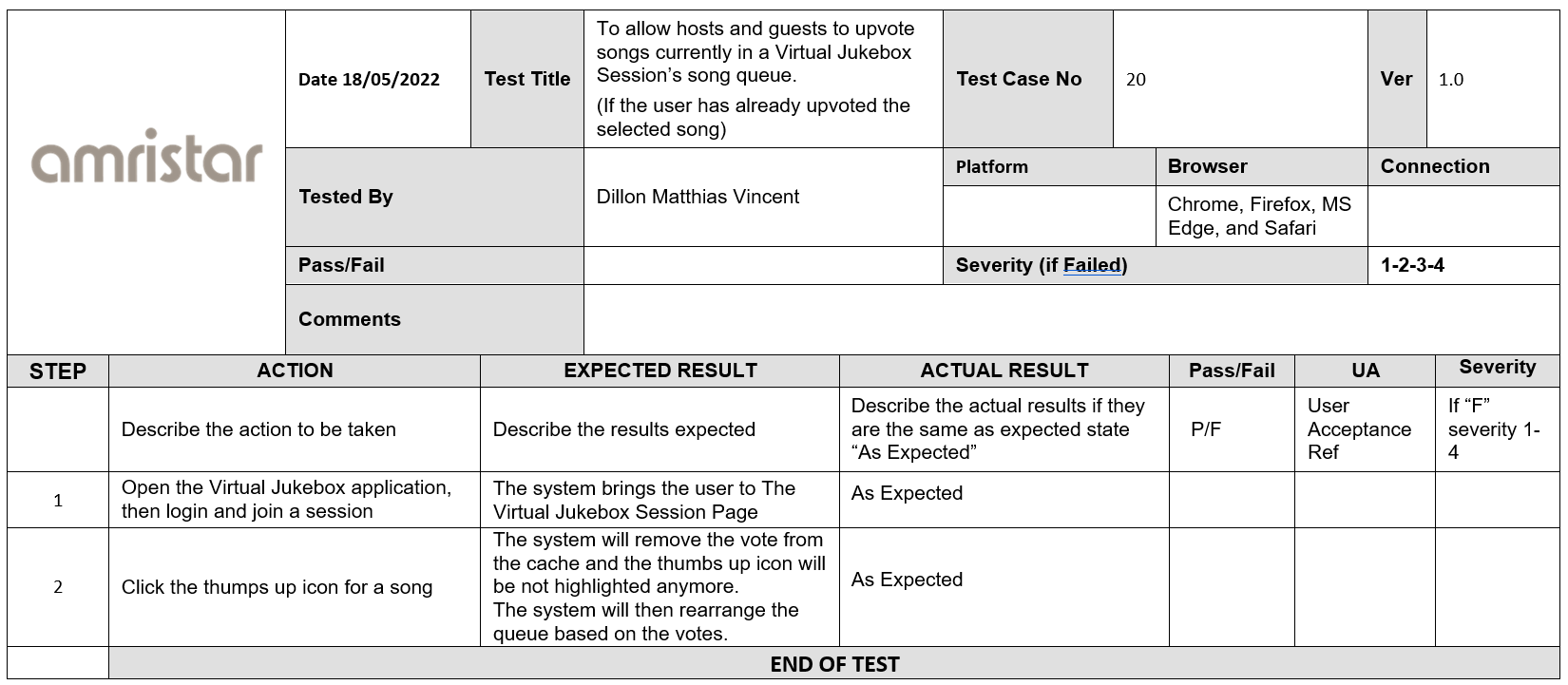


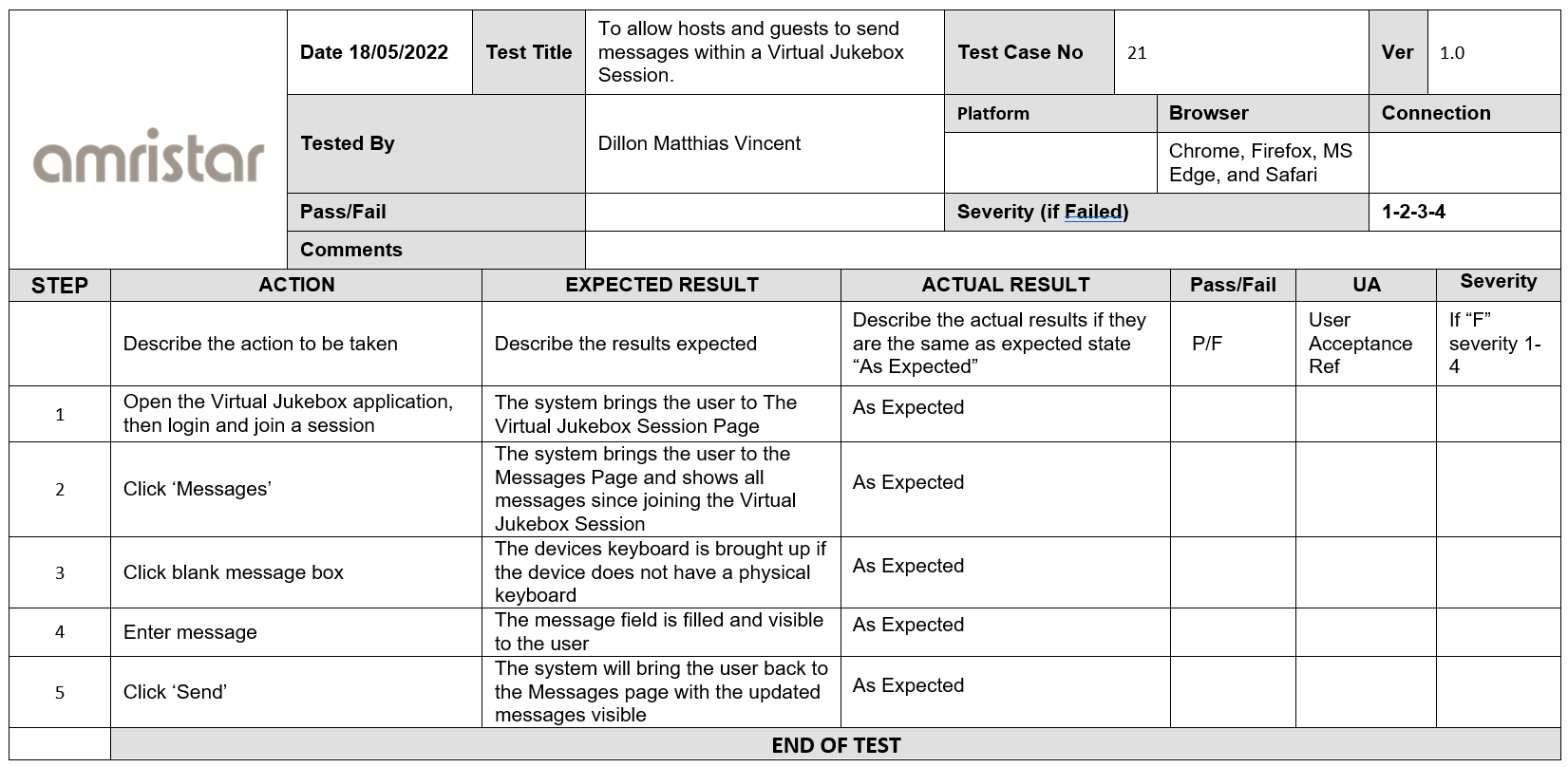


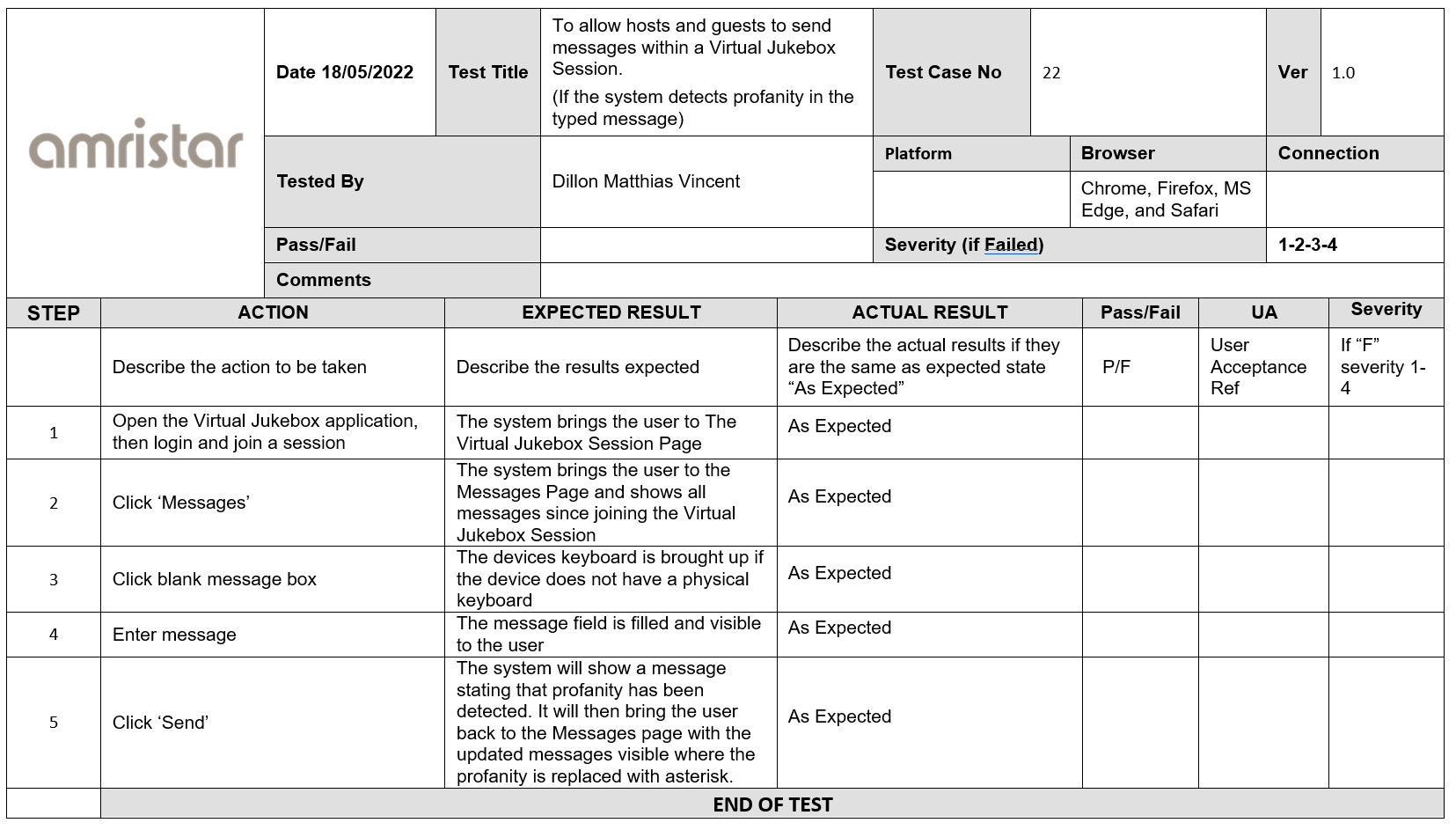


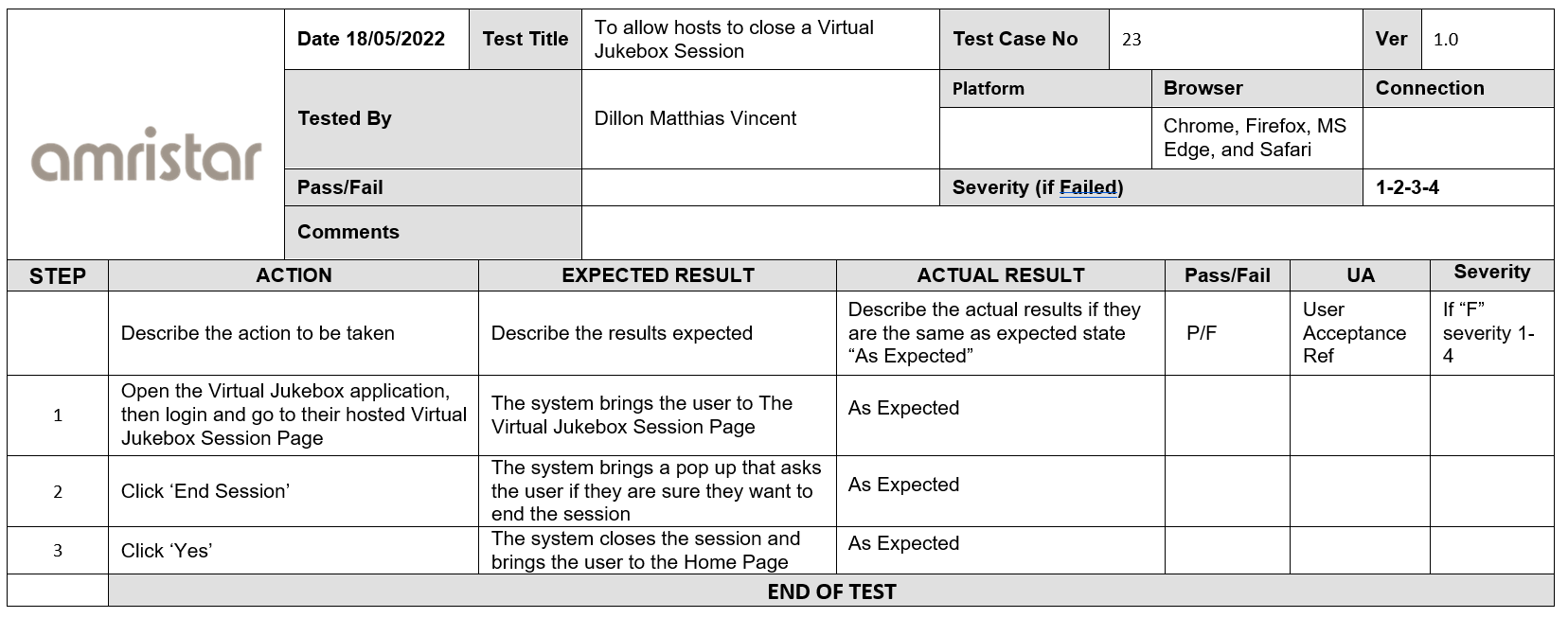












# 9. Phasing, Roadmap, and Implementation Plan

## 9.1 Technical Assumptions

### 9.1.1 User Interfaces

**Mobile and Desktop**

The UI belonging to the application is compatible with both mobile and desktop devices, and additionally, supports a dynamically scalable UI in order to provide a consistent UX between a variety of different mobile and desktop devices.

### 9.1.2 Hardware Interfaces

**Internet Capability**

The User will access the application via an internet-enabled mobile phone, tablet, laptop or desktop computer. In addition, if the User is the Host, they will require access to a speaker (or speakers) connected to their device either internally or externally, to act as an audio output for the application.

### 9.1.4 Communications Interfaces

**Web Browser**

The User will be accessing the Virtual Jukebox application only via either of the latest two publicly released versions of Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari.

**Client–Server Communication**

In addition, client–server communication will be facilitated by Django–for database integration; HTTPS for encrypted information transfer between, namely, the application, User, and the Spotify Platform; and WebSocket to facilitate two-communication between the aforementioned entities.

## 9.2 Solution Phases and Technical Milestones

### 9.2.1 Phase 1: Package 3

**Overview**

The initial phase, Package 3, will consist of two deliverables: the Software Architecture Specification (SAS), and Progress Report 3.

**Software Architecture Specification**

The SAS, with reverence to the SRS, provides a detailed design in regards to the implementation of the application–featuring use cases, walkthroughs, architectural diagrams, an ER diagram, test plans and component descriptions.

**Progress Report 3**

Progress Report 3 indicates the tasks completed by the team, both individually and collectively, during completion of the SAS. Moreover, it also provides an overview of the tasks to be completed during the delivery of the first release.

### 9.2.2 Phase 2: Package 4

**Overview**

The deliverables of Package 4 shall consist of the following: the first release of the application, and Progress Report 4.

**Application: 1st Release**

The first release will address the core requirements outlined in the SRS, without strict prioritisation of implementing a highly functional application. Development will be made with strict adherence to applicable coding standards–namely, Amristar’s coding standard (which shall be provided). Code quality, documentation quality and completeness against targets outlined in previous documents will be the subjects of marking.

**Progress Report 4**

Evidently, Progress Report 4 will outline work done by individuals and the team collectively in delivery of the first release of the application. Thereafter, the report will also detail the work which will be necessary in delivering the second release of the application and provide an updated Project Plan if any changes had been made.

### 9.2.3 Phase 3: Package 5

**Overview**

In the third phase, the following will be delivered: the second release of the application, and Progress Report 5.

**Application: 2nd Release**

The application’s second release will address the next round of requirements–becoming fully usable and useful upon its release. Once more, the subjects of marking will be code quality, documentation quality; and in addition, completed test cases and records detailing what has been done, what remains to be done, and all known issues

**Progress Report 5**

Progress Report 5 will detail the work done by the team towards the second release of the application. Work to be done in delivery of the third and final release will be outlined and an updated Project Plan may be provided if any changes had been made.

### 9.2.4 Phase 4: Package 6

**Overview**

The fourth and final phase, Package 6, shall consist of the following deliverables: the third and final release of the application, the Post-Implementation Evaluation, and Progress

Report 6.

**Application: 3rd (Final) Release**

The final release of the application will include final bug fixes and implement a limited set of new features. The focus of this release is to create a stable application fit for use by potential end users. Similar to the first and second releases, the focuses of marking will be code quality, documentation quality and completed test cases. In addition, the completeness and quality of technical documentation will also be assessed.

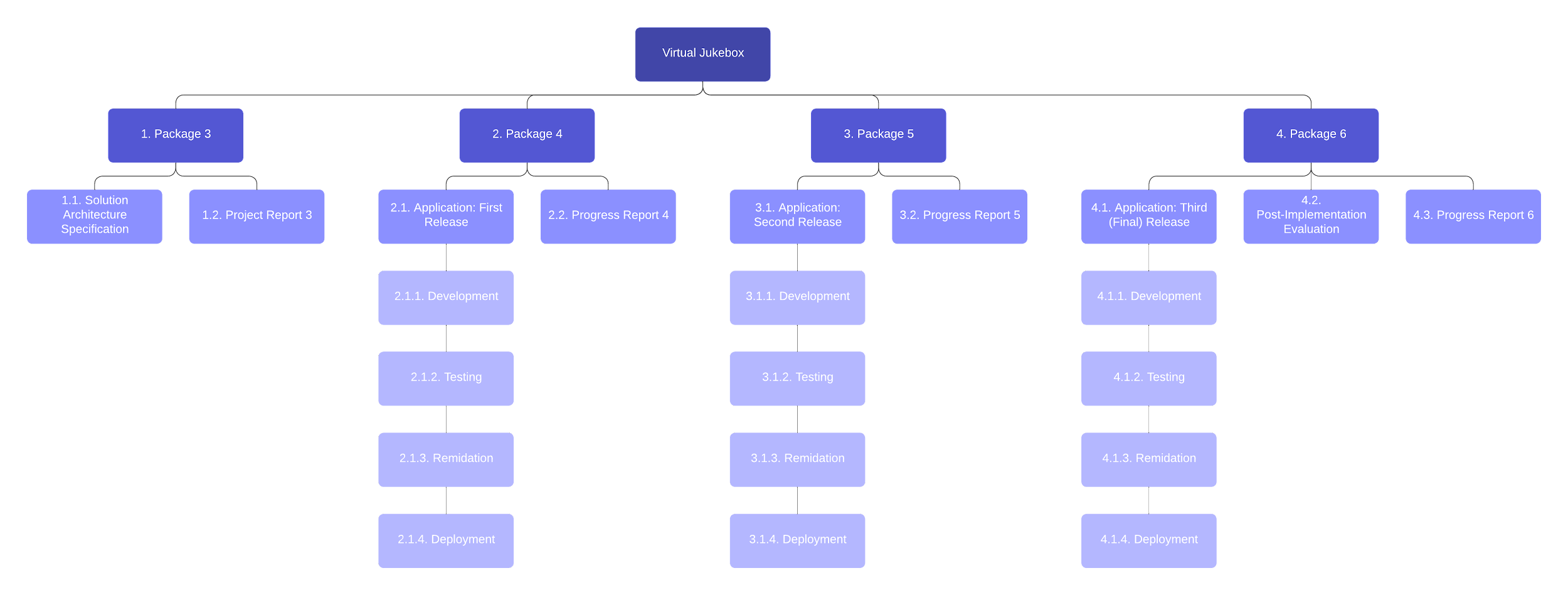
**Post-Implementation Evaluation**

In the Post-Implementation Evaluation of the final release, the team will reflect upon (i) what the final application has contributed; its commercial viability and potential benefits; and lessons learnt. The outcomes of this evaluation will also be included in the final presentation.

**Progress Report 6**

Progress Report 6 will act as a closure report–describing activities completed for the application and the project overall; highlights and achievements; and the activities and objectives which were not met.

## 9.3 Technical Work Breakdown Structure



## 9.4 Implementation Instructions

### 9.4.1 Coding Standards and Guidelines

**Vue**

The Vue.js Style Guide [[link](https://v2.vuejs.org/v2/style-guide/)] will act as the primary reference in regards to developing the frontend of the application.

**Spotify**

The Spotify for Developers Design Guidelines [[link](https://developer.spotify.com/documentation/general/design-and-branding/)] will ensure that the Spotify platform is integrated into the application while “respecting [Spotify’s] brand and legal/licensing restrictions. (Spotify, n.d.)”

**Django**

The Django JavaScript Coding Style Guide [[link](https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/javascript/)] will promote maximum interoperability between Django and Vue.js.

**Redis**

The Redis Manual [[link](https://redis.io/docs/manual/)] is “the developer’s guide to Redis” and offers a plethora of information, guides and advice in regards to all of the core functionality of the Redis platform.

**PostgreSQL**

The SQL Style Guide [[link](https://www.sqlstyle.guide/)] is an unofficial but no less complete style guide for using PostgreSQL and all other SQL-based languages.

### 9.4.2 Documentation Standards

Within the repository, README.md (Markdown) documents will be written where necessary to provide stakeholders with relevant information–namely, introduction, explanation, and usage–in regards to various components.

### 9.4.3 Repository Standards

The git Reference Guide [[link](https://git-scm.com/docs)] will be utilised in all manners concerning version control and codebase management, including: branching and merging; sharing and updating code; and inspection and comparison.

# 10 Appendix

TODO?

# 11 References

TODO

# 

# 

# 

# 

# 

# 

# 