¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ 47f3ae8b25d81b37fab0bd218f9d5318a3bb04da

CDN Music App

Project 2
9/29/2025

Name	**Email Address**
Aaron Downing	aaron.downing652@topper.wku.edu
Ryerson Brower	ryerson.brower178@topper.wku.edu

CS 396
Fall 2025
Project Technical Documentation

Contents

List of Figures

1 Introduction

1.1 Project Overview

The Content Delivery Network (CDN) Implementation project is designed to enhance the performance, scalability, and reliability of a music streaming and download service by caching audio content at multiple geographically distributed edge locations. By leveraging edge caching, the system reduces latency, improves download speeds, and provides a seamless user experience for streaming music, even during peak usage periods. The primary objective is to ensure that users can access audio files quickly and efficiently, regardless of their location or device, while the backend intelligently manages routing, caching, and content delivery.

The system architecture was developed using Node.js for the backend, which provides efficient server-side processing, request handling, and real-time updates to cached content. The frontend was built using HTML, CSS, and JavaScript, creating a responsive and intuitive user interface where users can easily find, stream, and download music. Together, this technology stack ensures a fast, scalable, and user-friendly experience.

Functionally, the CDN caches music files in multiple formats (MP3, AAC, OGG) at edge servers to maximize compatibility with diverse devices and playback applications. Requests are routed intelligently based on a user's geographical location and network conditions, ensuring minimal streaming latency. Real-time updates allow new releases and content changes to propagate quickly, guaranteeing that users always have access to the latest music. The system also incorporates analytics to track streaming quality, download speeds, and user interactions, providing administrators with actionable insights.

Non-functional requirements focus on ensuring response times under 100 milliseconds, scalability through the addition of edge servers, and industry-standard security protocols such as encryption and authentication. The system also emphasizes seamless usability with clear feedback and error handling. With built-in monitoring and analytics, the CDN can be continuously optimized for performance and efficiency, making it a robust solution for large-scale music distribution.

1.2 Project Scope

Text goes here.

1.3 Technical Requirements

1.3.1 Functional Requirements

Mandatory Functional Requirements
1. The CDN must cache audio files at edge locations to minimize latency for users accessing the music streaming or download service. Cached content should include various formats and qualities to accommodate different user preferences.
2. The system must intelligently route requests to the nearest edge location based on user geographical location and network conditions to ensure optimal performance during music streaming or downloads.
3. The CDN must support real-time content updates, allowing new music releases or changes to existing files to be propagated quickly across edge locations, ensuring that users always access the latest content.
4. The CDN must be capable of serving audio files in multiple formats (e.g., MP3, AAC, OGG) to accommodate various devices and playback requirements, ensuring compatibility with a wide range of client applications.
5. The system must provide analytics features that track user interactions, streaming quality, and download speeds, allowing administrators to monitor performance and make data-driven improvements.
Extended Functional Requirements
1. The CDN must provide users with the ability to adjust playback options, such as play, pause, and skip, during streaming.
2. The system must provide users with the option to download songs for offline playback in supported formats.
3. The CDN must allow users to view recently played or downloaded songs for quick access.

The functional requirements of the CDN music application define the core capabilities the system should provide to deliver music efficiently and effectively to users. The CDN is designed to cache audio files at multiple edge servers, reducing latency and improving performance for both streaming and downloads. Requests are routed to the nearest edge location, ensuring that users receive the fastest possible access based on their geographic location and network conditions. The system also supports real-time updates, so new released and modified files are released across the network. Guaranteeing users always have access to the most up-to-date content. Additionally, the CDN serves audio in multiple formats such as MP3, AAC, and OGG to support a wide range of devices and applications. Users can also download songs for offline playback, access recently played or downloaded content, and control playback with basic options such as play, pause, and skip during streaming.

1.3.2 Non-Functional Requirements

Mandatory Non-Functional Requirements
1. The CDN must ensure that the average response time for streaming and download requests is under 100 milliseconds, even during peak usage periods, to provide a seamless user experience.
2. The system must be designed to handle an increasing number of simultaneous users and requests without performance degradation, allowing for the addition of new edge servers as user demand grows.
3. The system must implement industry-standard security protocols to protect user data and audio content, including encryption of data in transit and robust authentication mechanisms to prevent unauthorized access.
4. The CDN must provide a seamless and intuitive user experience, ensuring that users can easily find, stream, and download music with minimal steps and without confusion, supported by clear feedback and error messaging throughout the process.
5. The system must provide real-time monitoring and analytics capabilities, allowing administrators to track performance metrics, user interactions, and content delivery efficiency, facilitating ongoing optimization and troubleshooting.
Extended Non-Functional Requirements
1. The system must maintain an uptime of at least 99 percent to ensure high availability of the music streaming and download service.
2. The system must maintain error rates (e.g., failed requests) below 1 percent to ensure a consistent and reliable user experience.
3. The CDN must be designed with a responsive user interface so that the service works smoothly on desktops

The non-functional requirements of the CDN project establish the quality standards and operational goals that ensure the system's efficiency, scalability, and reliability. The CDN must provide a fast and seamless experience, maintaining response times under 100 milliseconds, even during peak traffic. Availability is a priority, with the system expected to maintain 99.9 percent uptime, while reliability is ensured by keeping request error rates below 1 percent. Scalability is supported by a design that allows the easy addition of new edge servers as user demand grows. The system also emphasizes security, implementing encryption for data in transit and robust authentication to protect both users and audio content. From a usability perspective, the user interface is designed to be responsive across desktops, tablets, and mobile devices, ensuring accessibility for all. Finally, administrators benefit from real-time monitoring and analytics to track performance, optimize delivery, and troubleshoot issues efficiently.

2 DevOps - Continuous Integration and Continuous Delivery Approach and Results

The approach we took for our CI and CD we setting up GIT Actions. This really helped us be able to test our files the moment they are updated and see if there is any errors. We leverage GitHub Actions to implement our CI workflow, which automatically triggers a series of vital steps upon every code push. This includes running a fast build process and executing automated unit tests and integration tests. Automating these checks allows us to catch errors and conflicts early, significantly reducing the cost and complexity of fixing issues later in the

development cycle. The immediate feedback loop established by Git Actions ensures code quality remains high, making the entire release process inherently safer. Our CD practice focuses on creating a reliable, repeatable path to production. The same automated pipeline used for testing will also handle the process of packaging the application and preparing it for release to staging or production environments. This standardization removes human error and guarantees consistency across deployments, making releases safer and more predictable. This highly reliable, automated process is the primary driver for achieving a faster time to market for new music content and CDN features.

3 DevOps - Architecture Approach, Models, and Results

Our CDN project is designed with a loosely coupled architectural approach, where distinct functional components are logically separated. This modular design is the backbone of our ability to work independently, deploy new features frequently, and ensure efficient system scaling while maintaining high reliability. This strategy ultimately enables the rapid delivery of software value. The Content Origin: The source of truth for audio files (the music directory). Its sole purpose is content storage and serving, decoupled from the delivery logic. The Delivery and Routing Layer: This component, simulated by our Edge Cache and Geo-Routing logic in server.js, handles all user interactions. It independently manages caching decisions, routing, and time-to-live (TTL) updates. The Analytic and Monitoring Service: This block autonomously tracks performance metrics (latency, cache hits) via the /analytic endpoint, ensuring that data gathering is separate from content delivery. The independence of these components drives our operational efficiency. Because the routing logic is decoupled from the content storage, we can update the caching algorithm or geo-routing function without requiring a re-deployment or extensive re-testing of the entire Origin content library. This isolation enables frequent and safer deployments.

4 DevOps - Product and Process Approach and Results

Our approach to implementing the Content Delivery Network (CDN) for the music streaming service emphasizes customer-centric, feedback-driven development. At every stage, we prioritize end-user experience, ensuring low-latency streaming, fast downloads, and seamless access to audio content in multiple formats.

Feedback loops are embedded into the development cycle through performance analytics, user behavior metrics, and direct feedback, allowing iterative refinement of caching strategies, routing logic, and content delivery rules. To support continuous improvement and reliable deployments, we leverage GitHub Actions for automated testing, integration, and deployment pipelines. This enables rapid iteration while maintaining quality and stability. By combining customer insights, cross-functional collaboration, and DevOps automation, our CDN implementation continuously evolves to deliver superior user experiences and scalable, efficient content delivery.

5 DevOps - Product Management and Monitoring Approach and Results

For our music streaming CDN, we set up monitoring and feedback systems that give real-time information about system performance. We track metrics like streaming latency, cache hits, download speeds, and errors to see how the system is working at all times.

Dashboards and alerts help the team spot problems early, so we can fix them before users are affected. We also log user interactions and system behavior to guide improvements in caching, routing, and content delivery.

This approach helps keep the system stable, makes decision-making easier, and allows the team to continuously improve performance. By using real-time data and feedback, we ensure a better experience for users and more efficient operations for the team.

6 DevOps - Cultural Approach and Results

Our team focused on creating a collaborative environment built on trust and shared responsibility. Everyone had a voice in decisions, and we encouraged open communication to make sure ideas and concerns were heard.

We emphasized continuous learning by sharing knowledge, reviewing code together, and experimenting with new approaches to improve our CDN and music streaming service. Mistakes were treated as learning opportunities, not failures, which allowed the team to try innovative solutions without fear.

This culture of trust, collaboration, and experimentation helped the team perform at a high level. By supporting each other, sharing responsibility, and continuously learning, we were able to develop better solutions, improve system performance, and deliver a stronger user experience.

7 Software Testing and Results

7.1 Software Testing Plan Template

Test Plan Identifier:
Introduction:
Test item:
Features to test/not to test:
Approach:
Test deliverables:
Item pass/fail criteria:
Environmental needs:
Responsibilities:
Staffing and training needs:
Schedule:
Risks and Mitigation:
Approvals:

8 Conclusion

In conclusion this project went a bit smoother than the first project we did. This may be because we jumped on it a lot faster than the other. We were able to get the hardest part which was the coding done pretty fast. One thing that we could have improved on was the way we implemented the Music App. We not to sure how to make a complete network for the music app so we only used a local host. Similarly for the edge systems we only simulated it with our code that gave us the response time and the predicted edge system, such as New York, Asia, and more. We downloaded two songs, a longer song (around 7 mins) and a shorter song (around 3 mins). Some thing we didn't know how to implement was the ability to download and upload song remotely. So there was a lot we wished we could have polished out but we had to keep our scope pretty small to complete this project. But overall thanks to this project brought a lot of light to what DevOps was and how to use it to our advantage. We will be able to use these skills in the future.

9 Appendix

9.1 Software Product Build Instructions

Text goes here.

9.2 Software Product User Guide

Text goes here.

9.3 Source Code with Comments

Text goes here.