EMOTITUNES

A Mini-Project Report Submitted
For
Partial Fulfilment of the Requirements of the
Degree of Bachelor of Engineering

In

COMPUTER ENGINEERING

(Semester VI)
By

Crystal Fernandes (9539)
Sanika Patankar(9563)

Under the guidance of

Prof. Roshni Padate

DEPARTMENT OF COMPUTER ENGINEERING
Fr. Conceicao Rodrigues College of Engineering
Bandra (W), Mumbai - 400050
University of Mumbai

2023-2024

This work is dedicated to my family.

I am very thankful for their motivation and support.

CERTIFICATE

This is to certify that the mini-project entitled **" EMOTITUNES "** is a bonafide work of Crystal Fernandes (9539) and Sanika Patankar(9563) submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Engineering** (Semester- V).

Prof. Roshni Padate
Guide/ Supervisor

Dr. Sujata Deshmukh
HOD Computer Engg.

Dr. S. S. Rathod
Principal

Approval Sheet

Mini Project Report Approval for T.E. (Semester-VI)

This mini-project report entitled **EMOTITUNES** submitted by Crystal Fernandes (9539), Sanika Patankar(9563) is approved for the degree of Bachelor of Engineering in **Computer Engineering** (Semester-VI).

Examiner 1._____

Examiner 2._____

Date:

Place: Mumbai

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date:

Crystal Fernandes (9539)
Sanika Patankar(9563)

Abstract

This study describes an innovative project that harnesses the capabilities of cloud computing and machine learning techniques to deliver highly personalised music recommendations based on users' facial expressions. The system is designed to analyse and interpret facial expressions in real-time, captured through a camera stream, utilising a sophisticated combination of HTML, CSS, and JavaScript for the frontend interface, Django for robust backend functionalities, TensorFlow for advanced machine learning algorithms, and OpenCV for precise facial image processing. Through this comprehensive integration, the system accurately predicts the user's emotional state, which serves as a pivotal factor in curating music selections tailored to the individual's mood. This approach not only enhances the user experience by offering music that resonates with their current emotional state but also showcases the seamless integration of cutting-edge technologies to deliver a sophisticated and engaging application.

Keywords: Cloud computing, machine learning, personalised recommendations, facial expressions, HTML, CSS, JavaScript, Django, TensorFlow, OpenCV, user experience

Acknowledgments

We have great pleasure in presenting the report on **"EMOTITUNES"**. I take this opportunity to express my sincere thanks towards the guide Prof. Roshni Padate, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We enjoyed discussing the work progress with him/her during our visits to the department.

We thank Dr. Sujata Deshmukh, Head of Computer Engineering department, Principal and the management of C.R.C.E., Mumbai for encouragement and providing necessary infrastructure for pursuing the project.

We also thank all non-teaching staff for their valuable support, to complete our project.

Date:
Crystal Fernandes (9539)
Sanika Patankar (9563)

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

In today's digital age, the intersection of technology and human emotion presents exciting opportunities for innovative applications. Our project embodies this convergence by leveraging a combination of advanced technologies to create an interactive web experience that responds to users' moods in real-time.

At its core, our web application utilises the Django web framework to provide a solid foundation for development, offering essential features such as authentication, routing, and database management. Building upon this foundation, we integrate cutting-edge technologies including TensorFlow and OpenCV to analyse and interpret facial expressions, enabling the system to detect users' moods accurately.

Furthermore, we employ Amazon Web Services (AWS) to enhance scalability, reliability, and security. By utilising Amazon S3 for storing static files such as images and audio, we ensure seamless access to multimedia resources while optimising performance. Additionally, our application leverages Amazon RDS for database management, enabling efficient data storage and retrieval.

Moreover, we enhance the user experience by integrating Amazon Cognito for secure authentication, providing users with a seamless login experience across devices. Finally, our application is hosted on Amazon EC2, ensuring high availability and scalability to meet user demand.

In this report, we provide an overview of the technologies utilized, the architecture of our system, and a detailed explanation of the implementation process. Additionally, we discuss the challenges encountered and the strategies employed to overcome them, as well as future enhancements and potential applications of our project.

Chapter 2

OBJECTIVES

1. Implement cloud computing infrastructure to host and manage the system, ensuring scalability, availability, and efficient resource utilisation.

2. Provide personalised music recommendations based on users' emotional states, enhancing the overall user experience by delivering content that aligns with their current mood.

3. User Authentication: Implement user authentication and authorization using Amazon Cognito, providing a secure login mechanism for users and ensuring access control to the application's features and resources.

4. Static File Management: Store static files such as song images and MP3 files in an Amazon S3 bucket, allowing for efficient storage, retrieval, and management of media assets.

5. Database Connectivity: Establish connectivity to an RDS database to store and manage application data, such as user preferences, song metadata, and mood detection results.

6. Emotion Detection Integration: Integrate TensorFlow and OpenCV libraries into the Django website to accurately detect the mood of a person in real-time from images or video streams.

7. Scalable Hosting: Ensure the Django website's scalability and availability, enabling seamless user access without specifying the hosting platform.

Chapter 3

SCOPE OF THE PROJECT

The "Emotitunes" initiative intends to improve music recommendations by making individualised suggestions based on users' emotional states, ultimately improving the entire user experience with content that matches their present mood.

This project includes a thorough integration of AWS Services such as S3, RDS, Cognito, and EC2, which will allow for more effective administration of static file storage, database operations, user authentication, and deployment procedures.

The system uses TensorFlow to perform real-time analysis of users' facial expressions to accurately predict their emotional states, while OpenCV is used to precisely extract key features and information from captured facial expressions, resulting in efficient and effective facial image processing.

"Emotitunes" seeks to provide a smooth and personalised music experience based on each user's emotional environment by integrating innovative technologies.

Chapter 4

PROPOSED SYSTEM

4.1 PROBLEM STATEMENT

In contemporary web development, achieving personalized user experiences that adapt to users' emotions poses a challenge. Existing systems often struggle to integrate real-time emotion detection with dynamic content adaptation, resulting in static and impersonal interactions. Key obstacles include:

- Cloud-based Scalability: Systems lack flexibility in scaling resources based on demand, leading to performance bottlenecks during peak usage.
- Reliability and Availability: Maintaining high availability and reliability in distributed environments requires robust cloud infrastructure to mitigate downtime and service disruptions.
- Cost Management: Balancing performance with cost-effectiveness is challenging, as over-provisioning leads to unnecessary expenses, while under-provisioning compromises performance.

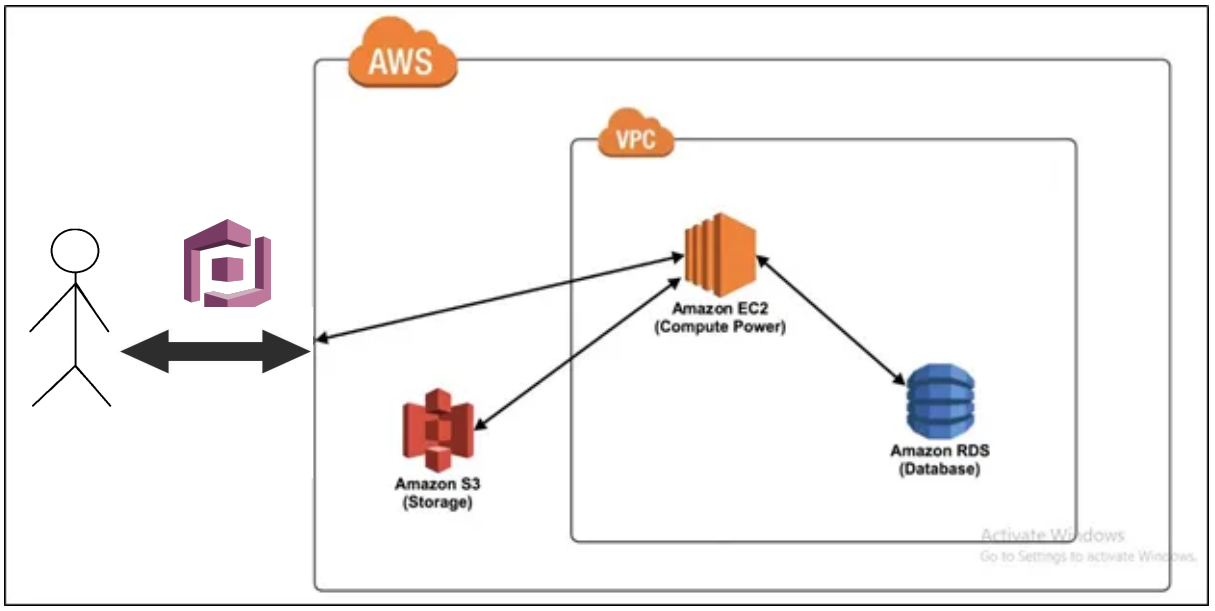
Our Solution Approach:

Our project addresses these challenges by leveraging Amazon Web Services (AWS) to build a scalable and resilient web application infrastructure. By utilizing AWS services such as Amazon EC2 for scalable compute resources, Amazon S3 for reliable storage, Amazon RDS for database management, and Amazon Cognito for secure authentication and user management, we aim to:

- Design a scalable architecture capable of dynamically provisioning compute resources to ensure optimal performance.
- Establish a robust data storage infrastructure for maintaining data integrity and availability.
- Provide a seamless and secure login experience for users, adhering to best practices in user authentication and authorization.
- Through the integration of AWS services, our project aims to create a foundation for delivering personalised and engaging user experiences while addressing the technical challenges of scalability, reliability, and cost management.

Chapter 5

SYSTEM DESIGN



5 AWS High Level Architecture

- **EC2 Instances:** The EC2 instances hosting the web application code are typically deployed within one or more subnets of the VPC. Public-facing EC2 instances serving web traffic are usually placed in public subnets, while backend instances may reside in private subnets for enhanced security.
- **RDS Database:** The RDS database is deployed within a private subnet of the VPC to restrict direct access from the Internet. This ensures that the database remains secure and isolated from unauthorized access.
- **S3 Bucket:** S3 buckets are not directly deployed within subnets, as they are accessed over the Internet by default. However, you can control access to S3 buckets using bucket policies, IAM policies, and VPC endpoints for enhanced security.
- **Cognito User Pool:** While Cognito itself does not reside within subnets, the applications using Cognito (such as web servers hosted on EC2 instances) can be deployed within the VPC's subnets to control inbound and outbound traffic and enforce security policies.

IMPLEMENTATION

CODE:

To connect our RDS to our Django application, we followed these steps:

- We began by configuring PostgreSQL in the RDS console.
- Then, within Django settings, we updated the default database from SQLite to PostgreSQL. Included details like name, user, password, host and port number.
- Migrated all tables to database by using the command makemigration

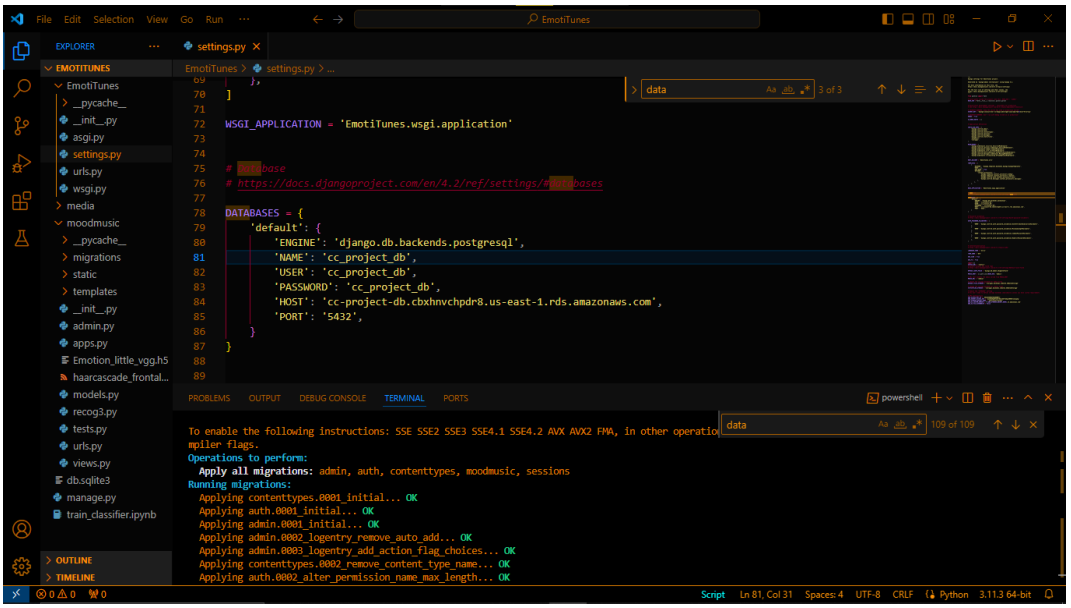


Fig. 6.1. RDS Connection

To connect our S3 bucket to our Django application, we followed these steps:

Generating Access Key ID and Secret Key:

- Created a new user in the IAM console with programmatic access and assigned the necessary policy to access the S3 bucket, enabling interaction with AWS services.
- Obtained the Access Key ID and Secret Access Key from the IAM console. These credentials serve as the authentication mechanism for securely accessing AWS resources.

Integration with Django settings.py:

- Integrated the access credentials into the Django application's settings.py file.
- Defined the AWS access key ID and secret key as environment variables within settings.py:

Declared the default file storage and static file storage and ran the command collectstatic

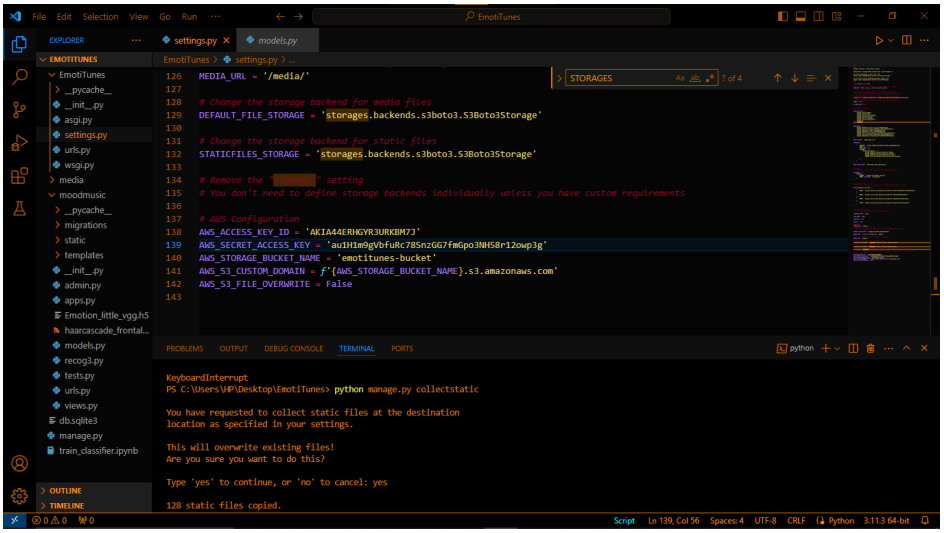


Fig 6.2. AWS S3 Connection

To connect our AWS Cognito to our Django application, we followed these steps:

Setting up Cognito User Pool:

- We configured a Cognito User Pool, defining user attributes and authentication methods ensuring that the appropriate client app settings were configured.

Implementing Cognito Authentication in Django:

- We utilized the lambda_handler function to interact with AWS Cognito to verify user tokens and retrieve user information. We integrated the necessary configuration parameters into our Django views.py including the Cognito region name, user pool ID, and client ID.

Handling Authentication Flow in Views:

- The getTokens function handles the token exchange process with Cognito, exchanging the authorization code for user tokens.

Maintaining User Sessions:

- We maintained user sessions through the getSession function which retrieves the user session token from cookies, allowing users to remain authenticated.

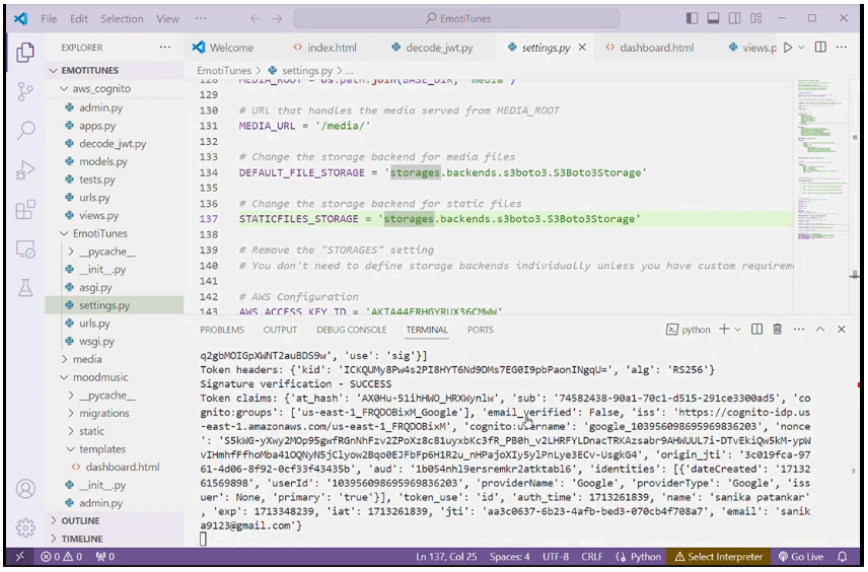


Fig 6.3. AWS Cognito [getTokens and verification]

LOGIN/SIGNUP USING AWS COGNITO

After seamlessly integrating AWS Cognito with our Django app, we leveraged its hosted UI domain to provide users with a branded and intuitive authentication experience. Offering seamless login options, users could effortlessly sign in or opt for single sign-on via Google. Configuring OAuth settings with Cognito's domain name and creating OAuth credentials ensured smooth integration for Google sign-in. Additionally, Cognito facilitated email verification upon registration, increasing security measures.

To enhance usability, we maintained user sessions and securely stored user information using cookies. With the flexibility of customizable authentication flows, we tailored the login experience to suit our app's specific requirements, ensuring a seamless and secure user journey.

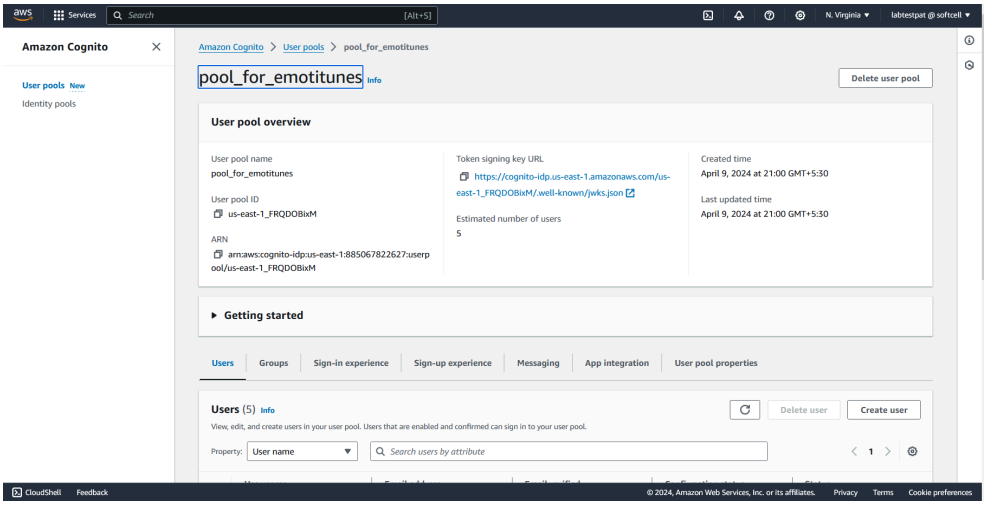


Fig 6.4. AWS Cognito User pool

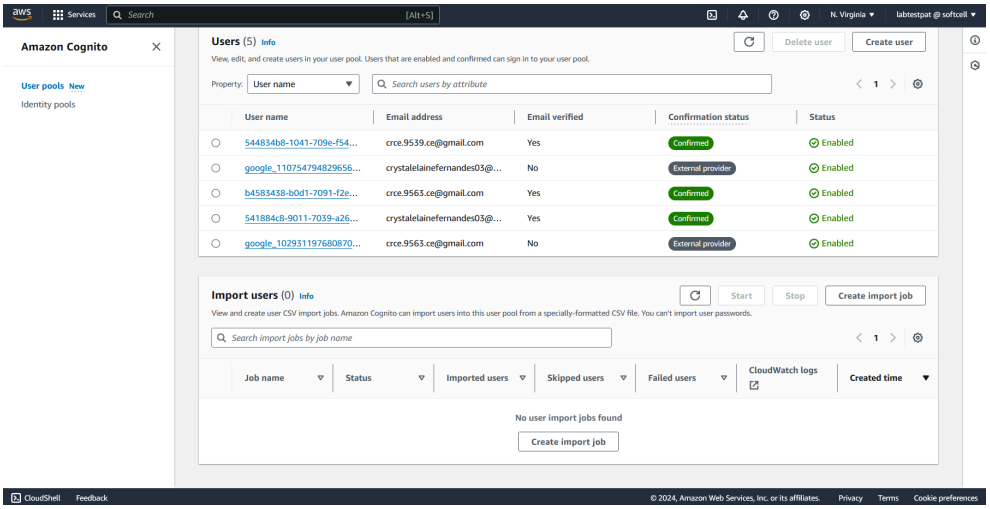


Fig 6.5. Viewing Our Authenticated Users

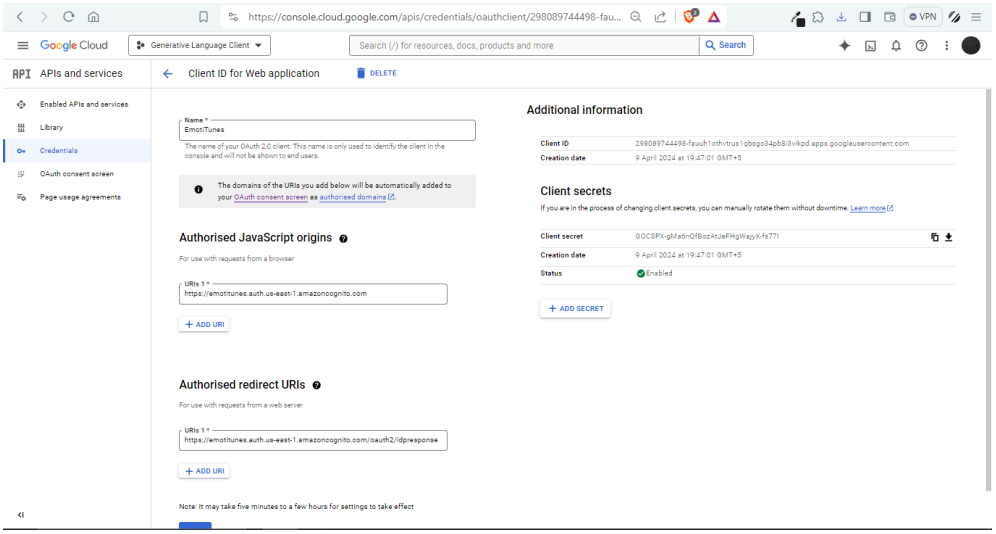


Fig 6.6. GCP OAuth Credentials for Single-sign on

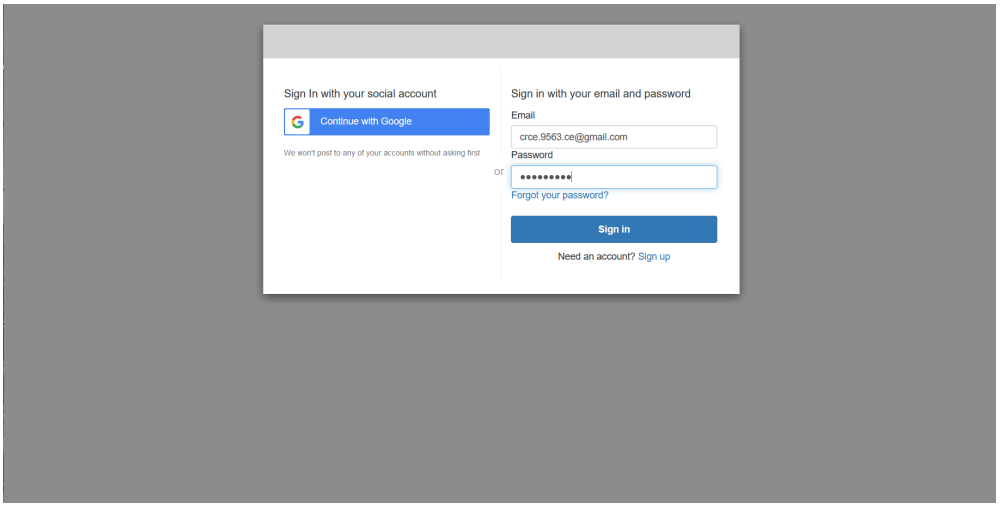
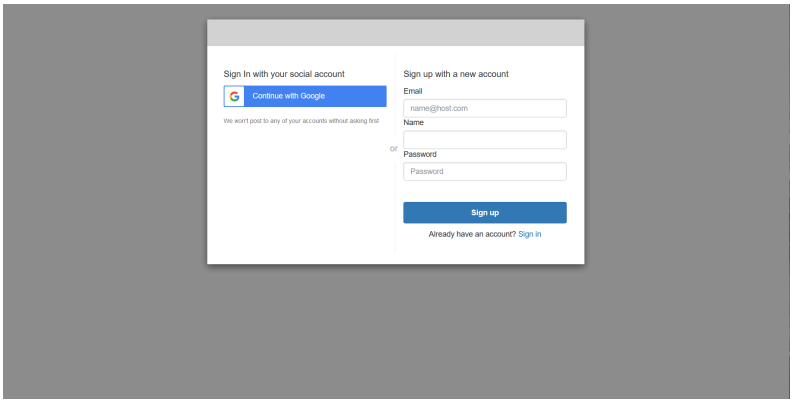


Fig 6.7. AWS Cognito Login Page



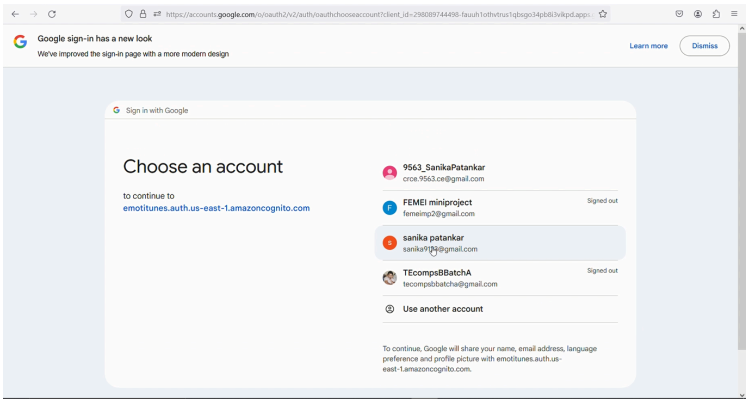


Fig 6.8.a. AWS Cognito Signup Page

Fig 6.8.b. Single-Sign on Page

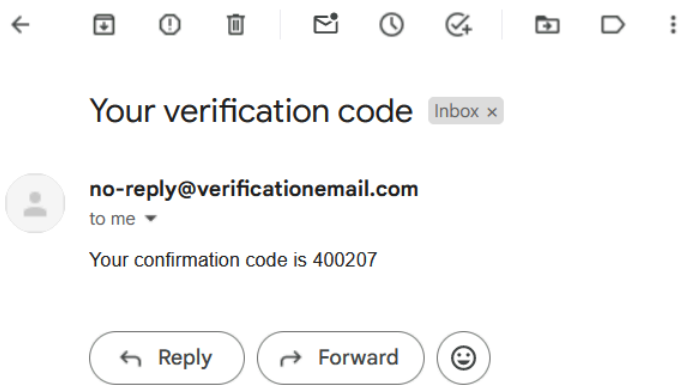


Fig 6.9. AWS Cognito Verification Code

S3 BUCKET

Following the configuration of our Django website to interface with the S3 bucket, administrators were equipped with a seamless process for uploading songs and album images. During this process, MP3 files were automatically organized within the "song_mp3" directory, with each file sorted into subdirectories based on mood. Similarly, album images were efficiently stored in the "song_images" directory within the S3 bucket.

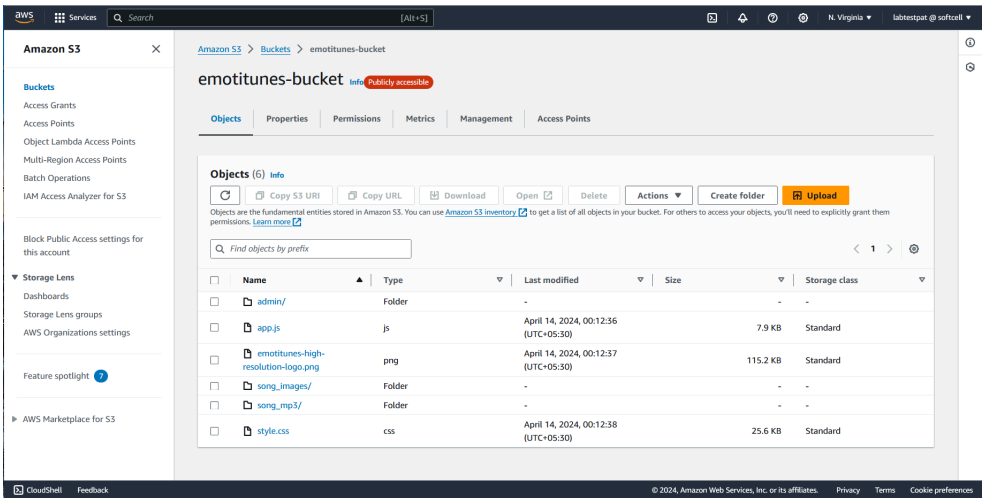


Fig 6.10. AWS S3 Bucket containing all static and media files

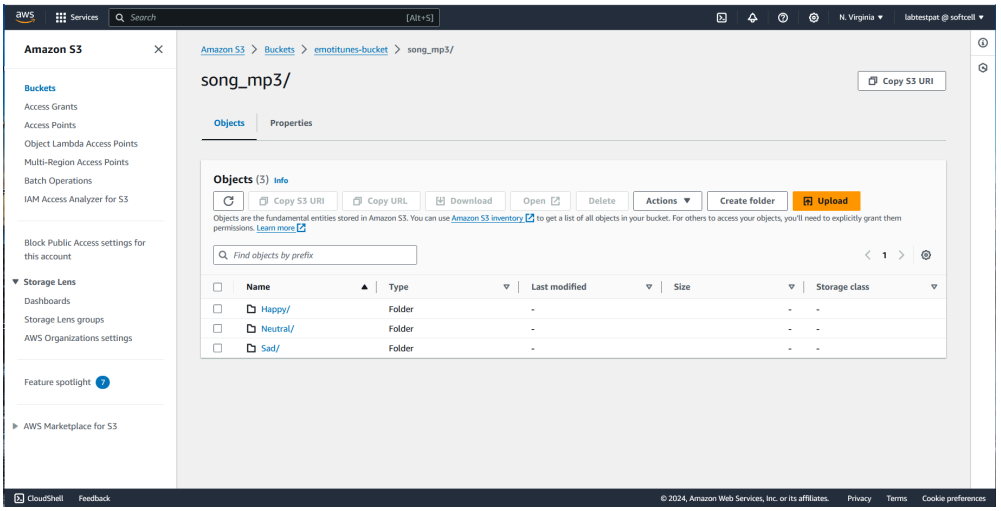


Fig 6.11. S3 Song Files

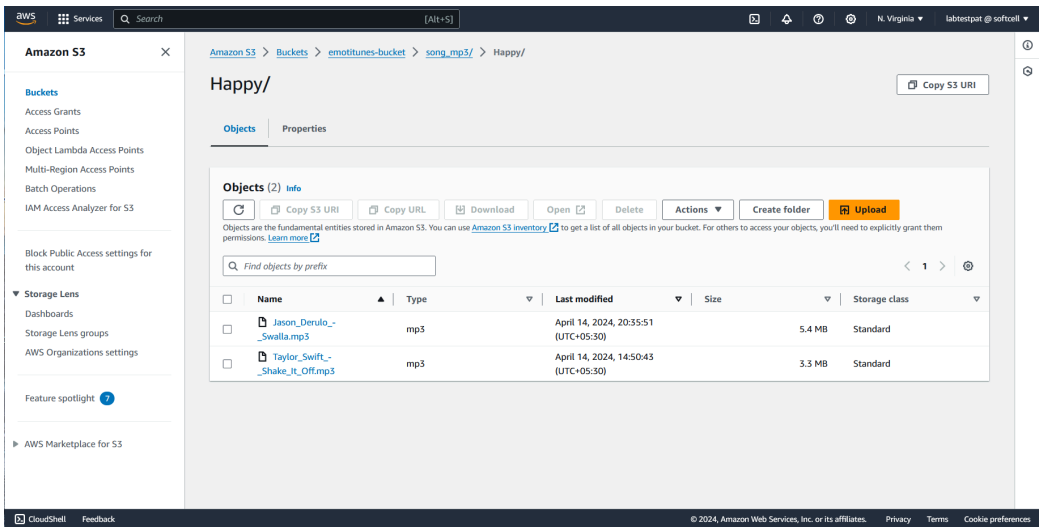


Fig 6.12. Songs in Happy Mood Folder

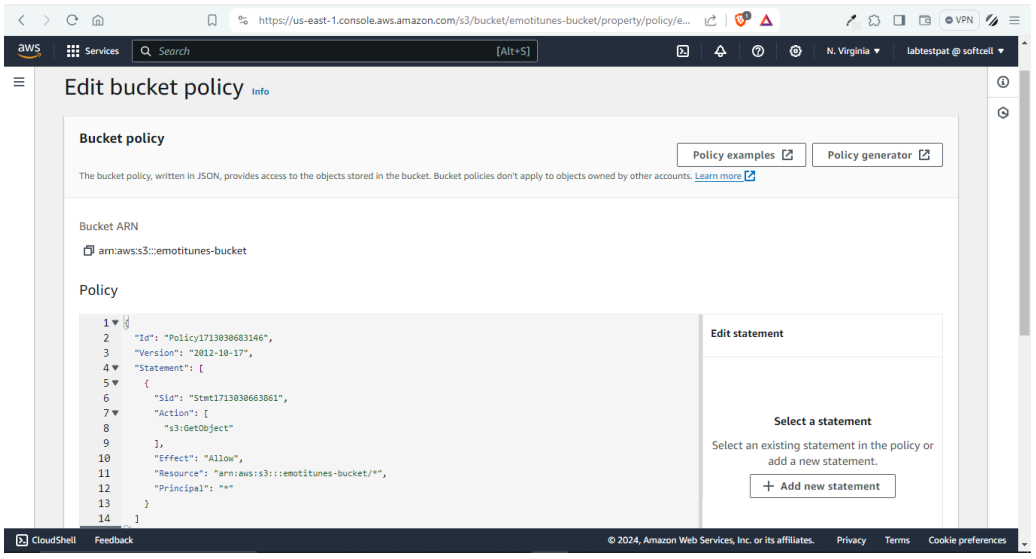


Fig 6.13. Bucket Policy Updated allowing GetObject

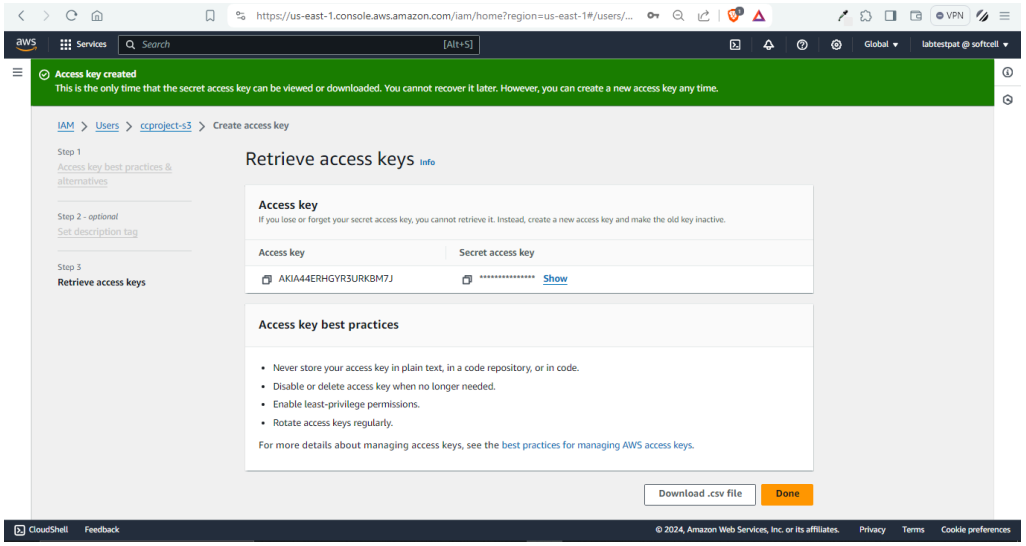


Fig 6.14. Created Access Key after creating user

RDS (DATABASE)

Facing issues with RDS database creation, we initiated VPC configuration, setting up subnets and attaching an Internet Gateway to facilitate outbound and inbound traffic. Creating a new route table, we associated subnets and configured a route for Internet-bound traffic. Additionally, DNS resolution and DNS hostname settings were enabled for seamless resource communication within the VPC. These actions resolved connectivity issues, allowing successful RDS database creation and operation.

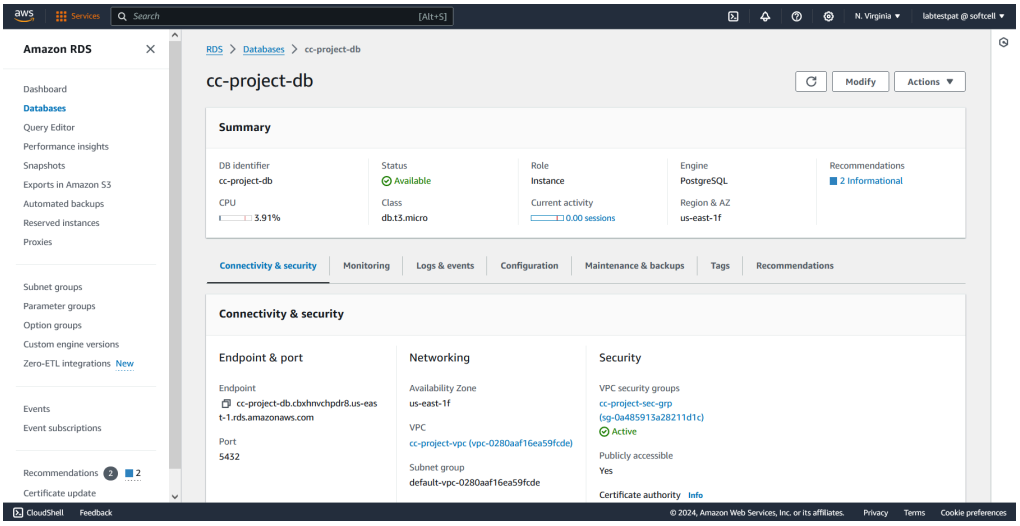


Fig 6.16. AWS RDS

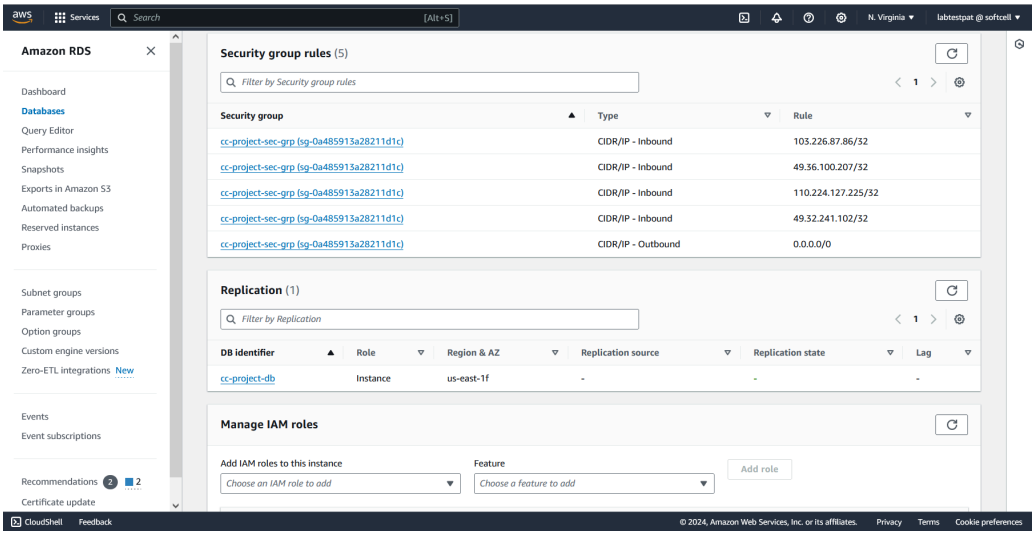


Fig 6.16. AWS RDS Security Groups

SECURITY GROUPS(VPC)

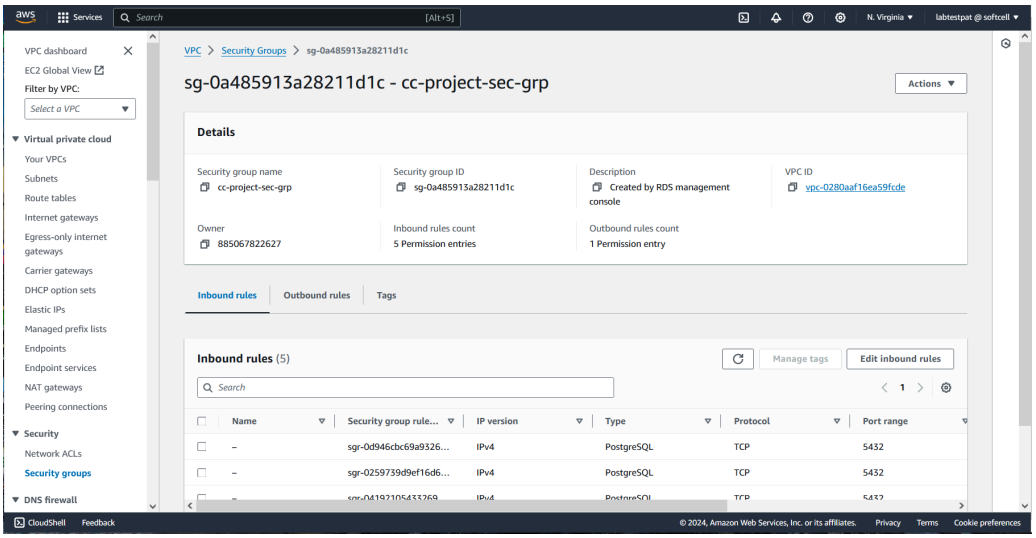


Fig 6.17. AWS VPC Security Groups

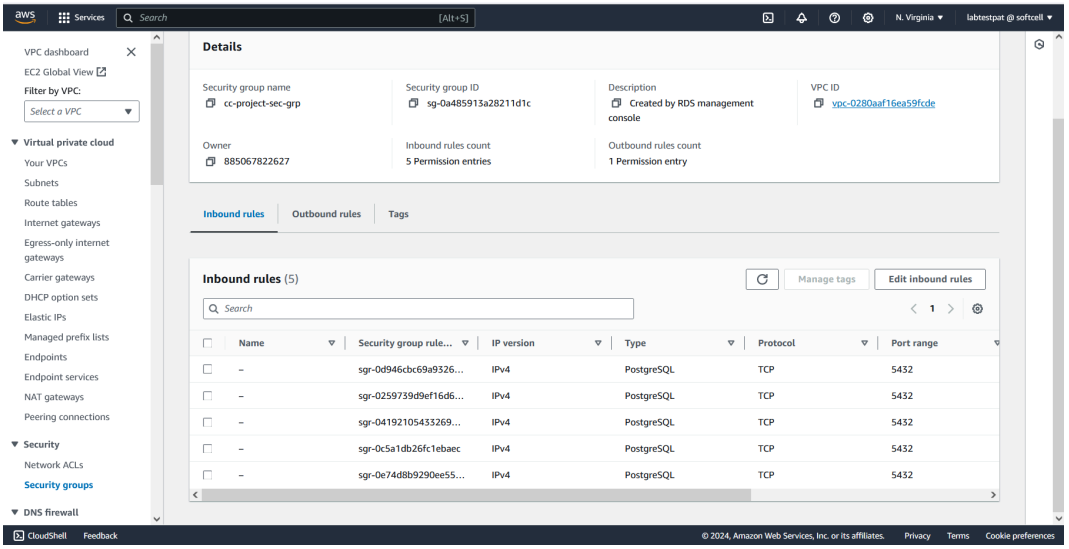


Fig 6.18. AWS VPC Security Groups (Inbound rules)

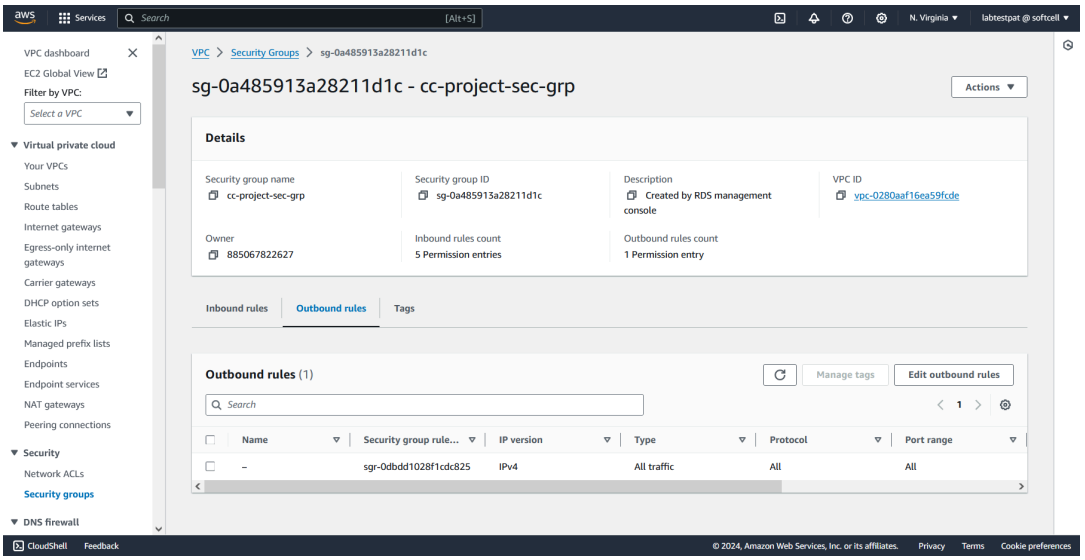


Fig 6.19. AWS VPC Security Groups (Outbound rules)

EC2 INSTANCES

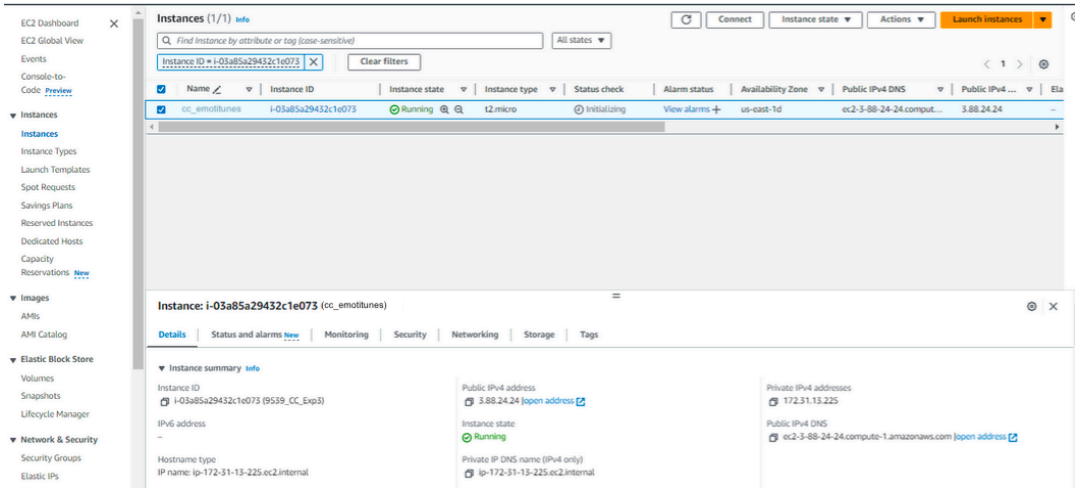


Fig 6.20. AWS EC2 INSTANCE

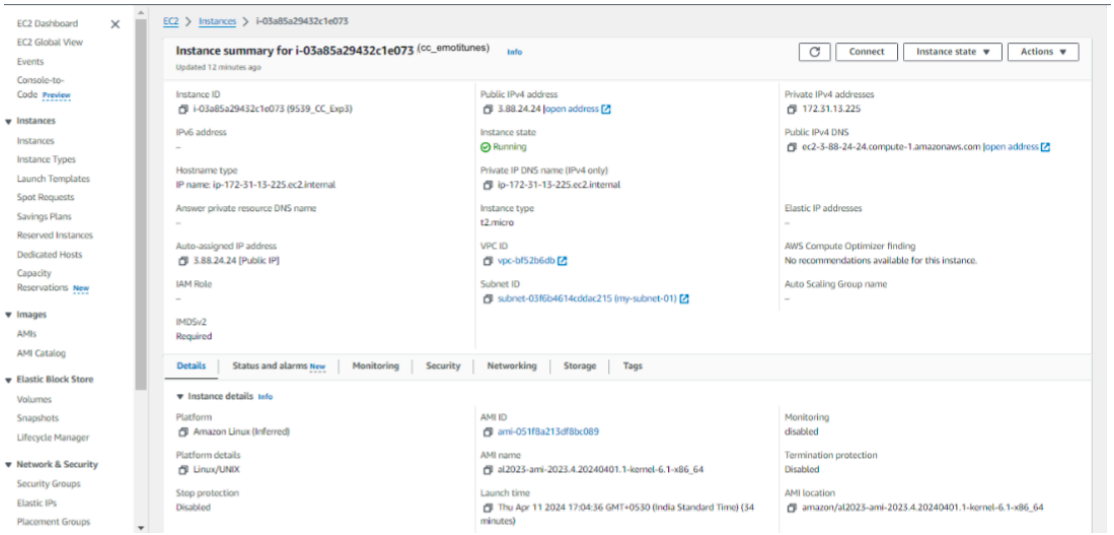


Fig 6.21. EC2 INSTANCE

Chapter 7

RESULTS

The images below depict the final website structure of the application.
The photographs also offer distinct mood predictions based on their facial expressions, as well as music recommendations for that specific mood.

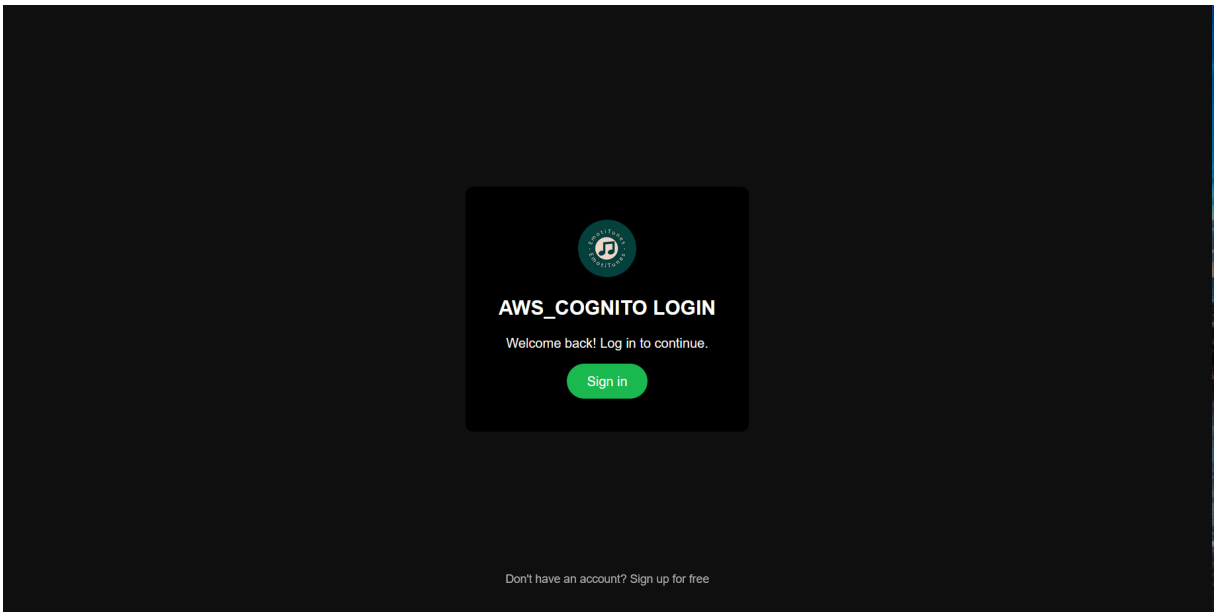


Fig 7.1. Login Page

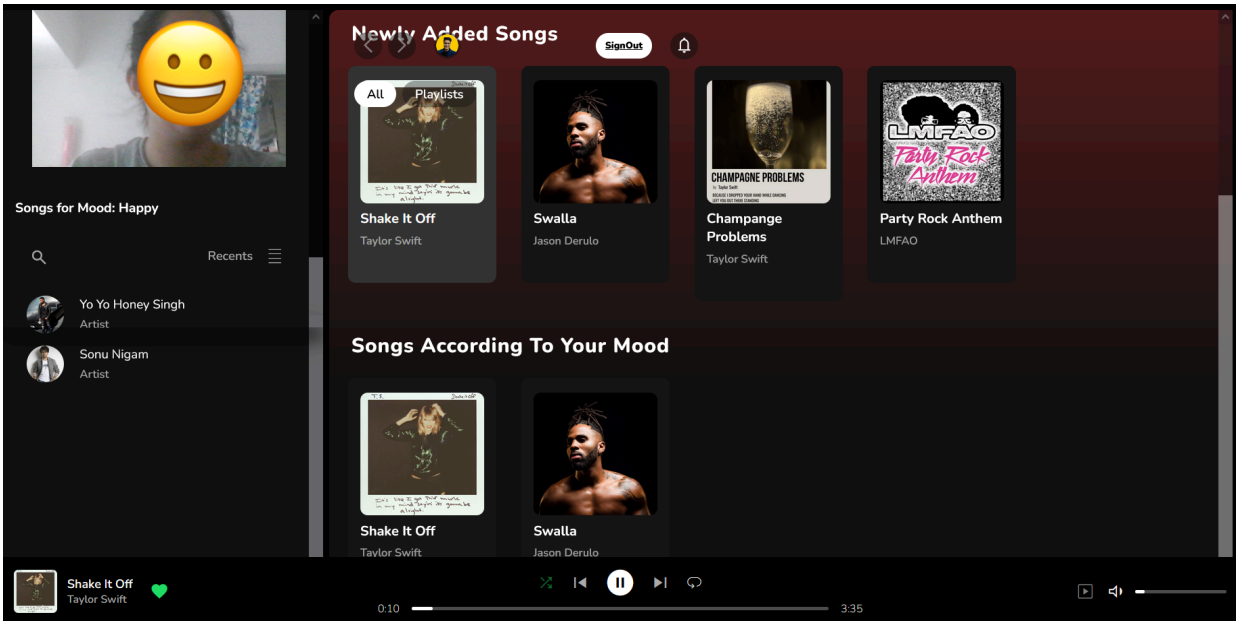


Fig 7.2. Happy Mood

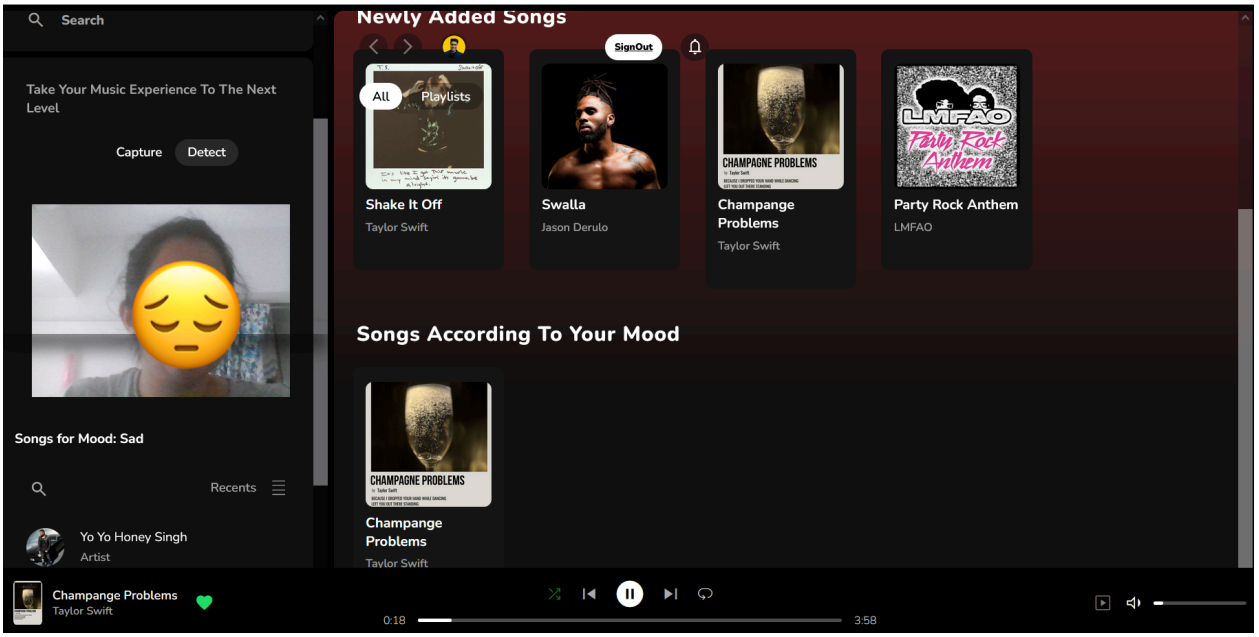


Fig 7.3. Sad Mood

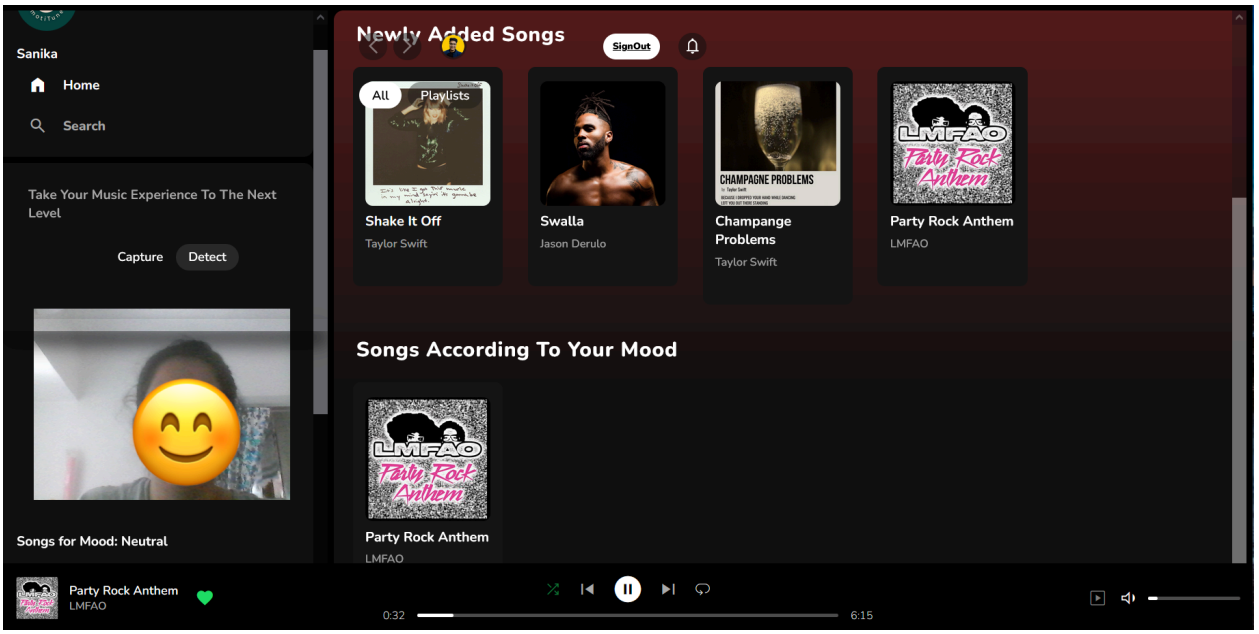


Fig 7.4. Neutral Mood

Chapter 7

CONCLUSION

By establishing a cloud-based architecture, we provide scalability, availability, and effective resource use for the system's continuous operation. The Django-based backend framework supports data processing and communication between the user interface and machine learning models. TensorFlow enables the system to create and deploy real-time facial expression analysis models, which result in accurate predictions of user moods. Furthermore, OpenCV integration enables accurate and efficient face image processing, as well as the extraction of crucial information from facial emotions. Finally, by adapting music suggestions to users' emotional states, the system customises the user experience and offers information that is relevant to their present mood.

In short, this research presents a complete and resilient system that uses cutting-edge technology to generate individualised and emotionally-aware music.