**Documentation - Cloud Computing Homework 5 - Music App**

1. **Case study**

Enhance the musical experience at social gatherings with our personalized, AI-driven playlist app that adapts in real-time to the party's mood and audience preferences. Beyond playing music, our app also allows users to share their favorite song parts and lyrics and connect with friends, making it a social media platform centered around music.

**Market Context:** Most music streaming services currently cater to individual experiences, emphasizing personal listening. Our app introduces a transformative concept by incorporating a social dimension into music listening. It features dynamic playlists that not only reflect the collective mood of a gathering but also promote user interaction, setting it apart from the typical static playlist functionalities.

**Differentiation from Existing Solutions:** Existing platforms like Spotify and Apple Music offer extensive catalogs but fall short in providing interactive features suitable for group settings. Our app fills this niche by focusing on the social and musical aspects of gatherings. It supports real-time adaptation of music to match the group's vibe and integrates social sharing features, allowing users to share and discuss their favorite song moments. This approach positions our app not just as a music streaming service but as a vibrant community hub for music enthusiasts.

### **2. Existing Solutions Overview: Spotify, Instagram**

#### **Spotify:**

#### **Architecture**

Spotify utilizes a sophisticated **microservices architecture**, where each microservice is responsible for a specific function and communicates with others via a network. This modular approach enables Spotify to efficiently scale its services to accommodate millions of users simultaneously.

#### **Technologies:**

##### **Backend Technologies**

* **Java**: As the primary programming language, Spotify leverages Java in conjunction with the Spring Framework to build RESTful APIs and manage dependencies.
* **Scala**: Employed for constructing some of the core services, Scala offers robust, statically-typed capabilities on the Java Virtual Machine (JVM).
* **Node.js**: Selected for certain backend services, Node.js facilitates the creation of scalable network applications.
* **Apache Kafka**: This distributed streaming platform supports real-time data processing and event handling, crucial for Spotify's music streaming capabilities.
* **Apache Cassandra**: A NoSQL database, Cassandra provides high availability and scalability, used by Spotify for managing user data like playlists and music libraries.
* **Redis**: This in-memory key-value store is utilized for caching frequently accessed data, helping speed up retrieval of song, album, and artist metadata.
* **Docker**: Essential for containerization, Docker allows Spotify to package services into portable containers that ensure consistency across various deployment environments.

##### **Frontend Technologies**

* **React**: Powers the user interface of Spotify's web application, facilitating dynamic content updates.
* **Redux**: Manages state across the application, providing a predictable container for JavaScript apps.
* **Sass**: A CSS preprocessor that enhances the styling capabilities of the web application.
* **Webpack**: A module bundler that optimizes the assembly of JavaScript files and other assets for deployment.

#### **Infrastructure**

* **Amazon Web Services (AWS)**: Hosts Spotify's services, offering a comprehensive suite of scalable computing resources.
* **Kubernetes**: An open-source system for automating deployment, scaling, and operations of containerized applications, Kubernetes aids Spotify in managing its microservices.
* **Terraform**: Utilized for provisioning and managing infrastructure on AWS via an infrastructure-as-code approach.
* **Prometheus and Grafana**: Together, these tools monitor the health and performance of Spotify’s services and infrastructure, with Grafana providing powerful data visualization capabilities.

**Marketing Approaches:**

Spotify engages in extensive digital marketing through social media, online advertisements, and partnerships with artists and record labels. They also use personalized marketing tactics, leveraging their vast data on user preferences to target ads and playlist recommendations.

#### **Instagram (Music Sharing Features)**

#### **Architecture**

Instagram employs a comprehensive **microservices architecture** similar to Spotify, enabling flexible and scalable service management. This architecture supports its vast global user base and allows for seamless integration of new features such as music sharing within stories and posts.

#### **Technologies:**

##### **Backend Technologies**

* **Python and Django**: Major parts of Instagram's backend are developed using Python and the Django framework, which help manage complex data models and high-volume traffic.
* **GraphQL**: Instagram uses GraphQL to optimize data retrieval, enabling efficient fetching of data such as music tracks associated with stories or posts.
* **Memcached and Redis**: These are used for caching data to improve the speed of data retrieval across Instagram’s distributed systems, crucial for features like displaying music information quickly in the user interface.
* **PostgreSQL**: Employed for database management, handling vast amounts of structured data, including links to music tracks integrated in posts and stories.

##### **Frontend Technologies**

* **React Native**: Allows for the development of Instagram’s mobile applications on both Android and iOS platforms using a common codebase, facilitating faster feature rollout and consistency across devices.
* **JavaScript and HTML/CSS**: For the web version, Instagram uses standard web technologies, ensuring robust and responsive user interfaces.

##### **Infrastructure**

* **Amazon Web Services (AWS)**: Hosts critical infrastructure components, providing the necessary scalability and reliability required to support Instagram's large user base.
* **Docker and Kubernetes**: Utilized for containerization and orchestration, these tools help Instagram manage its microservices efficiently across its massive global operations.
* **Elasticsearch**: Powers the search functionalities, crucial for quickly retrieving musical content tagged in posts and stories.

#### **Marketing Approaches**

* **Influencer Collaborations**: Instagram partners with musical artists and influencers to promote music sharing features, leveraging their vast followings for wider feature adoption.
* **Feature Highlights in App Updates**: New music sharing capabilities are often highlighted in app updates and through in-app notifications to engage users.
* **Interactive Campaigns**: Launches interactive campaigns where users are encouraged to share music or create content around music themes, driving user engagement and content creation.
* **Strategic Partnerships**: Forms partnerships with music streaming platforms to integrate music directly within the Instagram interface, allowing users to share music seamlessly.

**Overview of the technologies we wish to use:**

**Front End:**

The main UI is a React application and we have used it because it is a very popular framework which includes components like a router and http support. Another reason would be the fact that it helps in building reusable components since it has a dependency injection framework.

**BackEnd:**

We have implemented multiple services with azure functions and our backend server is based on multiple azure functions that triggers an https route.

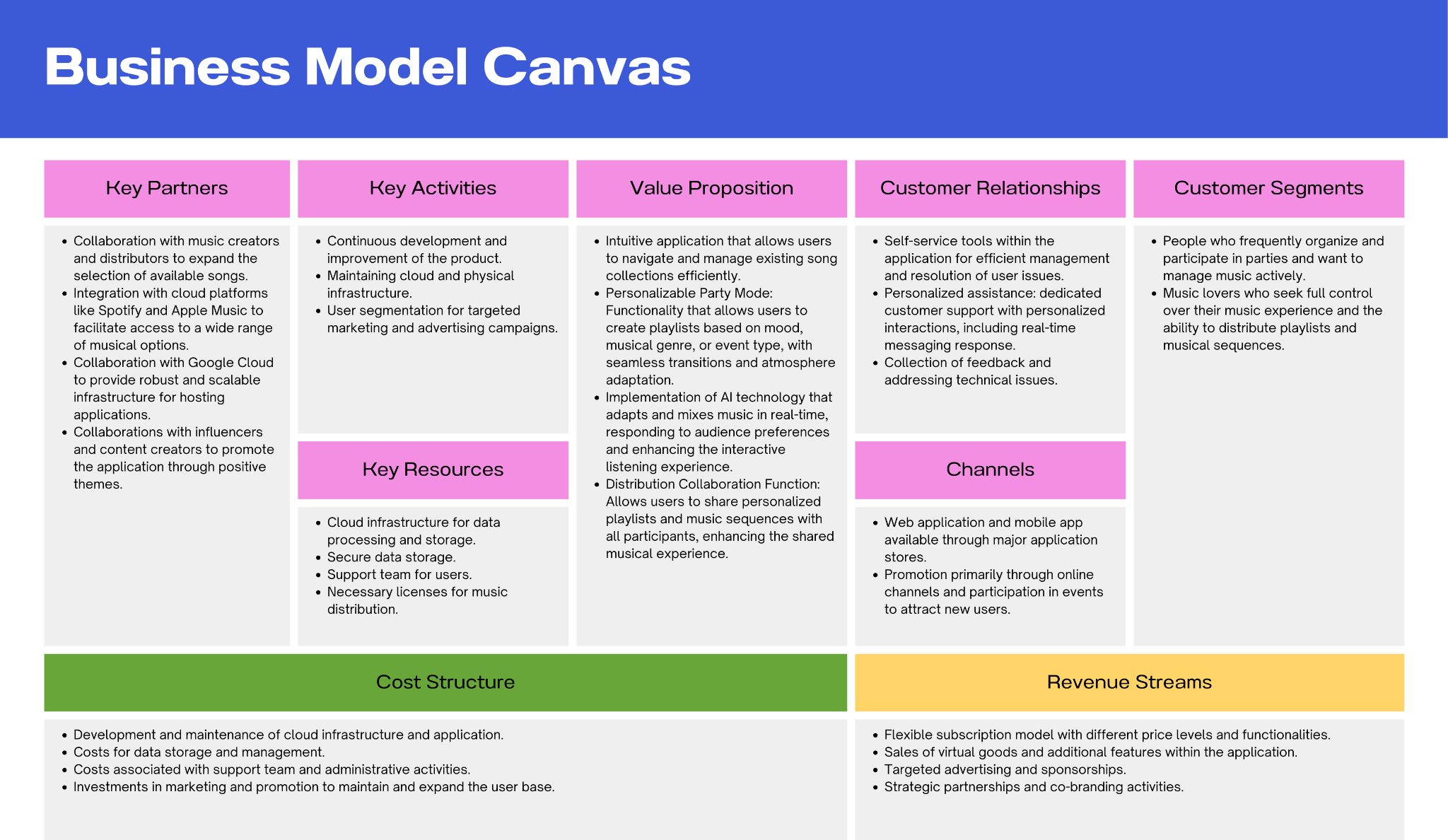
* News Management Service:
  + This service has the role for uploading news only by administrator
  + Another role is to listen and distribute news to all users
* Authentication Service
  + This service has the role of protecting app features.
  + Another role is for storing user personal library or music edits
* Recommendation Service
  + This service has the role of offering recommendation for every user party after some intelligent criteria, it mixes the music too.
* User Music Library Service
  + This service has the role for storing a private music library for every user
* Music Edit Service
  + Here we can edit differents fragments of music and it shows some audio editing options
* Music File Management Service
  + This services for uploading different audio files in an unstructured database. It has the role in managing those file types

### **Azure Services:**

* Azure Cosmos DB: An fast, easy and scalable type of NoSQL database that is used for saving differents collections for our music library
* Azure App Service: We rely on it not only for hosting the users microservice but also for authentication purposes. The platform offers a streamlined approach to building, deploying, and scaling web apps and APIs, aligning perfectly with our authentication needs. Its flexible architecture allows us to seamlessly integrate authentication mechanisms into our application, ensuring a secure and user-friendly experience for our users.
* Azure Functions: A fast and convenient solution tailored for our server architecture on Azure. These functions are designed to be triggered by HTTP requests, perfectly aligning with our serverless setup. With Azure Functions, we can efficiently handle incoming requests and execute specific tasks, offering a seamless and responsive experience for our users.
* Azure Blob Storage: A versatile and reliable solution tailored for storing audio files in our application. With Azure Blob Storage, we have a scalable and cost-effective platform to securely store and manage audio files. Its seamless integration with our Azure infrastructure ensures efficient data management and retrieval, empowering us to deliver a seamless audio experience to our users.

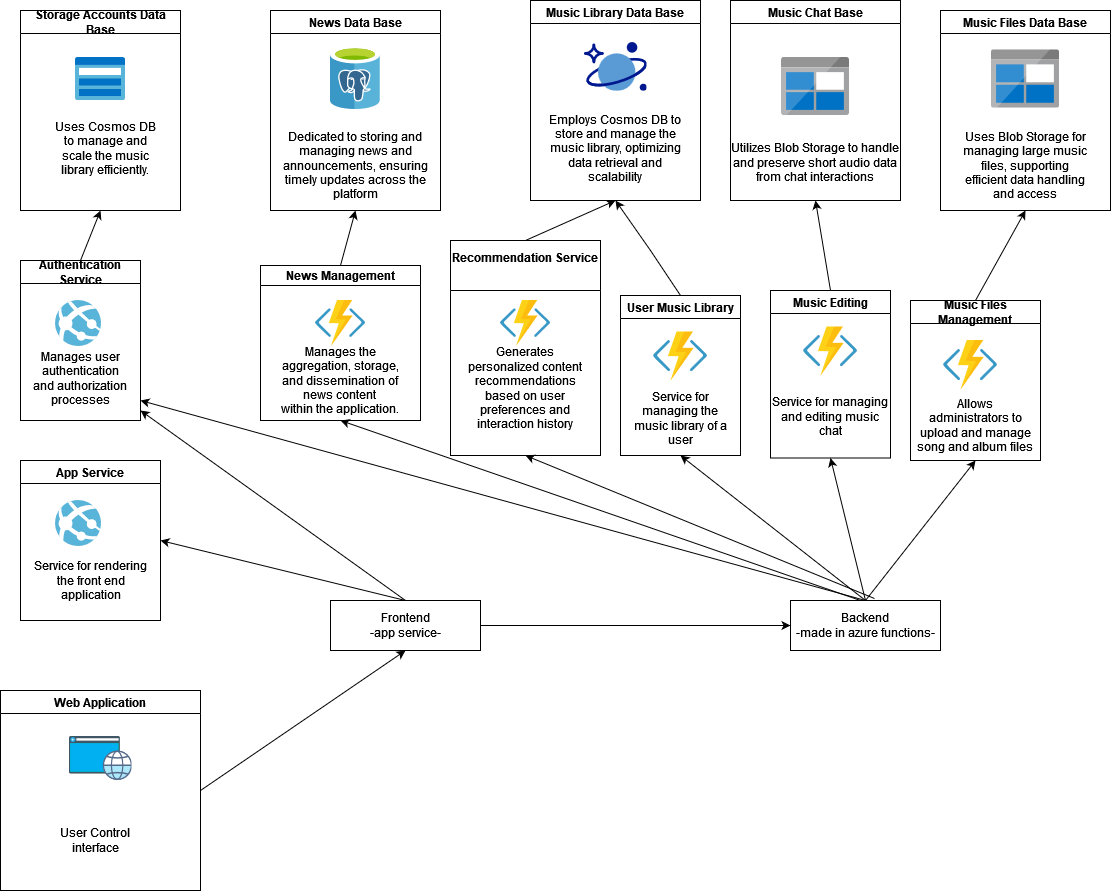
**Business canvas:**

<https://www.canva.com/design/DAGE55Bsq0c/00C3cIlDbsY4WY4rcLG8uw/edit>



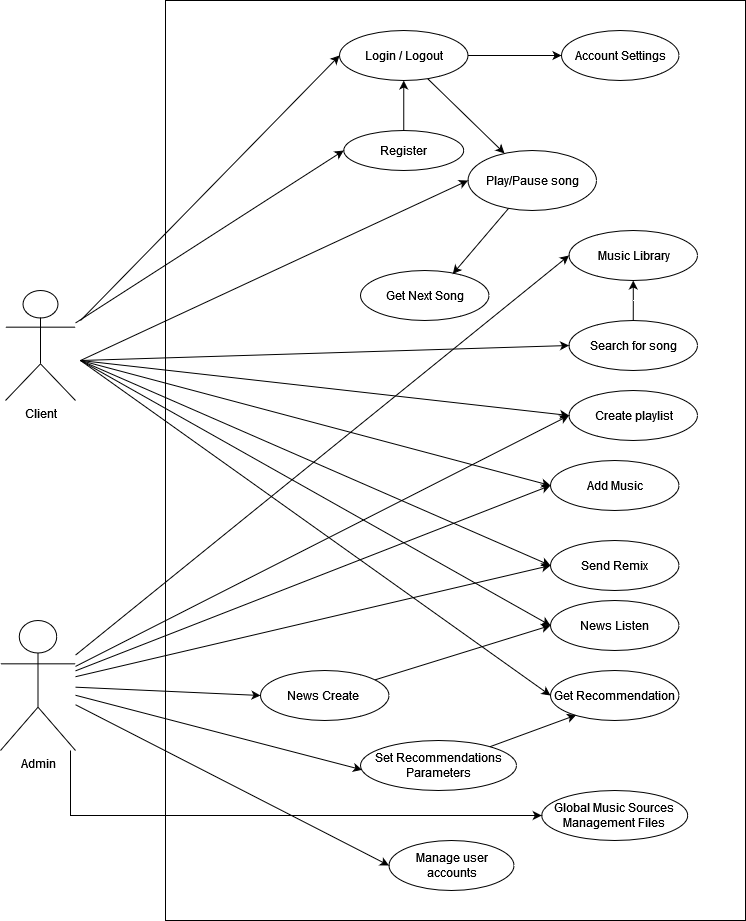
**Architectural diagram:**

<https://drive.google.com/file/d/1iCFGAgZbql5zw-S3yavxGZ3MvXayW3GM/view>



**Use-Case Diagram:**

<https://drive.google.com/file/d/16_ZeapHNnhcAD5HSueISClONu_ux9lI8/view>



**APIs:**

<https://app.swaggerhub.com/apis/Carmina-VivianaCantea/MusicAPI/1.0.0>