

# Cloud-Based Music Streaming Service with AWS Amplify

**Sujith Bojjawar (2210030340)**

*Under the esteemed guidance of*

**Ms. P. Sree Lakshmi**

Assistant Professor,

Department of Computer Science and Engineering



# Project Overview

This project addresses the challenge of building scalable full-stack apps without deep backend expertise. Using AWS Amplify, I developed a cloud-powered web application with authentication, API integration, storage, and hosting — all with minimal configuration.

## **Objective:**

To create and deploy a secure, full-stack web app using AWS Amplify, focusing on speed, simplicity, and real-world functionality.

## **Real-World Relevance:**

This solution reflects modern industry practices, making it highly applicable for startups and scalable cloud-based projects.



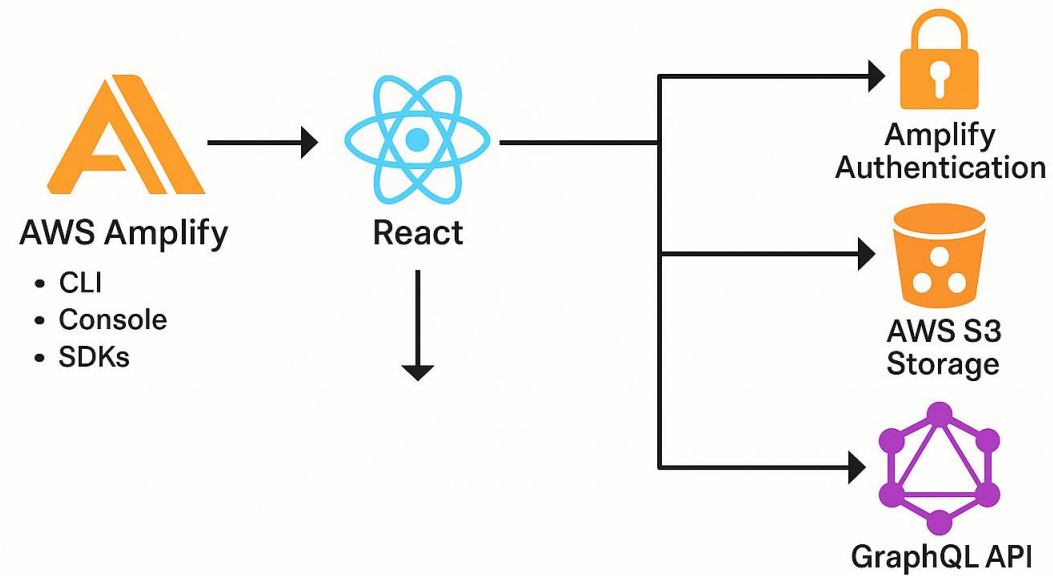
# Services Used

- **AWS Amplify:** Simplifies the process of integrating cloud services like authentication, storage, and hosting.
- **Amplify Authentication:** Provides easy user sign-up, sign-in, and secure user management.
- **Amplify Hosting:** Deploys and hosts the app with a secure, scalable solution.
- **AWS S3 Storage:** Used for storing and retrieving files such as images or documents.
- **GraphQL API:** Powers real-time data communication between the frontend and backend.



# Flow Diagram

## Architecture Diagram



# Implementation Process

**App Initialization:** Load resources and establish database connection (Firebase/MySQL).

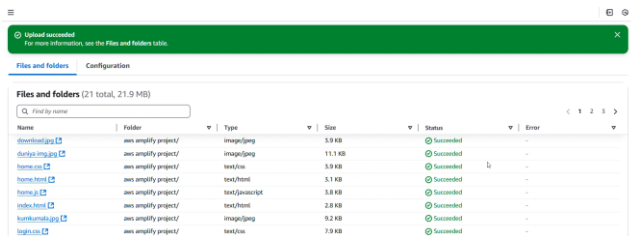
**Login/Signup:** User authentication via Firebase/Auth.

**Home Screen:** Display music and playlists fetched from the backend.

**Music Player:** Play music using MediaPlayer or ExoPlayer.

**Search:** Real-time search for tracks, albums, or artists.

**Settings & Logout:** Manage user preferences and sign out using Firebase.



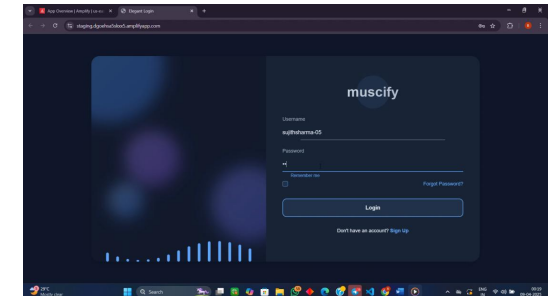
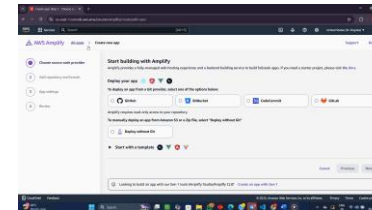
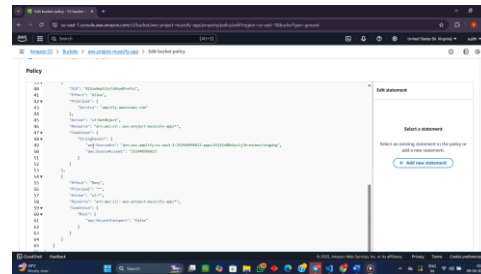
Upload succeeded  
For more information, see the Files and folders list.

Files and folders Configuration

Files and folders (21 total, 21.9 MB)

Find by name

Name	Folder	Type	Size	Status	Error
amazon.jpg	aws-empify-project/	image/jpeg	5.9 KB	Succeeded	-
amazon.png	aws-empify-project/	image/png	11.1 KB	Succeeded	-
home.png	aws-empify-project/	image/png	5.9 KB	Succeeded	-
home.html	aws-empify-project/	text/html	5.1 KB	Succeeded	-
home.js	aws-empify-project/	text/javascript	5.8 KB	Succeeded	-
index.html	aws-empify-project/	text/html	2.8 KB	Succeeded	-
home.html.js	aws-empify-project/	image/png	9.2 KB	Succeeded	-
home.js	aws-empify-project/	text/html	7.9 KB	Succeeded	-



# Key Features and Functionality

**User Authentication:** Users can sign up and log in via Firebase/Auth.

**Music Streaming:** Songs are streamed in real-time from AWS S3 or Firebase Storage.

**Playlist Management:** Users can create, edit, and manage personal playlists.

**Search Functionality:** Real-time search for tracks, artists, and albums.

**Push Notifications:** New music alerts via Firebase Cloud Messaging (FCM).

**User Preferences:** Settings for audio quality, theme mode (light/dark), etc.

**Home Screen:** Displays featured tracks, playlists, and search options.

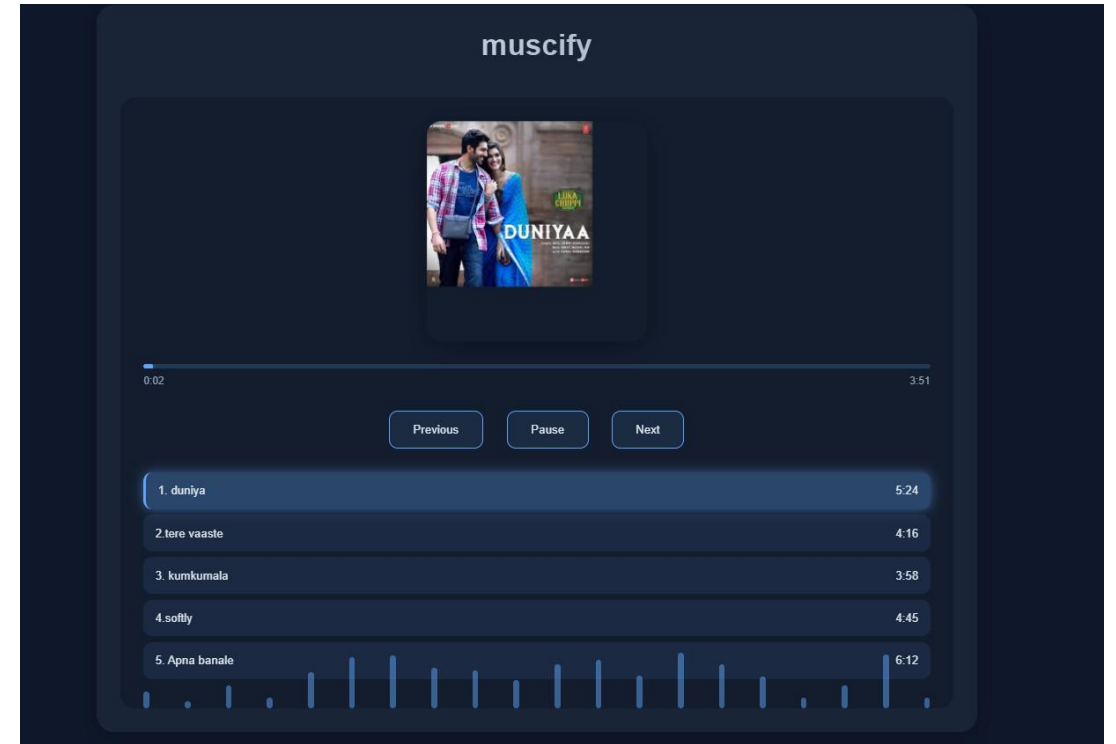
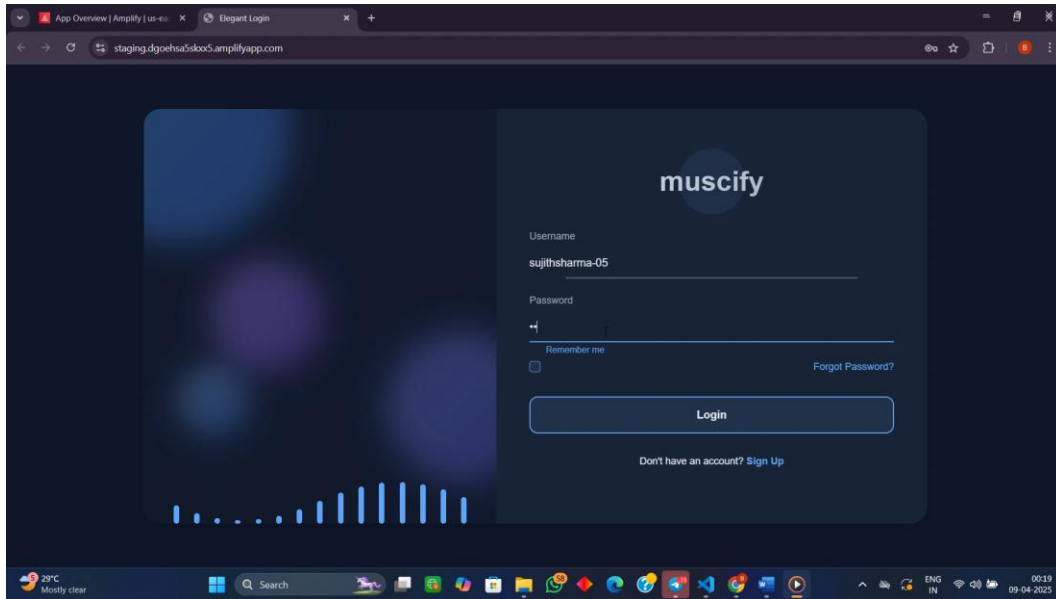
**Music Player:** Includes play/pause, skip buttons, and progress bar.

**AWS S3:** Stores and serves music files for streaming.

**AWS Lambda & RDS:** Manages backend processing and user data in MySQL.



# Results and Outputs



# Challenges and Solutions

- 1.**Music Streaming:** Latency in streaming was resolved by using **AWS S3** and **CloudFront** for fast, reliable delivery.
- 2.**User Authentication:** Secured login was achieved by integrating **Firebase Authentication** with token-based security.
- 3.**Search Functionality:** Real-time search was optimized using **Elasticsearch** or **Firebase Firestore** for quick data retrieval.
- 4.**Push Notifications:** **Firebase Cloud Messaging (FCM)** ensured timely notifications, even in the background.
- 5.**Data Synchronization:** Real-time syncing across devices was handled using **Firebase Firestore** or **AWS DynamoDB**.
- 6.**Debugging:** Used `console.log()` and `console.error()` to track errors in both frontend and backend.
- 7.**Firebase Debugging Tools:** Monitored Firebase logs for issues with authentication and real-time data syncing.
- 8.**Postman:** Tested and debugged API calls like music streaming and user management.
- 9.**AWS CloudWatch:** Monitored AWS services to track errors and performance issues.
- 10.**Unit Testing:** Used **Jest** and **Mocha** for unit testing frontend components and backend APIs.





# Learnings & Takeaways

- Technical Skills Gained:** Gained hands-on experience with **AWS S3**, **CloudFront**, **Firebase Authentication**, and **FCM** for push notifications.
- Backend Development:** Worked with **AWS Lambda** for backend processing and **AWS RDS/DynamoDB** for database management.
- Search Optimization:** Enhanced skills in **Elasticsearch** and **Firebase Firestore** for real-time search functionality.
- Cloud Monitoring:** Used **AWS CloudWatch** to monitor logs and set alarms for AWS services.
- API Testing:** Improved API testing skills with **Postman** and **Jest/Mocha** for unit testing.
- Soft Skills:** Enhanced **teamwork** by collaborating on cloud integrations and app functionality.
- Project Planning:** Strengthened **project planning** skills by breaking down app features into manageable tasks.
- Communication:** Developed better **communication** skills through regular updates and troubleshooting with the team.
- Certification:** Successfully completed **AWS Certified Solutions Architect - Associate**.
- Learning Growth:** Gained a deeper understanding of **cloud architecture**, **security**, and **real-time application development**.



# Future Scope

Possible extensions for the AMP&OFY Music App could include integrating AI-powered recommendation systems to offer personalized music suggestions based on user preferences and listening habits. Additionally, adding social features such as playlist sharing and user interactions could enhance engagement. For real-time deployment, leveraging AWS Lambda and Elastic Beanstalk would ensure auto-scaling to handle traffic spikes, providing a smooth user experience even during high demand. Cost optimization can be achieved by using AWS Spot Instances for compute resources and implementing S3 lifecycle policies to archive less frequently accessed music data. To scale efficiently, deploying an AWS Elastic Load Balancer can distribute incoming traffic, while AWS CloudFront can be used to cache and deliver content more quickly, reducing server load and improving performance.



# References

- [1] Amazon Web Services (AWS), "Amazon S3 User Guide," Amazon, 2021. [Online]. Available: <https://docs.aws.amazon.com/s3/index.html>.
- [2] Amazon Web Services (AWS), "Managing Access with AWS Identity and Access Management (IAM)," Amazon, 2021. [Online]. Available: <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>.
- [3] AWS Amplify, "Hosting with AWS Amplify," AWS, 2021. [Online]. Available: <https://docs.amplify.aws/hosting>.
- [4] Amazon Web Services (AWS), "Securing Your Static Website with AWS," AWS, 2021. [Online]. Available: <https://aws.amazon.com/blogs/security/securing-your-static-website-with-aws/>.
- [5] Amazon Web Services (AWS), "Overview of Static Website Hosting with Amazon S3," AWS, 2021. [Online]. Available: <https://aws.amazon.com/s3/storage-classes/static-website-hosting/>.
- [6] Cloud Academy, "Introduction to AWS Amplify," Cloud Academy, 2021. [Online]. Available: <https://cloudacademy.com/learn/aws-amplify/>.

•



*THANK YOU*

