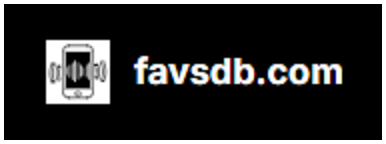


Phillip Sweeney

phillipjsweeney@gmail.com

(203) 820-4666

This document provides a reverse-engineered system design and documentation for your FAVSDB application, based on the provided Node.js code, requirements, and database schema files.

Diagrams rendered using Mermaid syntax, generated by Gemini.

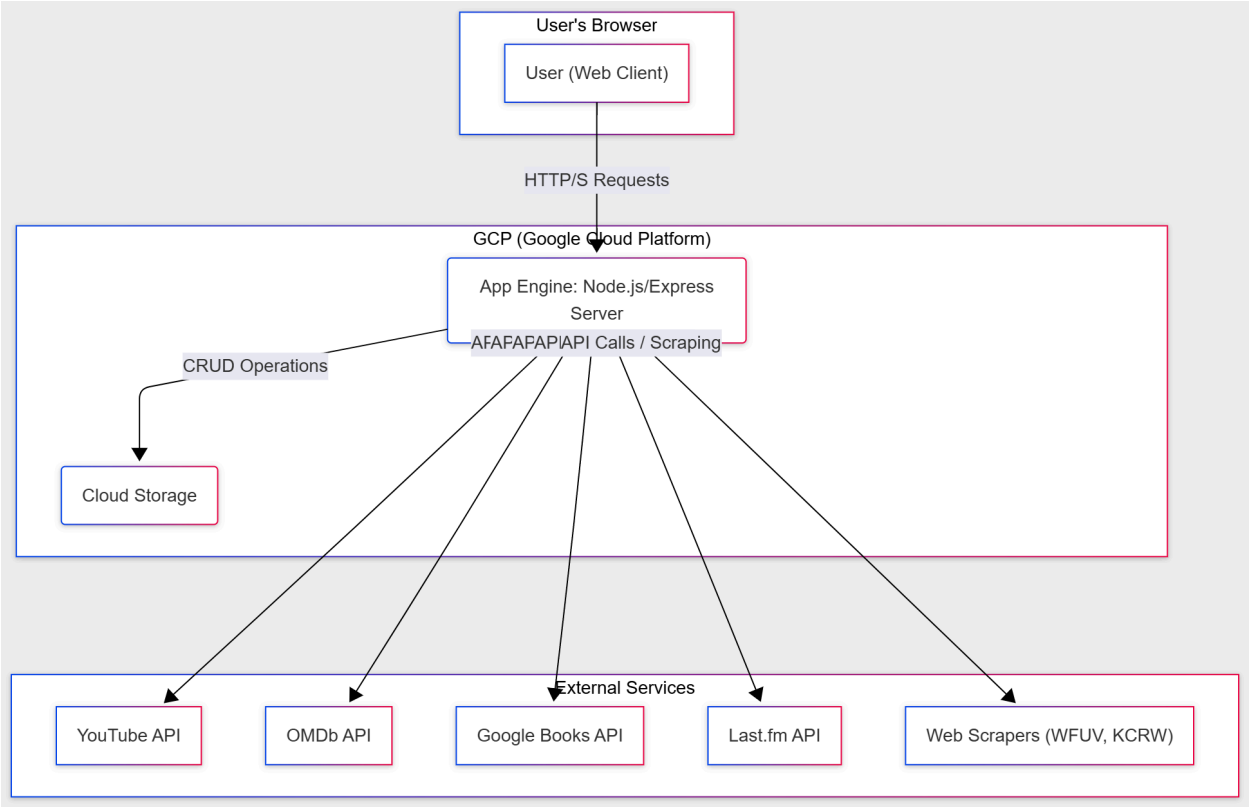
---

1. Software Overview and Architecture

The FAVSDB application is a modern three-tiered API-driven web application hosted on the Google Cloud Platform (GCP) App Engine Standard environment.

Component	Technology / Role	Key Files
Frontend (Presentation Tier)	HTML/CSS/JavaScript (Runs in browser)	index.html, app-dashboard-*.html
Backend (Application/API Tier)	Node.js with Express	server.js, package.json
Data Layer (Persistence Tier)	Hybrid "Filesystem-as-DB" (GCP Filesystem)	db.json, db-data-cache.json, db-data-user.json
Data Sources	External APIs & Web Scraping	_config.js, axios, cheerio, yt-search

The server handles routing, session management (express-session), application logic, data enrichment (using external APIs), web scraping, and file-based persistence. The core philosophy of the application is centered on open data sharing of user favorites and providing transparency.



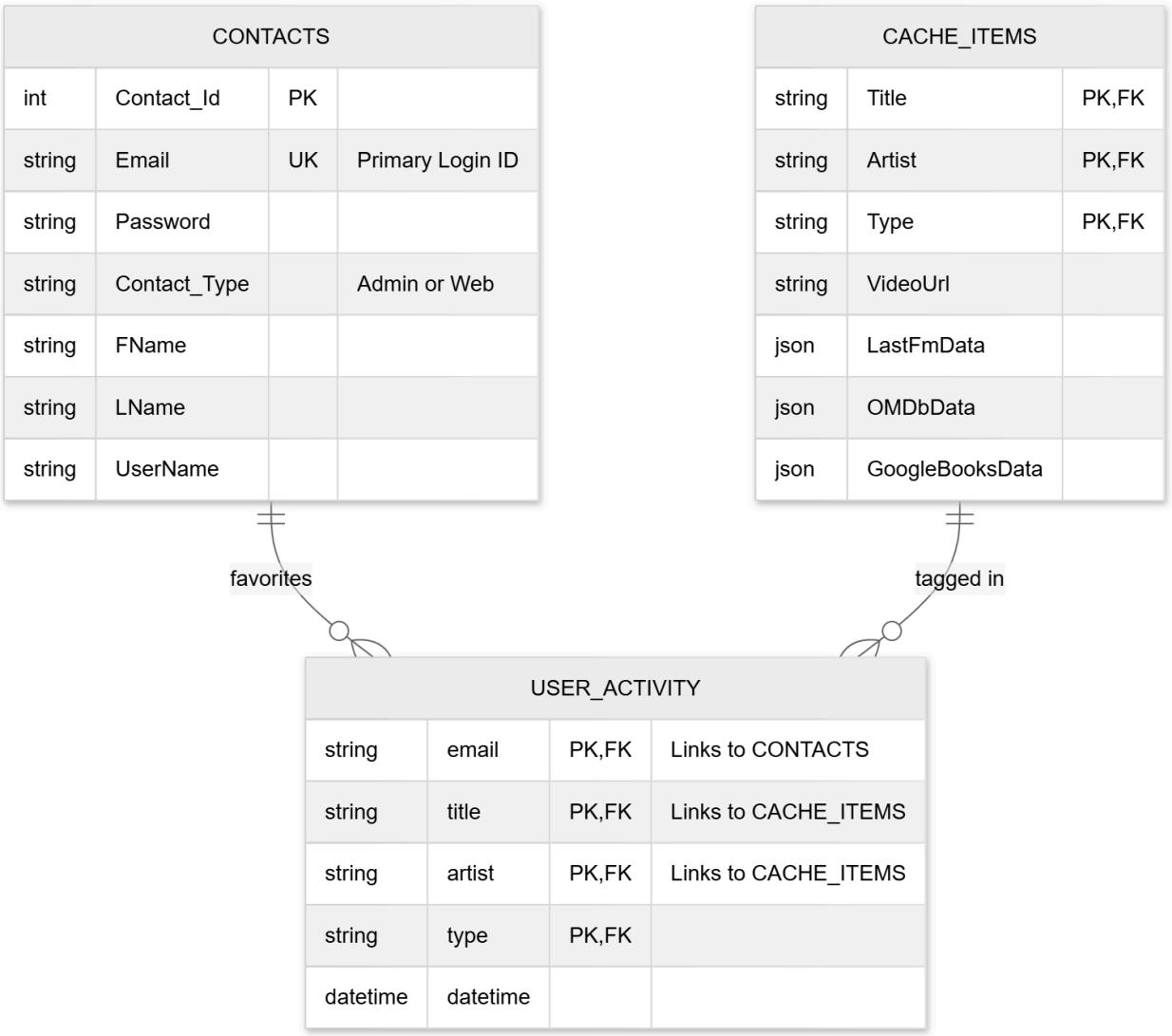
---

2. Entity-Relationship Diagram (ERD) and Schema

The application currently uses a non-relational, document-based storage structure (JSON files) but models data relationships similar to a normalized relational database, relying on keys (IDs/Emails) and composite identifiers for linking.

Logical Data Model (Mermaid ER Diagram)

The data model consists of three core logical entities: Contact, CacheItem, and UserActivity (Link Table).



Data Dictionary: Fields and Keys

The application utilizes three main logical data sets. The persistence location of these data sets is deliberately kept separate (db.json vs. db-data-*.json) based on the requirement to keep them recognizable as simple files.

2.1. Contact (Stored in db.json)

This entity manages user/customer access and metadata. The primary key used for login logic is the Email address.

Field Name	Data Type (JSON/SQL Schema)	Key / Constraints	Description / Source

Contact_Id	**INT**	**Primary Key (SQL Schema)**	**Unique auto-incrementing ID (internal use).**
Email	**VARCHAR(255)**	**Unique Key (Logical PK for Login)**	**User's unique identifier and login credential.**
Password	**VARCHAR(255)**	**(Login Credential)**	**Hashed or plain text password (can be null for "Web" users).**
FName / LName	**VARCHAR(255)**		**User's first and last name.**
UserName	**VARCHAR(255)**		**Derived from email address for new "Web" users.**
Contact_Type	**VARCHAR(255)**		**"Admin" or "Web" (Standard user).**

2.2. CacheItem (Stored in db-data-cache.json)

This is the central repository for all fetched and enriched media data.

Field Name	Data Type (JSON)	Key / Constraints	Description / Source
Title	**String**	**Composite Key (Part 1)**	**Title of the item (Song, Film, Book).**

Artist / Author	**String**	**Composite Key (Part 2)**	**Creator of the item.**
Type	**String**	**Composite Key (Part 3)**	**"MUSIC", "FILMS", or "BOOKS".**
VideoUrl	**String**		**YouTube URL for trailers/music videos.**
LastFmData	**JSON Object**		**Full raw data payload from the Last.fm API for MUSIC items.**
OMDb/GoogleBook	**JSON Object**		**Expected similar fields for Films/Books (implied by API calls).**

2.3. UserActivity (Link Table, Stored in db-data-user.json)

This acts as a join table linking a user's email to a specific item in the cache.

Field Name	Data Type (JSON)	Key / Constraints	Description / Source
email	**String**	**Foreign Key**	**User's email (links to Contact).**
title	**String**	**Foreign Key (Part 1)**	**Item title (links to CacheItem).**
artist	**String**	**Foreign Key (Part 2)**	**Item artist/author (links to CacheItem).**

type	**String**		**Item type.**
datetime	**String**		**Timestamp of when the item was tagged by the user.**

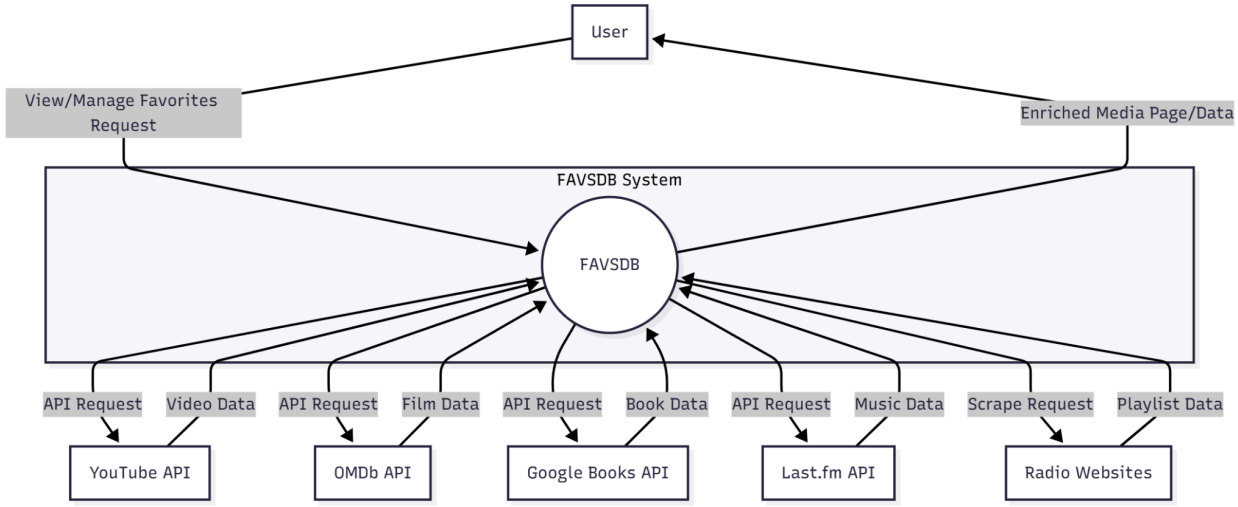
---

3. Data Flow and API Endpoints

Data Flow Diagram (DFD)

This DFD shows the high-level process flow for a user-initiated data import (like the Shazam uploader or scraping trigger) and the subsequent API-driven data retrieval.

Code snippet



Key API Endpoint Logic

API Endpoint	HTTP Method	Action / Purpose	Data Sources / Persistence Used

/api/auth/login	POST	Authenticates regular users against db.json:contacts.	db.json (Read/Update Session)
/api/shazam-user	POST	Handles login/onboarding for uploaders, creating a user (Contact_Type="Web", no password) if they don't exist.	db.json (Read/Write)
/api/save-music-cache	POST	Saves/merges enriched data items into the central data store. Key Logic: Upsert based on `Artist	Title` composite key.
/api/log-user-activity	POST	Logs which user (email) tagged which item (title/artist).	db-data-user.json (Append)
/api/user-data	GET	Main retrieval endpoint. Joins data from db-data-user.json (which items belong to the user) with db-data-cache.json (the full item metadata). Supports JSON, CSV, and HTML output.	db-data-user.json, db-data-cache.json (Read)
/api/scrape/wfuv	GET	Triggers scraping of WFUV, then fetches additional track info from Last.fm to enrich the data.	External (WFUV, Last.fm)
/api/yt-search*	POST	Searches YouTube for the corresponding video URL (for playback/trailers).	External (YouTube)

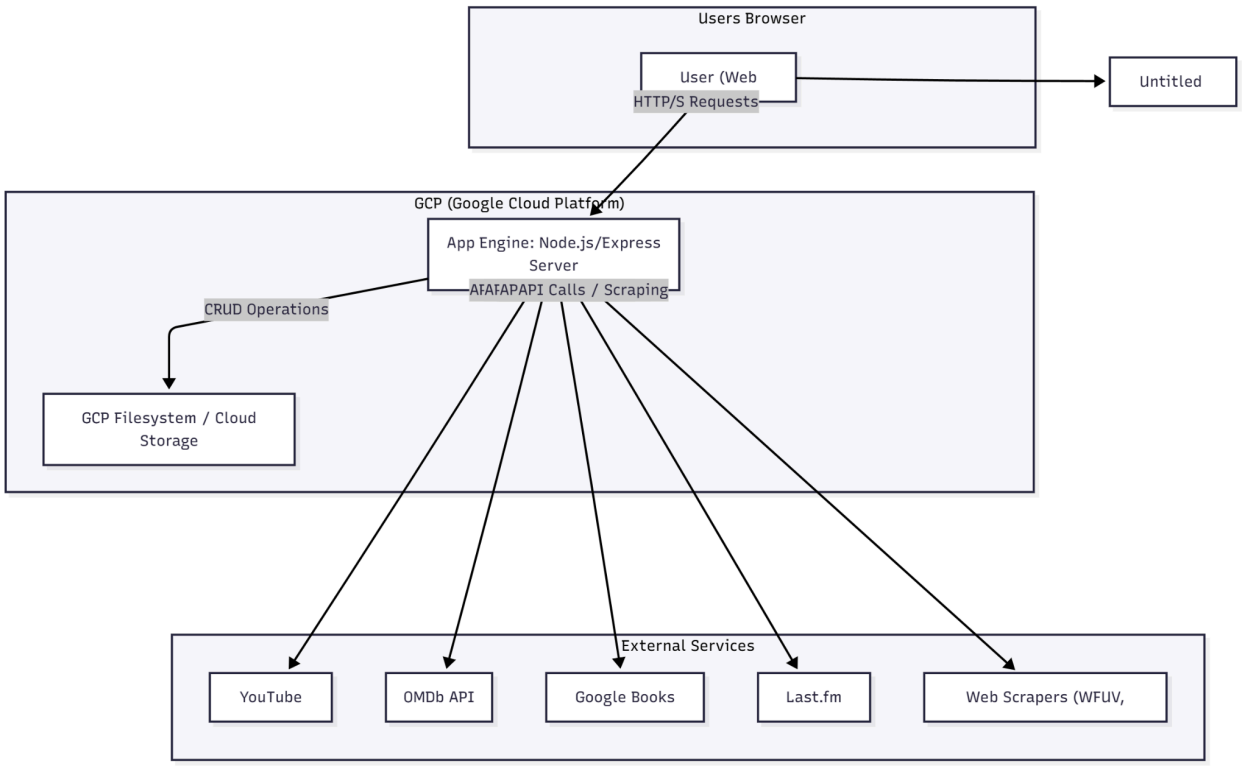
---

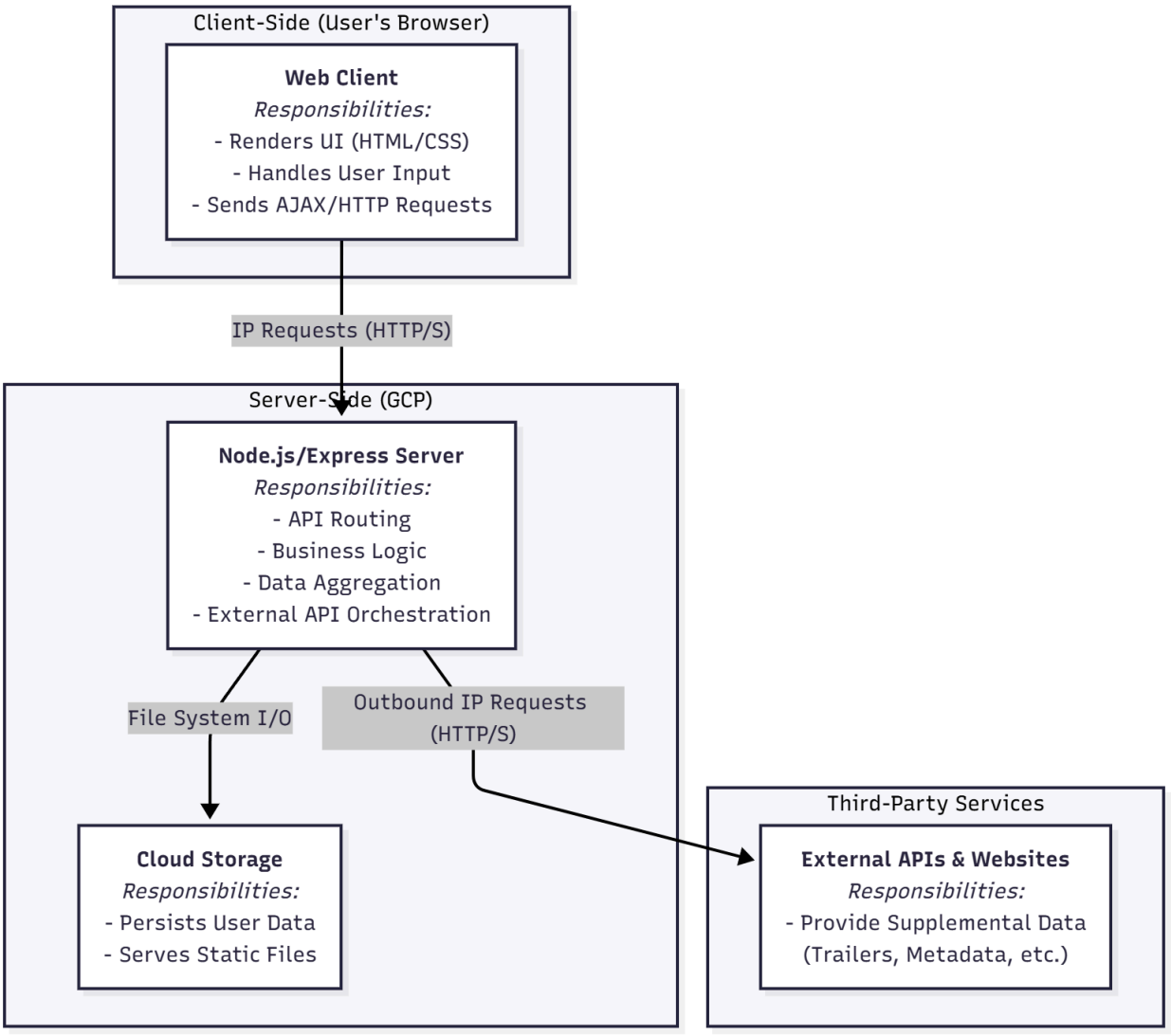
4. System and Deployment Overview

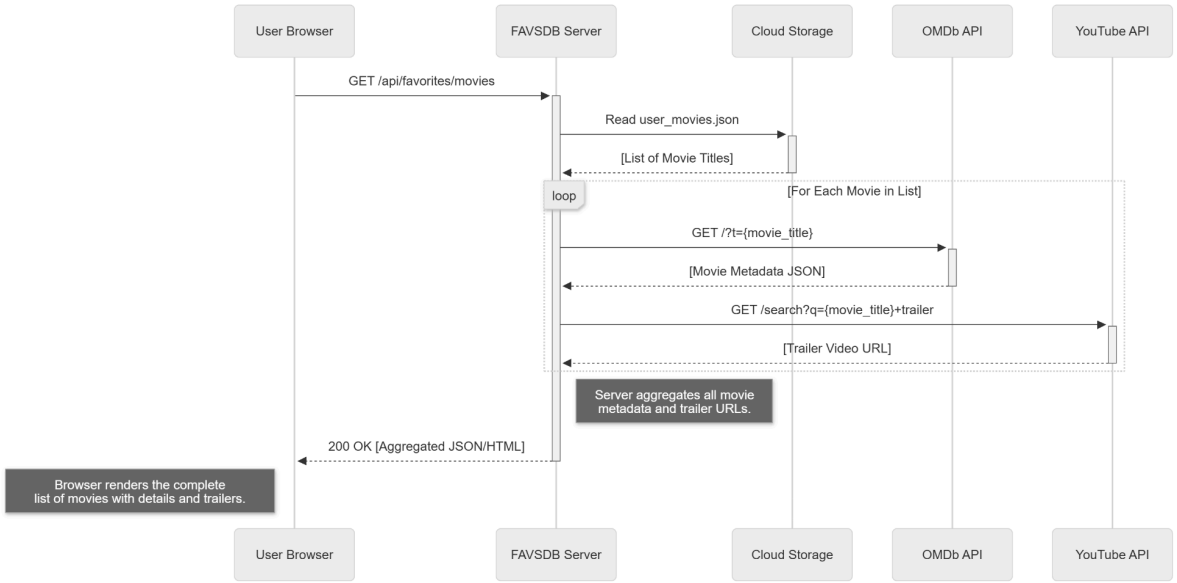
Networking and Infrastructure Design (Text-Based Mockup)

The system design focuses on a minimal setup in the Google Cloud Platform (GCP) environment, adhering to the requirement for a text-based, visually-oriented networking representation.

Code snippet




Client-Side (User's Browser)
Web Client
Responsibilities:
- Renders UI (HTML/CSS)
- Handles User Input
- Sends AJAX/HTTP Requests
IP Requests (HTTP/S)
Server-Side (GCP)
Node.js/Express Server
Responsibilities:
- API Routing
- Business Logic
- Data Aggregation
- External API Orchestration
File System I/O
Outbound IP Requests (HTTP/S)
Cloud Storage
Responsibilities:
- Persists User Data
- Serves Static Files
Third-Party Services
External APIs & Websites
Responsibilities:
- Provide Supplemental Data (Trailers, Metadata, etc.)



Key Business Rules and Design Decisions

The requirements document outlines several key business and design decisions enforced by the code:

1. **Identity: A user's identity is uniquely defined by their Email address.**
2. **Onboarding (Shazam Uploader): Users created via the upload process automatically receive the Contact_Type of "Web", and their Password field is set to null. This implies they cannot use the standard password login form unless an admin adds a password later.**
3. **Data Integrity/Keying: The unique identifier for a piece of content (a CacheItem) is a composite key consisting of Title, Artist/Author, and Type. All update and merge logic relies on this triplet.**
4. **Data Loading Logic: When a user accesses a dashboard (app-dashboard-*.html), the system first attempts to load their personalized favorites from the db-data-user.json file. If they have no saved favorites (or are not logged in), a pre-defined default list is loaded (e.g., db-data-music-default.json for the music dashboard).**
5. **Data Enrichment: When new data is scraped (e.g., WFUV playlist), the core track list (Artist, Title) is immediately enriched using secondary APIs (e.g., Last.fm) before being stored in the db-data-cache.json.**
6. **Future Data Storage (Planned): The current filesystem-based storage is recognized as temporary, with a plan to migrate the data files to Google Cloud Storage Buckets.**