Design Database structure

Status	Chosen for development
Assign	Ⓐ Aaron Li
Figma	
Completed Phases	

1. Introduction

Deciding how customer information will be stored can be tricky. There are many factors to consider such as scalability, structured or unstructured data, and query patterns. This doc will provide the structure for the database on storing data.

2. Glossary

Relational Database	A relational database is a type of database that organizes data into tables, where each table consists of rows and columns. It uses a structured approach to store and manage data, with relationships established between tables using keys. This allows for efficient retrieval, manipulation, and analysis of data through the use of SQL (Structured Query Language) queries, providing a flexible and scalable solution for data storage and retrieval in various applications.
Non-relational Database	A non-relational database, also known as a NoSQL (Not only SQL) database, is a type of database that does not rely on the traditional tabular structure of relational databases. Instead, it offers a flexible data model that can store and retrieve unstructured or semi-structured data. Non-relational databases are designed to handle large volumes of data, and they often prioritize scalability, high performance, and horizontal scaling across distributed systems. They are commonly used in applications with rapidly changing data requirements, such as big data analytics, real-time data streaming, and content management systems.

Time Series Database	• Time series databases are specifically designed for handling time-stamped data and optimizing storage and analysis of such data. They excel at efficiently ingesting, storing, and querying data points associated with timestamps. Time series databases are commonly used in applications that involve monitoring, sensor data analysis, financial data analysis, IoT devices, and any scenario where analyzing trends and patterns over time is crucial.
ACID acronym	Atomicity ensures that a transaction is treated as a single, indivisible unit of work. It means that either all the changes made within a transaction are committed to the database, or none of them are. There is no partial execution. Consistency ensures that a transaction brings the database from one valid state to another. It enforces predefined rules and constraints, ensuring that the data remains consistent throughout the transaction. Isolation ensures that concurrent transactions do not interfere with each other. Each transaction operates in isolation from others, and the intermediate state of a transaction is not visible to other transactions until it is committed. Durability ensures that once a transaction is committed, its changes are permanently saved and will survive any subsequent failures, such as power outages or system crashes.

3. Problem

- This solution should anticipate for all future features and functionalities
- May need to consider incorporating multiple types of databases

4. Solutions

4.1 Relational Database

This solution uses relational database to remedy the problems. Looking at this application’s intended use, a rigid structure for data entries is feasible and appropriate. Because this is a small scaled project, there would be no immediate or forseeable need to scale horizontally. Because of these, by using this method, we are able to reap the additional security benefits. These include authorization and authorization checks as well as backup and recovery.

Option 1: **Microsoft SQL Server**

Pros	Cons
integration with Microsoft ecosystem	costs for lisences can ramp up as application scales
scalable	designed to run on Windows OS
high performance	vendor lock-in
offers developer friendly tools	scalability can be difficult at high volumes
analytics reports	steep learning curve
robust security features	

Option 2: **Amazon RDS**

Pros	Cons
scalable	vendo lock-in
automated backups	limited control and customization to configurations
point-in-time backups	limited performance
robust security features	
high availability	
fault tolerance	
AWS ecosystem	
managed service	

Option 3: **Oracle**

Pros	Cons
scalable	costs for lisences can ramp up as application scales
various addiontal features	steep learning curve
high availability	vendor lock-in
robust security features	resource-intensive in terms of CPU, memory, and storage requirements
disaster recovery support	
high-performance transaction processing	

4.2 Document Store

This solution uses non-relational database, specifically the docuemnt store. This is because with frequent updates to the user's listening activity, the data may change and thus need to be updated. This would result in more writes than reads which would be great because this method would be faster to write than to read. This mehtod provides a dynamic, future-proof solution for scalability. If there is a feature later on that requires the strucutre to be changed, it can be done with minimal effort. In addition, scaling horizontally is feasible.

Option 1: **Firebase**

Pros	Cons
real-time features	cannot perform complex queries
can be integrated with other tools	vendor lock-in
concise documentation	can be very costly as your application scales
quick and easy integration and setup	authentication and authorization may not meet desired requirements
authentication feature	limitations on serverless functions
Google Analytics and Crashlytics for additional insights	performance drops as you scale
horizontal scalability	free plan only supports basic functions
Hosting capabilities	
serverless functions	

Option 2: **mongoDB**

Pros	Cons
flexible data model by using BSON	not ACID compliant
scalable horizontally	limitations on data integrity
powerful querying capabilities	complex queries may hinder performance
horizontal scalability	may lead to duplicate data
high availability and fault tolerance	not able to utilize joins to pull data from different collections efficiently
simple installation	can lead to large consumption of memory
cloud integration	

Option 3: Amazon DynamoDB

Pros	Cons
scalable horizontally	can be costly as your application scales
high performance with high traffic	limited query flexibility
high availability and durability	eventual consistency and not immediate consistency for all replicas
flexible data model by key value store	not ACID compliant
high performance read and write operations	data size limitations at 400KB per item
efficient querying by automatically creating and retaining indexes	vendor lock-in
multi-region replication to reduce latency for users globally	
integrated caching	
integration with AWS ecosystem	

4.3 Time Series

This solution incorporates a time series database with one of the previous methods. A time series database would be used to consistently store up to date information on users' listening activity and data. This could be used for many potential features related to analysis on user's past data such as end of the year review on a user's listening experience similar to Spoify Wrapped. When needed, necessary data can be extracted from this time series database onto another one.

Option 1: InfluxDB

Pros	Cons
flexible data model	high injestion rates will cause high resource consumption
powerful query capbilities	limited use case
scalable	may contain bugs
high availibility	
fault tolerance	

Pros	Cons
ability to integrate other functionalities within ecosystem	
built-in time series functions	

Option 2: OpenTSDB

Pros	Cons
scalable	difficult to get started
high performance for read and write	relies on HBase for data storage and indexing
flexible data model	is a distributed database which is difficult for small scalred applications to manage and maintain
integration with Hadeep ecosystem	limited query capacilities
wide range of additional features	limited help and community online compared to competitors
built-in times series functions	

Option 3: Prometheus

Pros	Cons
powerful metrics collections and querying	not as scalable as other competitors
multi-dimensional data model to allow for quick and efficient query	large datasets while require large resource consumption
alert system	lack of distributed query capabilities
PromQL for complex queries	diffiuclt to achieve high availability
dynamic service discovery for integrations	
various expoerters and integrations available	