**SP-27- Blue Spotify**

**Software Design Document**

**CS 4850 – Section 02–Senior Project**

**Term: Fall 2024**

**Sunday September 01, 2024**

|  |  |
| --- | --- |
| Courtney Faulkner  Team Leader and Tester | Naomie Dongmo  Developer |
| Kapawa Nounamo  Developer | Serena Noboudem  Developer |
| Inserting image...  Leslie Keyangue  Documentation |  |

**Team Members:**

|  |  |  |
| --- | --- | --- |
| Name | Role | Cell Phone / Alt Email |
| Courtney Faulkner | Team Leader and Tester | 706-248-9234  [courtney.faulkner11@gmail.com](mailto:courtney.faulkner11@gmail.com) |
| Naomie Dongmo | Developer | 404-706-1561 |
| Kapawa Nounamo | Developer | 404-202-7469  [helenanaomie3@gmail.com](mailto:helenanaomie3@gmail.com) |
| Serena Noboudem | Developer | 404-599-9575  [nobserena@gmail.com](mailto:nobserena@gmail.com) |
| Leslie Keyangue | Documentation | 404-606-4353 [keyangueleslie@gmail.com](mailto:keyangueleslie@gmail.com) |
| Sharon Perry | Project Owner or Advisor | 770.329.3895  [Sperry46@kennesaw.edu](mailto:Sperry46@kennesaw.edu) |

**Table of Content**

[1. Introduction and Overview 2](#_Toc1849951239)

[1.1. Document Purpose 2](#_Toc893928913)

[2. Design Considerations 2](#_Toc1878373605)

[2.1. Assumptions and Dependencies 2](#_Toc1155496395)

[2.2. General Constraints 3](#_Toc452683143)

[2.3. Development Methods 4](#_Toc961001253)

[3. Architectural Strategies 5](#_Toc511691049)

[4. System Architecture 5](#_Toc1833109852)

[4.1. Client-Side Architecture 5](#_Toc582961826)

[4.2. Backend Architecture 6](#_Toc37383452)

[4.3. Database Layer: 6](#_Toc1789222786)

[5. Detailed System Design 6](#_Toc1068215338)

[ Presentation Layer (Frontend): 6](#_Toc538690404)

[ Application Layer (Backend): 7](#_Toc1490734203)

[ Data Layer: 7](#_Toc839031278)

[6. Glossary 10](#_Toc1758893849)

[7. Bibliography 12](#_Toc1343480210)

# Introduction and Overview

## Document Purpose

The Software Design Document (SDD) provides a detailed blueprint of the system's architecture, design, and implementation. Its primary purpose is to facilitate clear communication among all stakeholders, including developers, project managers, and clients, by offering a comprehensive overview of how the system will be built and function. The SDD ensures that the design aligns with the specified requirements and guides developers in the implementation process. Additionally, it plays a key role in validating the design against the intended functionality, performance, and security standards, helping to identify potential risks and plan for their mitigation.

# Design Considerations

## Assumptions and Dependencies

**Assumptions:**

* **User Access to the Internet:** It is assumed that users of the Spotify mobile app will have reliable access to the internet, either through Wi-Fi or mobile data, as streaming music requires a stable internet connection.
* **Device Compatibility:** It is assumed that the app will be compatible with a wide range of mobile devices, including those running on iOS and Android operating systems, and that users will keep their devices updated with the latest OS versions to ensure optimal performance.
* **User Familiarity with Streaming Services:** It is assumed that users are familiar with basic mobile app navigation and streaming services, reducing the need for extensive user onboarding or tutorials within the app.
* **API Availability:** It is assumed that Spotify’s core APIs, such as those for authentication, music streaming, and user data, will be available and functional throughout the project’s lifecycle, allowing seamless integration into the mobile app.
* **Stable Cloud Infrastructure:** The project assumes that the cloud infrastructure supporting Spotify's services, including databases and storage, will remain stable and scalable to handle user demand and data.

**Dependencies:**

* **Third-Party Services:** The app's functionality may depend on third-party services, such as payment gateways for premium subscriptions, and these services must remain operational and accessible for the app to function as intended.
* **Licensing Agreements:** The availability of music content on Spotify is dependent on licensing agreements with music labels and artists. The app’s catalog and features are dependent on these agreements being in place and continuously renewed.
* **App Store Policies:** The release and updates of the Spotify mobile app depend on compliance with Apple App Store and Google Play Store policies. Changes in these policies could impact the app’s distribution and update processes.
* **Security Protocols:** The app's security measures, including user data encryption and secure communication protocols, depend on the continuous support and updates of industry-standard security libraries and tools.
* **Legal and Regulatory Compliance:** The app must comply with regional and international laws, including data privacy regulations like GDPR. Changes in these laws could impact the app's features and how user data is managed.

## General Constraints

1. **Data Privacy and Security**: Compliance with global data protection regulations is mandatory. This requires the encryption of user data, secure handling of user preferences, and the implementation of robust authentication mechanisms using Firebase.
2. **Resource Availability:** The app is constrained by the availability and performance of third-party services like Google Ad Mob for ads and Firebase for authentication.

Any downtime or API changes from these services could directly affect app functionality.

1. **Performance Requirements:** The app must maintain high performance standards, ensuring minimal latency during music playback, quick response times for user interactions, and smooth navigation across the interface. This is especially critical in regions with variable network speeds.
2. **Interoperability:** The app must seamlessly integrate with various external APIs for features like lyrics synchronization and song recommendations. These APIs must be interoperable across different platforms and devices, ensuring consistent functionality regardless of the operating system.
3. **User Interface Consistency:** The UI must be consistent across all supported devices, including various Android and iOS versions. The app should maintain a uniform look and feel while adapting to different screen sizes and resolutions.
4. Security Compliance: The app must implement secure coding practices, including input validation, secure session management, and protection against common vulnerabilities such as SQL injection, cross-site scripting (XSS),, and cross-site request forgery. (CSRF). Firebase's security rules must be strictly enforced to protect user data.
5. **Memory and Storage Constraints:** The app must efficiently manage memory and storage, especially on lower-end devices with limited resources. Caching strategies for lyrics and music streaming data must balance performance with memory usage to avoid app crashes or slowdowns.
6. **Ad Integration:** The integration of Google Ad Mob for ad delivery must be done in a way that does not significantly disrupt the user experience. Ads should be strategically placed to minimize intrusiveness while maximizing revenue. Compliance with Google’s ad policies is mandatory.

## Development Methods

Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.

**1. Agile Development:**

The project will adopt an Agile development methodology, allowing for iterative and incremental progress. Agile's flexibility enables the team to adapt to changes in requirements, user feedback, and market conditions. Development will be organized into sprints, typically lasting 2-4 weeks, with each sprint focusing on delivering a specific set of features or improvements.

**2. Scrum Framework:**

Within the Agile methodology, the Scrum framework will be employed to manage the project. The Scrum team will consist of a Product Owner, Scrum Master, and Development Team. Regular ceremonies, including Sprint Planning, Daily Standups, Sprint Reviews, and Retrospectives, will be held to ensure continuous progress and address any blockers. The Product Backlog will be maintained and prioritized to reflect the most critical features and improvements.

**3. Test-Driven Development (TDD):**

To ensure code quality and reduce the number of bugs in the production environment, Test-Driven Development (TDD) will be implemented. In TDD, automated unit tests are written before the actual code, ensuring that each new feature or function is thoroughly tested as it is developed. This approach helps in maintaining high code quality and facilitates easier refactoring.

**4. User-Centered Design (UCD):**

A User-Centered Design approach will be employed to ensure that the app meets the needs and expectations of its users. This involves conducting user research, creating personas, and engaging in usability testing throughout the design and development phases. The goal is to create an intuitive and accessible user interface that provides a seamless experience across different devices.

**5. Code Review and Pair Programming:**

Code reviews will be a mandatory part of the development process, ensuring that all code is checked for quality, security, and adherence to coding standards before being merged into the main branch. Pair programming will be encouraged, where two developers work together at one workstation. This collaborative method improves code quality, facilitates knowledge sharing, and helps in solving complex problems more effectively.

**6. Documentation and Knowledge Sharing:**

Comprehensive documentation will be maintained throughout the development process, including technical documentation, API references, and user guides. Knowledge-sharing sessions and regular team meetings will be conducted to ensure that all team members are aligned and have access to the necessary information and resources

# Architectural Strategies

The app will employ a microservices architecture, with specific microservices dedicated to handling monetization-related functionalities. These services could include subscription management, in-app purchases, advertising, and payment processing. By isolating these services, the app can scale each component independently, ensuring that high-traffic monetization services do not impact the performance of the core music streaming functions.

A data-driven approach will be used to enhance user engagement and monetization. This includes tailoring playlists, recommendations, and promotional offers. For users on the free tier, the app will integrate a scalable advertising platform that can deliver targeted ads. An ad-serving microservice will handle the delivery, and tracking of ads, ensuring minimal disruption to the user experience.

A modular billing and subscription management system will be implemented to handle different subscription tiers, free trials, discounts, and renewals

# System Architecture

## ****Client-Side Architecture****

 **User Interface (UI):** The UI layer is responsible for rendering the app's visual elements, such as menus, music controls, playlists, and advertisements. It is designed using responsive and adaptive UI principles to ensure a consistent experience across various devices and screen sizes.

 Playback **Engine:** This component handles the streaming and playback of audio content. It communicates with the backend to fetch streaming URLs.

 Caching **and Storage:** Local caching is used to store frequently accessed data (e.g., album art, playlists). This reduces network usage and improves app performance, especially in low-connectivity scenarios.

 API **Client:** The API client manages communication with backend services via RESTful APIs. It handles user authentication, fetching music content, updating user profiles, and other interactions with the server.

 Monetization **Components:** These components manage in-app purchases, subscription status checks, and ad delivery.

## ****Backend Architecture****

 UserService**:** Manages user accounts, authentication, preferences, and subscriptions. It interfaces with OAuth providers for secure login and handles user data storage.

 ContentService**:** Handles the catalog of music, playlists, podcasts, and other media. It manages metadata, licensing restrictions, and the delivery of streaming URLs.

 AdService**:** Manages the scheduling and delivery of ads for free-tier users. It interacts with external ad networks and tracks ad performance.

 BillingService**:** Handles payments, subscription management, renewals. It integrates with payment gateways and maintains records of user transactions.

## Database Layer:

 UserDatabase**:** Stores user data, including profiles, playlists, subscription status, and listening history.

 ContentDatabase**:** Contains meta data for all available music, podcasts, and other media content. This data is indexed and optimized for fast retrieval.

 BillingandTransactionsDatabase**:** Stores payment information, transaction history, and subscription details.

# Detailed System Design

The system design for the music streaming app includes a comprehensive breakdown of the components, their interactions, constraints, resources, and interfaces. This section will describe how the system is organized, how data is processed, and the specific roles and responsibilities of each component.

1. **Classification**

The system is divided into three primary layers:

# Presentation Layer (Frontend):

* + **User Interface (UI):** Built with React Native for cross-platform compatibility (iOS and Android).
  + **Responsibilities:** Manages user interactions, displays content, and communicates with the backend services.

# Application Layer (Backend):

* + **Backend Framework:** Node.js with Express.js serves as the core backend framework.
  + **Responsibilities:** Processes business logic, handles API requests, manages sessions, and interfaces with the database and third-party services like Firebase.

# Data Layer:

* **Database:** MongoDB is used for storing user data, playlists, song metadata, and user preferences.
* **Caching:** Redis is employed for caching frequently accessed data to optimize performance.

1. **Definition**

**2.1. Presentation Layer (Frontend)**

* **Components:**
* **Home Screen:** Displays featured playlists, recommended songs, and quick access to different sections.
  + **Search and Discovery Component:** Allows users to search for songs, artists, and albums, providing real-time search suggestions.
  + **Player Component:** Manages song playback, displays lyrics, and allows users to like songs.
  + **Playlist Management:** Enables users to create, view, and manage their playlists (available only for premium users).
  + **Settings and Profile Management:** Lets users update personal information, subscription details, and app preferences.
* **Data Flow:**
  + The UI components communicate with the backend via RESTful API calls to fetch or update data.
  + State management is handled for efficient state handling and data flow.

**2.2. Application Layer (Backend)**

* Components:
  + User Management Service: Manages user authentication, authorization, and profile data using JWT tokens.
  + Content Management Service: Handles CRUD operations for song data, playlists, and user preferences.
  + Recommendation Engine: Utilizes user data and preferences to recommend songs, albums, and playlists.
  + Ad Service: Integrates Google Ad Mob to display targeted ads to free-tier users.
  + Notification Service: Uses Firebase Cloud Messaging (FCM) to send notifications about new releases, playlist updates, and recommendations.
* **Data Flow:**
  + Receives and processes HTTP requests from the frontend, interacts with the MongoDB database, and communicates with third-party APIs like Google Ad Mob.
  + Utilizes Redis for caching frequently accessed data to reduce latency and improve user experience.

**2.3. Data Layer**

* **Components:**
* **MongoDB Database:** Stores all user-related data, including authentication details, playlists, song metadata, and user interactions.
* **Redis Cache:** Caches frequently accessed data, such as popular playlists or user preferences, to enhance performance.
* **Data Flow:**
  + Handles read and write operations from the backend services.
  + Ensures data consistency and integrity with proper indexing and schema design.

1. **Constraints**

* **Performance Constraints:**
  + API response times should be optimized to under 200 milliseconds to ensure smooth user experience.
  + The system should support up to 10,000 concurrent users without significant performance degradation.
* **Security Constraints:**

All data must be transmitted over HTTPS to ensure secure communication.

* + User data stored in MongoDB must be encrypted at rest using AES-256 encryption.
  + JWT tokens should be short-lived and require periodic refreshments to maintain security.
* **Scalability Constraints:**
  + The system must be capable of horizontal scaling to handle an increasing number of users and data volume.
* **Compliance Constraints:**
  + - * + The app must comply with relevant data protection regulations such as GDPR and CCPA.
        + User consent is required for collecting and processing personal and sensitive data.

1. **Resources**

* **Hardware:**
  + Cloud servers, such as AWS EC2 instances, for hosting backend services.
  + Managed MongoDB clusters, like MongoDB Atlas, for database services.
  + Redis instances for caching services.
* **Software:**
  + **Frontend:** React Native, Axios for HTTP requests, Firebase SDK, Riverpod/Bloc for state management,
  + **Backend:** Node.js, Express.js, Mongoose (for MongoDB interactions), Redis, JWT.
  + **Database:** MongoDB for data storage, Redis for caching.
  + **CI/CD Tools:** GitHub Actions for continuous integration and deployment.
* **Third-Party Services:**
  + **Google Ad Mob:** For displaying ads to free-tier users.
  + **Firebase Cloud Messaging:** For sending notifications to users.
  + **Cloud Storage (e.g., AWS S3 or Google Cloud Storage):** For storing user-uploaded content.

1. **Interface/Exports**

**5.1. Frontend Interfaces**

* **User Actions:**
  + **Login/Signup:** User authentication via a login/signup form, utilizing JWT-based authentication.
  + **Song Playback:** Interface to play, pause, and seek within songs, displaying lyrics and allowing users to like songs.
  + **Playlist Management:** Interface for premium users to create, edit, and delete playlists.
  + **Search:** Interface for searching songs, artists, and albums, displaying results in real-time.
* **API Endpoints (Interacts with Backend):**
  + Fetch user profile data.
  + Update user profile settings.
  + Retrieve user playlists.
  + Create a new playlist.
  + Fetch personalized song recommendations.

**5.2. Backend Interfaces**

* **Internal Services Communication:**
  + Microservices within the backend communicate using RESTful APIs and message queues for decoupled services.
* **Database Interaction:**
  + Uses Mongoose for MongoDB interactions, with well-defined schemas for structured data access and management.
* **External APIs:**
  + **Google Ad Mob:** Integrates for ad management and display.
  + **Firebase Cloud Messaging:** Manages push notifications.
* **Exports:**
  + Provides RESTful API endpoints for front-end consumption.
  + Sends data and notifications to the frontend and third-party services.

**5.3. Data Exports**

* **To Frontend:**
  + JSON formatted responses for all RESTful API requests.
  + Real-time data updates via WebSocket’s or Firebase for specific features like notifications and live updates.
* **To Third-Party Services:**
  + Sends user interaction data to Google Ad Mob for ad targeting.
  + Pushes notification data to Firebase for user alerts.

# Glossary

* + **API (Application Programming Interface):** A set of protocols and tools that allow different software applications to communicate with each other. In the context of the Spotify app, APIs enable the mobile app to interact with backend services for tasks such as user authentication, music streaming, and data retrieval.
  + **Authentication:** The process of verifying the identity of a user or system. In the Spotify app, authentication ensures that only authorized users can access their accounts and use the app’s features.
  + **Authorization:** The process of determining what an authenticated user is allowed to do within the app. For example, a user with a premium subscription might have access to features that are restricted for free-tier users.
  + **Backend:** The server-side component of the Spotify app that handles data storage, processing, and communication with the mobile app. It includes microservices, databases, and APIs that work together to deliver content and services to the app.
  + **Caching:** The practice of storing data temporarily in a location that allows for faster access. In the Spotify app, caching is used to store frequently accessed data, such as user playlists and album artwork, to reduce load times and improve performance.
  + **Client-Side:** Refers to the components of the Spotify app that run on the user's device, such as the user interface, playback engine, and local storage. The client-side interacts with the backend to retrieve and display data.
  + **Encryption:** The process of encoding data to protect it from unauthorized access. Encryption is used in the Spotify app to secure sensitive information such as user credentials and payment details.
  + **IAP (In-App Purchase):** A feature that allows users to purchase additional content or services within the app. In Spotify, IAP is used for purchasing premium subscriptions directly through the app.
  + **JWT (JSON Web Token):** A compact, URL-safe token used to securely transmit information between parties. In the Spotify app, JWTs are used for stateless user sessions, ensuring that users remain authenticated as they interact with the app.
  + **Microservices:** A software architecture pattern where a large application is divided into small, independent services that communicate with each other. In the Spotify backend, microservices handle different functionalities such as user management, music streaming, and billing.
  + **Monetization:** The process of generating revenue from the app. In Spotify, monetization strategies include offering premium subscriptions and displaying ads to free-tier users.
  + **NoSQL:** A type of database that provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Spotify uses NoSQL databases for storing user data, playlists, and metadata.
  + **OAuth2:** An open standard for access delegation commonly used to grant websites or applications limited access to a user's data without exposing their credentials. Spotify uses OAuth2 for secure user authentication.
  + **Playback Engine:** The component of the Spotify app responsible for streaming and playing audio content. It manages buffering, streaming quality, and offline playback.
  + **REST (Representational State Transfer):** A style of architecture for designing networked applications. It relies on a stateless, client-server communication protocol, typically HTTP. Spotify uses REST APIs for communication between the mobile app and backend services.
  + **Subscription Management:** The process of handling user subscriptions, including purchasing, renewing, and canceling subscriptions. In Spotify, this involves managing access to premium features based on the user's subscription status.
  + **UI (User Interface):** The visual part of the Spotify app that users interact with. It includes buttons, menus, and other graphical elements that allow users to navigate and control the app.
  + **WebSocket:** A communication protocol that provides full-duplex communication channels over a single TCP connection. Web Sockets are used in the Spotify app for real-time communication, such as live notifications and updates.

# Bibliography

* **React Native Documentation**

Facebook, Inc. "React Native – A Framework for Building Native Apps Using React." React Native Docs. Accessed August 30, 2024.

This resource offers extensive documentation on React Native, essential for developing cross-platform mobile applications, with guidance on UI components, state management, and performance optimization.

* **Node.js and Express.js Documentation**

OpenJS Foundation. "Node.js Documentation." Node.js Docs. Accessed August 30, 2024.

Express.js. "Express - Fast, Unopinionated, Minimalist Web Framework for Node.js." Express.js Docs. Accessed August 30, 2024.

These documents provide essential guidelines for building backend services with Node.js and Express.js, focusing on RESTful API development, middleware integration, and server-side logic handling.

* **MongoDB Documentation**

MongoDB, Inc. "MongoDB Manual." MongoDB Docs. Accessed August 30, 2024.This manual covers MongoDB’s use cases, data modeling techniques, CRUD operations, and strategies for optimizing performance, which are vital for the app's data storage and management.

* **Spotify API Documentation**

Spotify. "Spotify Web API Documentation." Spotify Developer Docs. Accessed August 30, 2024.

Comprehensive information on integrating Spotify’s API to access music, playlists, and user data, including endpoints, authentication methods, and rate limits, essential for building core features of the app.

* **Firebase Authentication Documentation**

Google. "Firebase Authentication." Firebase Docs. Accessed August 30, 2024.

Instructions on using Firebase Authentication for managing user accounts and securing access to the app, including setup, integration with Dart, and handling user sessions.

* **Google AdMob Documentation**

Google. "Google AdMob." Google AdMob Docs. Accessed August 30, 2024.

Documentation on integrating Google AdMob to monetize the music streaming app, covering setup, ad placement strategies, and revenue optimization techniques.

* **JWT (JSON Web Tokens) Specification**

Jones, M., Bradley, J., & Sakimura, N. "JSON Web Token (JWT)." RFC 7519. Accessed August 30, 2024.

The official specification for JSON Web Tokens, providing details on their structure, secure data transmission usage, and implementation for user authentication and authorization in the app.

* **Redis Documentation**

Redis Labs. "Redis Documentation." Redis Docs. Accessed August 30, 2024.

Documentation on using Redis for caching frequently accessed data to enhance application performance and reduce load on MongoDB, critical for maintaining fast response times in the app.