CinemaPulse – Real-Time Customer Feedback Analysis Powered by AWS

Project Description :

CinemaPulse is a cloud-inspired, full-stack web application designed to collect, process, and analyze real-time movie audience feedback. The platform enables viewers to submit ratings and reviews, while producers, analysts, and administrators gain access to structured insights, analytics dashboards, and management tools.

The system is developed using **Flask (Python)** for backend processing, **HTML, CSS, and JavaScript** for frontend interfaces, and **SQLite** for secure local data storage. The application follows a role-based architecture with four user roles: **Viewer, Producer, Analyst, and Admin**, ensuring controlled access and secure system operations.

CinemaPulse is designed as a **concept simulation of AWS cloud architecture**, where core components are mapped to AWS services:

1. Flask Backend → AWS EC2 (Scalable compute)
2. SQLite Database → Amazon RDS (Managed database service – conceptual)
3. Role-Based Authentication → AWS IAM (Secure access control – conceptual)

This architecture enables the platform to simulate a cloud-native, scalable,

and secure OTT analytics system. CinemaPulse helps studios and production houses understand audience sentiment, feedback trends, engagement levels, and performance insights in real time, making it a powerful decision-support platform for the entertainment industry.

Key Objectives :

- To collect real-time audience feedback and ratings for movies
- To analyze user sentiment using rule-based logic
- To provide role-based dashboards for decision-making
- To simulate cloud-based scalable architecture using AWS concepts
- To ensure secure access and data integrity
- To build a premium OTT-style user experience with modern UI/UX

Scenario 1: **Scalable Infrastructure for Movie Feedback Collection**

During movie premieres or blockbuster releases, viewer feedback tends to spike significantly. CinemaPulse uses AWS EC2 to provide a scalable infrastructure that can automatically adjust to handle fluctuating loads of customer feedback in real-time. Whether it's managing a small indie release or a major Hollywood blockbuster, Flask serves the backend by collecting viewer ratings, comments, and reactions. The EC2 environment ensures that CinemaPulse can scale as needed, ensuring a smooth and seamless experience for viewers and studios alike, without compromising performance or uptime during peak activity.

Scenario 2: **Real-Time Data Analytics with Amazon RDS**

CinemaPulse leverages Amazon RDS for efficient data storage and real-time analysis of movie feedback. Using RDS with MySQL, the platform stores customer reviews, ratings, and sentiment data securely, while

benefiting from automated backups, high availability, and rapid scalability. This managed database solution enables film studios to get immediate insights into audience reactions, allowing them to track feedback trends, identify audience preferences, and make data-driven decisions in real-time, all while ensuring data reliability and security.

Scenario 3: Secure Access to Movie Analytics with AWS IAM

Given the sensitive nature of audience feedback and proprietary data, CinemaPulse ensures robust access control using AWS IAM. The platform implements role-based permissions so that only authorized personnel, such as film producers, marketing teams, and data analysts, can access specific parts of the system. This secure setup allows stakeholders to view critical movie feedback, analyze viewer trends, and adjust marketing strategies, all while maintaining strict security protocols to protect viewer privacy and intellectual property. Combined with AWS EC2's scalability, this approach ensures both secure access and efficient handling of large datasets.

AWS Scenario Mapping

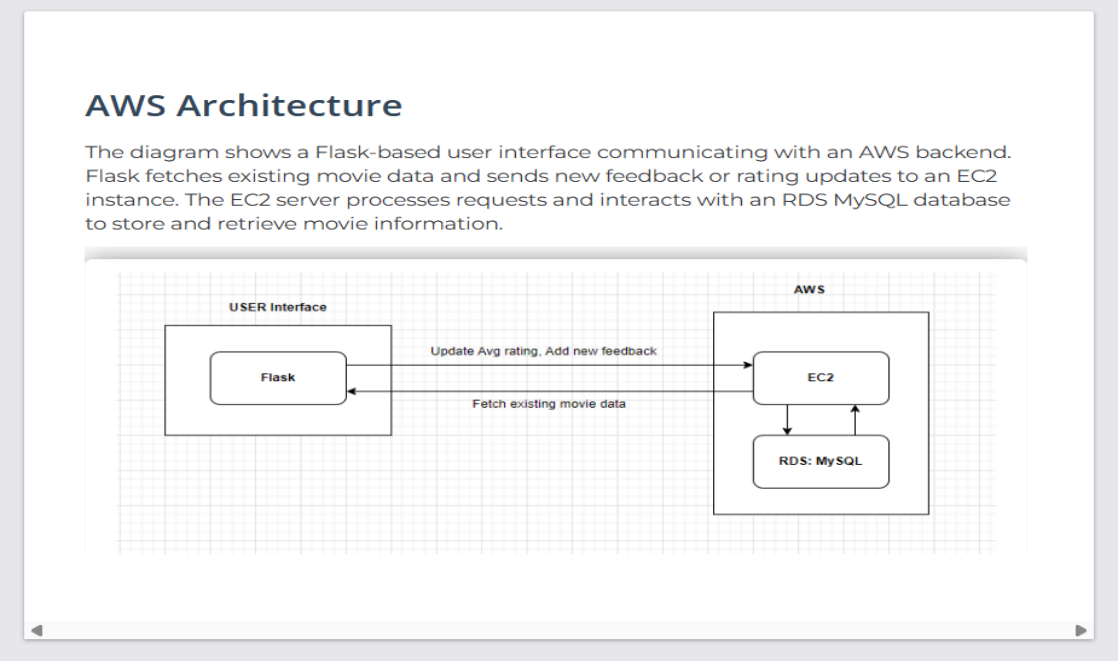
Scenario Used: AWS Scenario 3 – Secure Access to Movie Analytics

CinemaPulse is implemented based on **AWS Scenario 3**, which focuses on secure access control, role-based permissions, and protected cloud data services. The system uses AWS IAM roles to manage secure communication between EC2-hosted Flask services and DynamoDB cloud storage through the boto3 SDK. This scenario ensures controlled access, data integrity, and secure analytics processing for real-time customer feedback and movie analytics.

AWS Architecture :

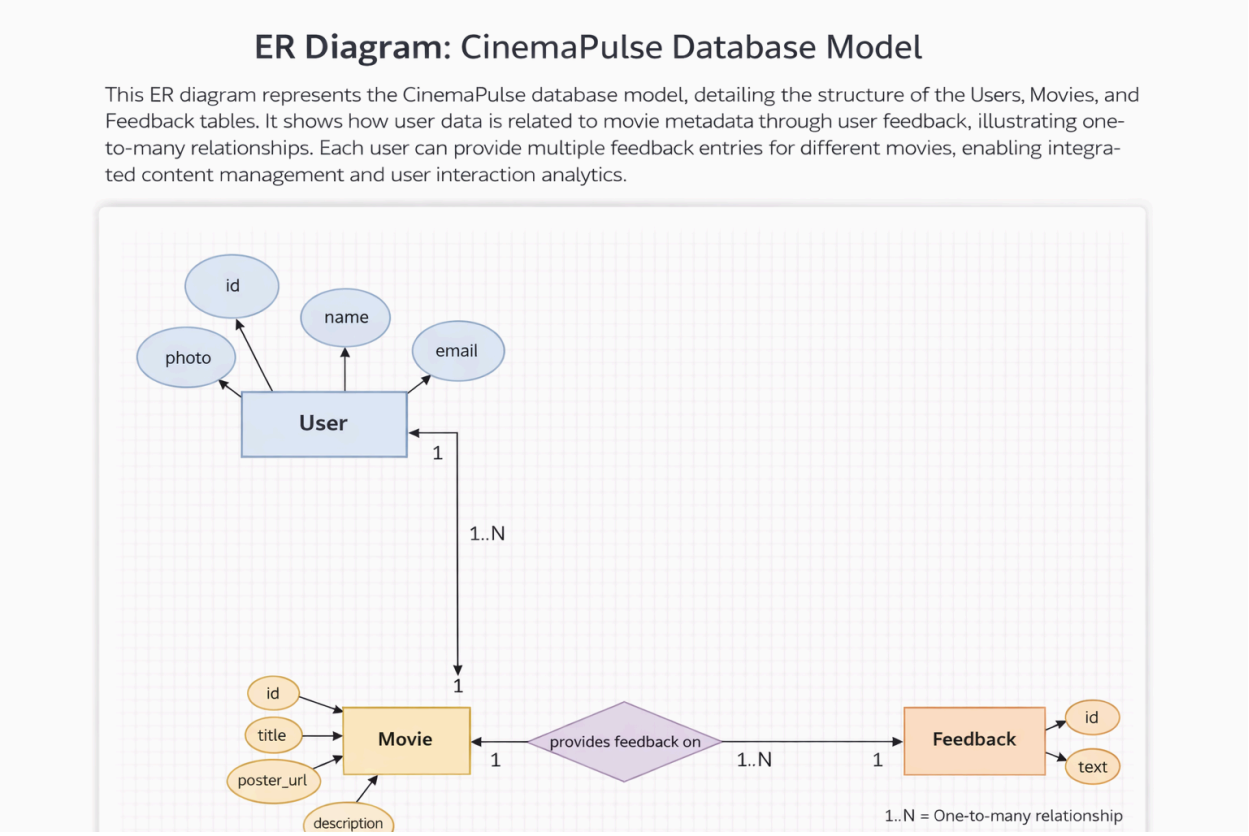
The diagram shows a Flask-based user interface communicating with an AWS backend. Flask fetches existing movie data and sends new feedback or rating updates to an EC2 instance. The EC2 server processes requests and interacts with an RDS MySQL database to store and retrieve movie information.

:



ER diagram

The ER diagram represents Movie and Feedback tables. Movie stores details like id, title, description, image URL, average rating, and total reviews. Feedback stores user name, email, rating, and comments. Each feedback entry is linked to a movie, showing a one-to-many relationship



Pre-Requisites

1. AWS Account Setup: https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh

2. Understanding of IAM: https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR

3. Knowledge of Amazon EC2 : https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz

4. RDS : https://www.youtube.com/live/MPau9c7PT74?si=A8OK-zFGbSKkAFWN

5. MySQL WorkBench: https://youtu.be/wALCw0F8e9M?si=ovMF9qMx5rLxaznB

CinemaPulse – Real-Time Customer Feedback Analysis

🔹 Milestone 1: AWS Account Setup and Login

Activity 1.1: AWS Account Creation

- AWS account setup
- Billing configuration
- Basic security configuration

Activity 1.2: AWS Management Console Access

- Login to AWS console
- Access EC2 and IAM services

🔹 Milestone 2: Local Development Environment Setup

Activity 2.1: Development Tools Installation

- Install Python
- Install Node.js & npm
- Install VS Code
- Project folder structure creation

Activity 2.2: Project Initialization

- Flask project setup (app.py)
- Frontend setup (HTML, CSS, Vanilla JS structure)
- Environment configuration (.env)
- Dependency installation (pip install, npm install)

🔹 Milestone 3: Database Creation and Setup (SQLite)

Activity 3.1: SQLite Database Creation

- Create cinema_pulse.db
- SQLite connection configuration with Flask

Activity 3.2: Table Creation

- users
- movies
- feedback

Verified via Flask backend logic (init_db() concept)

🔹 Milestone 4: Backend Development (Flask API)

Activity 4.1: Flask Server Setup

- Flask app initialization
- CORS configuration
- Session management

Activity 4.2: API Development

- /api/login

- /api/signup
- /api/movies
- /api/feedback (GET, POST)
- /api/feedback/<id> (PUT, DELETE)

Activity 4.3: Business Logic

- Role-based access handling
- Feedback sentiment logic (rule-based)
- Rule-based analytics logic

🔹 Milestone 5: Frontend Development (HTML, CSS, Vanilla JS)

Activity 5.1: UI Structure Development

- Landing page
- Role selection page
- Login page
- Signup page
- User home
- Movie detail pages
- Dashboards (User, Analyst, Producer, Admin)

Activity 5.2: UI/UX Implementation

- Neon theme
- Glassmorphism design
- Animations
- Popups
- Cinematic UI components

🔹 Milestone 6: Frontend–Backend Integration

Activity 6.1: API Bridge Integration

- Frontend ↔ Flask API connection
- fetch() integration
- JSON data mapping
- Error handling & fallback logic

Activity 6.2: Data Synchronization

- Feedback sync
- Movie sync
- User session sync

🔹 Milestone 7: Authentication & Role-Based Access

Activity 7.1: Authentication System

- Login
- Signup
- Session handling

Activity 7.2: Role-Based Routing

- User access
- Producer access
- Analyst access
- Admin access

Milestone 8: AWS EC2 Instance Setup

Activity 8.1: EC2 instance creation

- Instance launch
- Key pair setup
- Security group configuration

Activity 8.2: Server environment setup

- Python installation
- Node.js installation
- Dependency installation
- Project upload

Milestone 9: Deployment on AWS EC2

Activity 9.1: Backend deployment

- Flask server hosting on EC2
- Public IP configuration
- Port configuration

Activity 9.2: Frontend deployment

- React build
- Static hosting via EC2
- API connection configuration

🔹 Milestone 10: Security Configuration (AWS Scenario3)

Activity 10.1: AWS IAM configuration

- Role-based permissions
- Secure access control

Activity 10.2: Application-level security

- Role validation
- Route protection
- Session security

📌 Mapped to AWS Scenario 3: Secure Access to Movie Analytic

🔹 Milestone 11: Testing and Validation

Activity 11.1: Functional testing

- Login
- Signup
- Feedback
- Rating
- Dashboard

Activity 11.2: Integration testing

- Frontend ↔ Backend
- Backend ↔ SQLite

Activity 11.3: Deployment testing

- EC2 public IP testing
- API testing
- Stability testing

🔹 Milestone 12: Final Documentation and Submission

Activity 2.1: Final deployment validation

- Stability check
- Security check

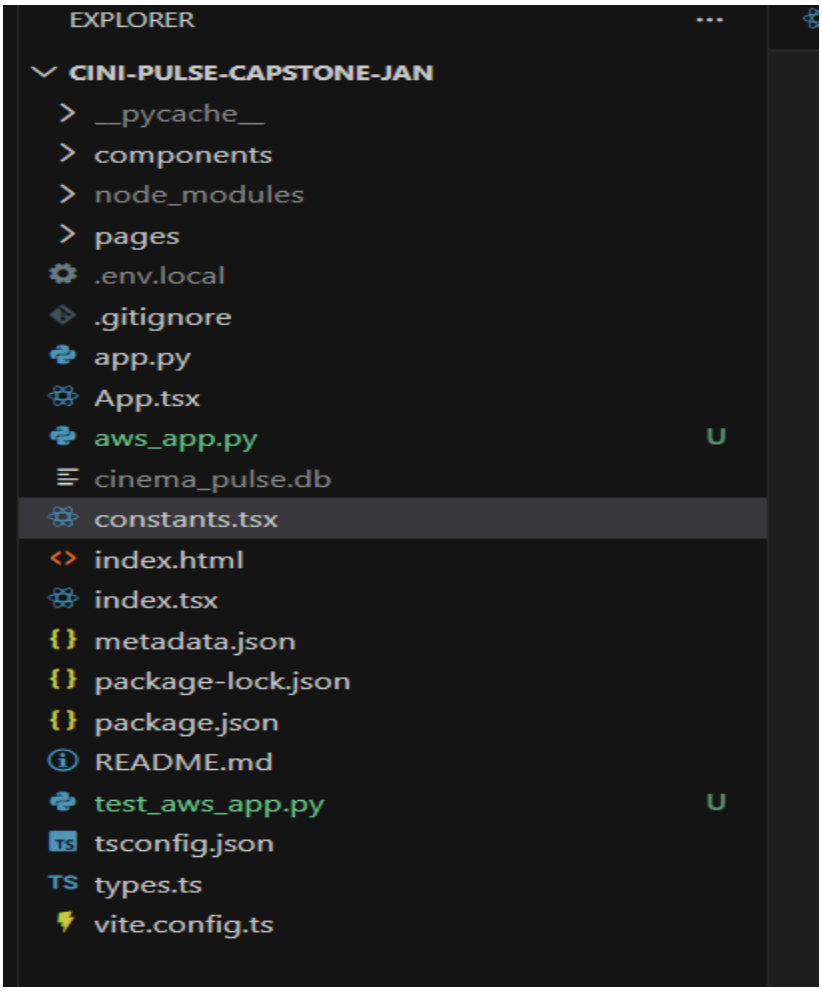
Activity 12.2: Documentation preparation

- Project workflow
- Architecture
- Database design
- AWS integration
- Scenario mapping
- Screenshots
- Conclusion

Project File Structure Description

Root Directory

CINI-PULSE-CAPSTONE-JAN/



→ Main project root directory containing frontend, backend, database, AWS integration, and configuration files.

Folders

__pycache__/

→ Python cache directory automatically generated for Flask backend execution optimization.

components/

→ Reusable React UI components such as Navbar, UI elements, cards, and layout components.

pages/

→ Application page-level components including LandingPage, LoginPage, SignupPage, Dashboards, and MovieDetails pages.

node_modules/

→ Node.js dependency directory containing installed frontend libraries and packages.

Environment & Config Files

.env.local

→ Environment configuration file used to store sensitive values such as API URLs, secret keys, and environment variables.

.gitignore

→ Git configuration file to exclude sensitive files, build files, databases, and dependencies from version control.

Backend Files (Flask + AWS)

app.py

→ Core Flask backend server file handling API routes, session management, CORS configuration, SQLite database connection, and business logic.
(Includes: app = Flask(__name__), API routes, SQLite integration)

aws_app.py

→ AWS-integrated Flask backend using **boto3** for DynamoDB operations, IAM authentication, and cloud-based data storage.
(Includes: boto3.resource('dynamodb'), DynamoDB tables, AWS API logic)

test_aws_app.py

→ Testing file for validating AWS DynamoDB connectivity and API functionality.

Database Files

cinema_pulse.db

→ SQLite database file used for local development and testing, storing users, movies, and feedback data.

Frontend Core Files (React + TypeScript)

App.tsx

→ Main React application controller file handling routing, state management, role-based navigation, API integration, and UI flow.

index.tsx

→ React entry point file responsible for rendering the root React component into the DOM.

index.html

→ HTML base template file for mounting the React application.

Data & Constants

constants.tsx

→ Centralized configuration file containing movie data, static content, and predefined UI constants.

types.ts

→ TypeScript interface definitions for system entities such as User, Movie, Feedback, Notification, and AppState.

Build & Tooling Configuration

vite.config.ts

→ Vite build configuration file for frontend bundling, development server, and optimization.

tsconfig.json

→ TypeScript compiler configuration for React project structure, module resolution, and type checking.

package.json

→ Node.js project configuration file containing scripts, dependencies, and project metadata.

package-lock.json

→ Auto-generated dependency lock file ensuring consistent package versions across environments.

Documentation

README.md

→ Project documentation file containing setup instructions, run commands, project description, and usage guidelines.

metadata.json

→ Project metadata configuration file for internal system reference and integration mapping

AWS Integration Mapping

AWS SDK Integration

→ `import boto3` **used for connecting Flask backend to AWS services.**

DynamoDB Connection

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

```
# AWS Configuration
REGION = 'us-east-1'
dynamodb = boto3.resource('dynamodb', region_name=REGION)
```

DynamoDB Tables

```
# DynamoDB Tables
users_table = dynamodb.Table('Users')
movies_table = dynamodb.Table('Movies')
feedback_table = dynamodb.Table('Feedback')
```

UsersTable(users)

→ Stores user authentication and profile information including role-based access control, notification preferences, and user identity management for the CinemaPulse system.

Movies Table (movies)

→ Stores complete movie metadata including title, genre, cast, category, rating, and release information for content display, analytics, and feedback mapping.

Feedback Table

The `feedback` table stores real-time user reviews, ratings, and sentiment data for movies. It acts as a core analytics entity that connects users and movies, enabling sentiment analysis, customer satisfaction tracking, and movie performance evaluation in the CinemaPulse system.

This table supports both:

- **Local storage (SQLite)** for development
- **Cloud storage (AWS DynamoDB)** for production deployment

```
# Tables
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id TEXT PRIMARY KEY,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        role TEXT NOT NULL,
        photo TEXT,
        notifications_enabled INTEGER DEFAULT 1
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS movies (
        id TEXT PRIMARY KEY,
        title TEXT NOT NULL,
        poster TEXT,
        description TEXT,
        genre TEXT,
        category TEXT,
        cast TEXT,
        director TEXT,
        hero TEXT,
        heroine TEXT,
        vibe TEXT,
        release_type TEXT,
        rating REAL
    )
''')
```

Flask Initialization

```
app = Flask(__name__)
```

→ Initializes Flask backend server for API handling.

CORS Configuration

```
app = Flask(__name__)
# In production, set this via Environment Variables
app.secret_key = os.environ.get('SECRET_KEY', 'cinema_pulse_ultra_secret_key')
CORS(app, supports_credentials=True)
```

→ Enables secure frontend-backend communication.

Authentication & User Management

Login API (`/api/login`)

→ Handles user authentication by validating user credentials from the database, creating a session, and enabling secure role-based access to the CinemaPulse system.

Signup API (`/api/signup`)

→ Manages new user registration by securely storing user details in the database, assigning roles, and enabling controlled access to the

application.

```python
@app.route('/api/login', methods=['POST'])
def login():
    data = request.json
    email = data.get('email')
    conn = get_db_connection()
    user = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
    conn.close()

    if user:
        user_dict = dict(user)
        session['user_id'] = user_dict['id']
        return jsonify(user_dict)
    return jsonify({'error': 'User not found'}), 404

@app.route('/api/signup', methods=['POST'])
def signup():
    data = request.json
    conn = get_db_connection()
    try:
        conn.execute('''
            INSERT INTO users (id, name, email, password, role, notifications_enabled)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (data['id'], data['name'], data['email'], data['password'], data['role'], 1))
        conn.commit()
        return jsonify({'status': 'success'})
    except sqlite3.IntegrityError:
        return jsonify({'error': 'Email already exists'}), 400
    finally:
        conn.close()
```

Feedback Management & Analytics

```python
@app.route('/api/feedback', methods=['GET', 'POST'])
def handle_feedback():
    conn = get_db_connection()
    if request.method == 'POST':
        data = request.json
        conn.execute('''
            INSERT INTO feedback (id, movie_id, user_id, user_name, rating, text, created_at, sentiment)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', (data['id'], data['movieId'], data['userId'], data['userName'],
              data['rating'], data['text'], data['createdAt'], data['sentiment']))
        conn.commit()
        conn.close()
        return jsonify({'status': 'success'})

    rows = conn.execute('SELECT * FROM feedback ORDER BY created_at DESC').fetchall()
    conn.close()
    return jsonify([dict(row) for row in rows])

@app.route('/api/feedback/<id>', methods=['DELETE', 'PUT'])
def modify_feedback(id):
    conn = get_db_connection()
    if request.method == 'DELETE':
        conn.execute('DELETE FROM feedback WHERE id = ?', (id,))
        conn.commit()
    elif request.method == 'PUT':
        data = request.json
        conn.execute('''
            UPDATE feedback SET rating = ?, text = ?, sentiment = ? WHERE id = ?
        ''', (data['rating'], data['text'], data['sentiment'], id))
        conn.commit()
    conn.close()
    return jsonify({'status': 'success'})
```

Feedback API (`/api/feedback`)

→ Manages real-time customer feedback by enabling users to submit ratings and reviews, storing sentiment data, and retrieving feedback for analytics and dashboard visualization.

Feedback Modification API (`/api/feedback/<id>`)

→ Supports feedback lifecycle management by allowing update and deletion off feedback records to maintain data accuracy and content moderation.

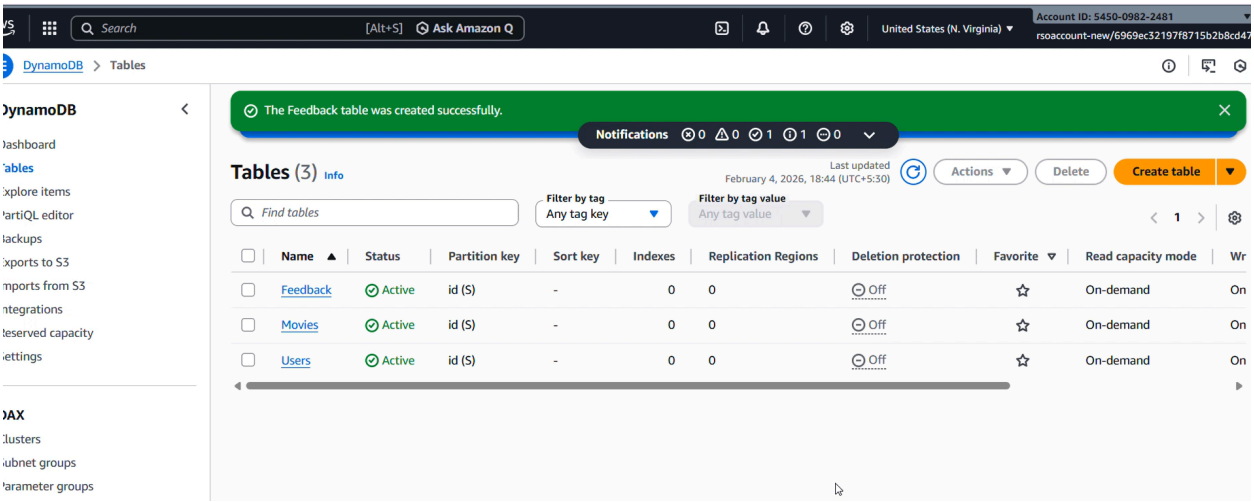
Cloud Feedback Management (AWS DynamoDB)

Cloud Feedback API (`/api/feedback` – DynamoDB Version)

→ Handles cloud-based feedback storage and retrieval using AWS DynamoDB, enabling scalable, secure, and real-time customer feedback management in the CinemaPulse system.

DynamoDB

Figure: AWS DynamoDB Table for CinemaPulse Feedback Storage
This screenshot shows the DynamoDB table created for the CinemaPulse system. The table is used to store real-time user feedback data and analytics information. The Flask backend connects to DynamoDB using the boto3 SDK with IAM-based secure authentication, enabling scalable and secure cloud data management.



```python
@app.route('/api/feedback', methods=['GET', 'POST'])
def handle_feedback():
    try:
        if request.method == 'POST':
            data = request.json
            feedback_item = {
                'id': data.get('id', str(uuid.uuid4())),
                'movie_id': data['movieId'],
                'user_id': data['userId'],
                'user_name': data['userName'],
                'rating': int(data['rating']),
                'text': data['text'],
                'created_at': data['createdAt'],
                'sentiment': data['sentiment']
            }
            feedback_table.put_item(Item=feedback_item)
            return jsonify({'status': 'success'})

        # GET Feedback
        response = feedback_table.scan()
        sorted_feedback = sorted(response.get('Items', []), key=lambda x: x['created_at'], reverse=True)
        return jsonify(sorted_feedback)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

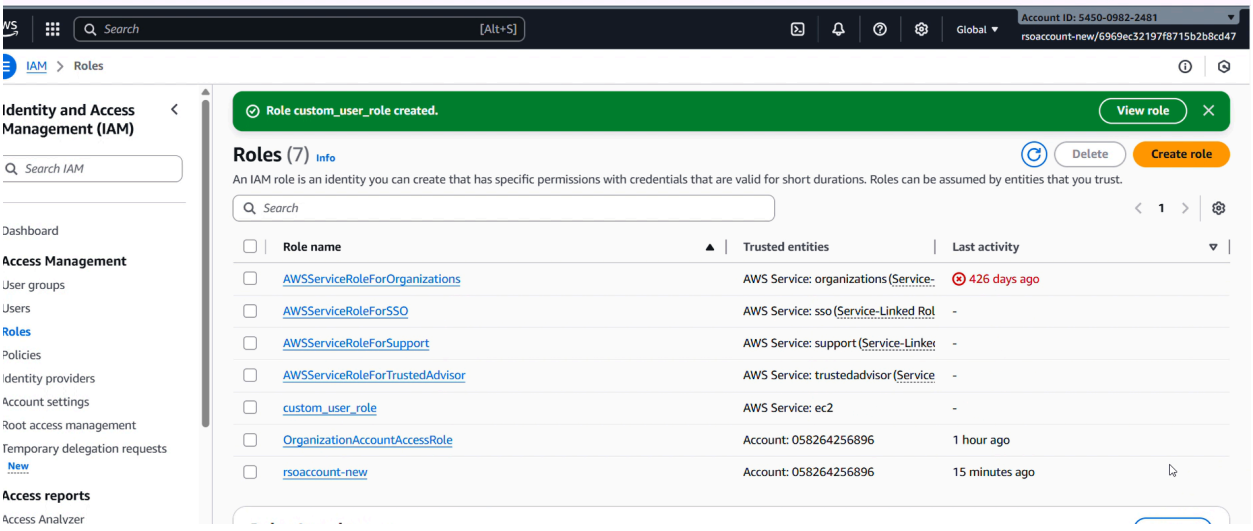
Application Server Configuration

This configuration enables the Flask application to run as a network-accessible server on AWS EC2. By binding the application to 0.0.0.0, the server listens on all available network interfaces, allowing external access through the EC2 public IP address. This setup supports cloud deployment, public API access, and frontend-backend communication in a distributed architecture. The application runs on port 5000, which is configured in the EC2 security group to allow inbound traffic.

```python
# Start App
if __name__ == '__main__':
    # Use 0.0.0.0 to allow external access via EC2 Public IP
    app.run(host='0.0.0.0', port=5000)
```

AWS IAM Integration (Security Architecture)

CinemaPulse implements AWS Identity and Access Management (IAM) to secure cloud service communication between the Flask backend and AWS resources. The EC2 instance hosting the Flask application is attached with a dedicated IAM role, enabling secure, credential-less access to AWS DynamoDB through the boto3 SDK. This approach eliminates hardcoded AWS credentials and enforces controlled access policies. IAM permissions are configured to allow only required operations such as PutItem, Scan, UpdateItem, and DeleteItem on DynamoDB tables, ensuring secure cloud data access, compliance with AWS security best practices, and protection against unauthorized access



AWS EC2 Integration

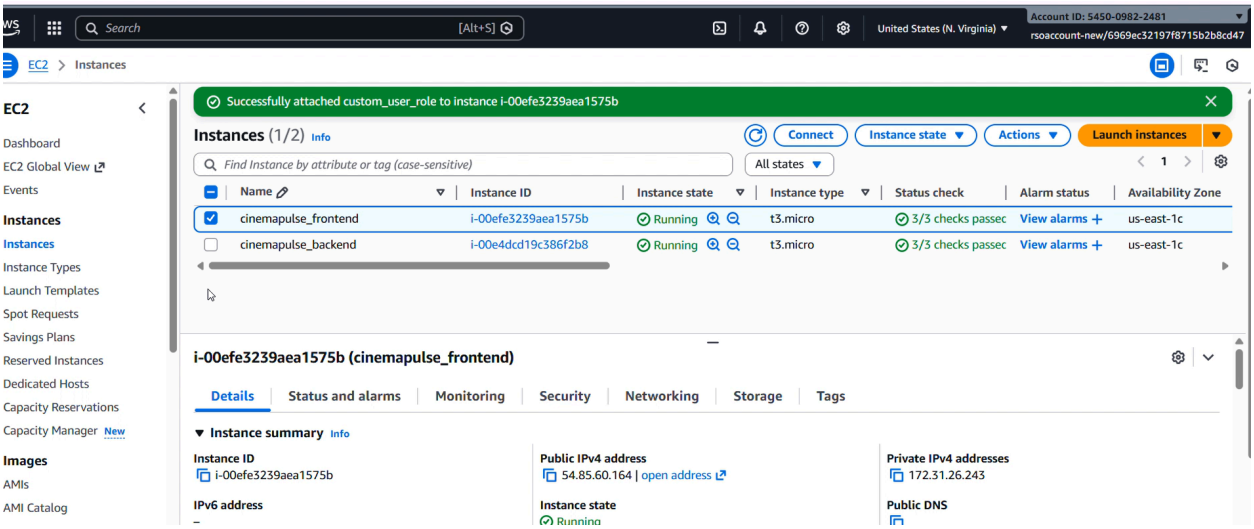
Description

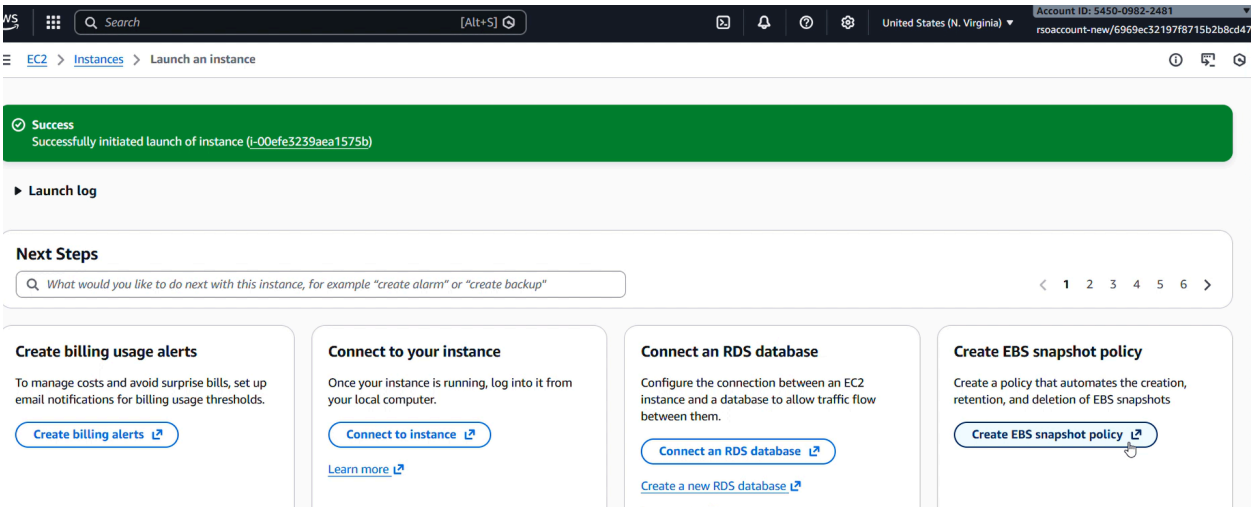
Amazon Elastic Compute Cloud (EC2) is used as the core cloud infrastructure to host the CinemaPulse backend application. The Flask server is deployed on an EC2 instance, enabling public access to APIs through the EC2 public IP address. This provides scalable compute resources, high availability, and cloud-based deployment for the application.

🔹 Purpose in Project

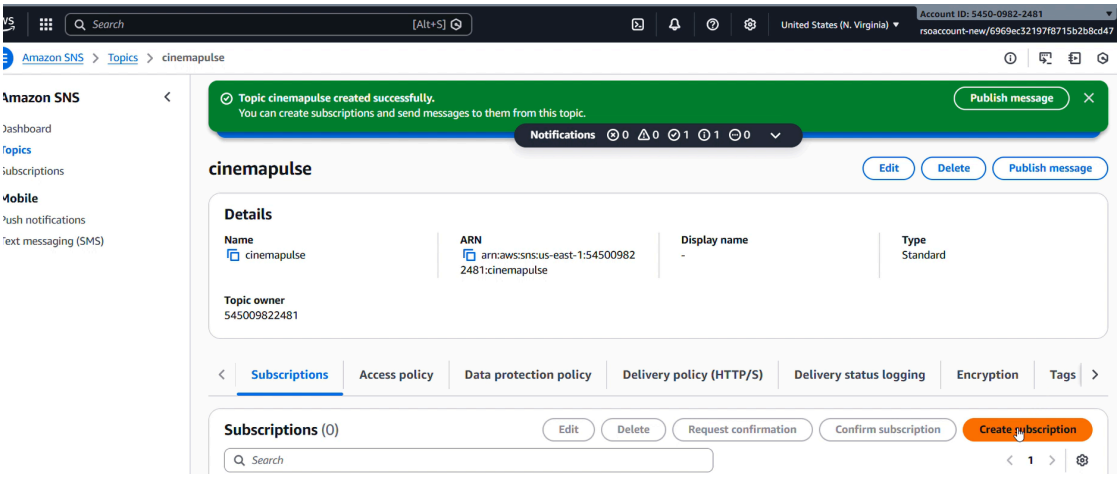
EC2 is used to:

- Host the Flask backend server
- Run the AWS-integrated Flask application (aws_app.py)
- Enable public API access via EC2 Public IP
- Connect securely to AWS services using IAM roles
- Act as the core execution environment for cloud deployment





SNS CREATED SUCCESSFULLY :



SUBSCRIPTION CREATED SUCCESSFULLY:

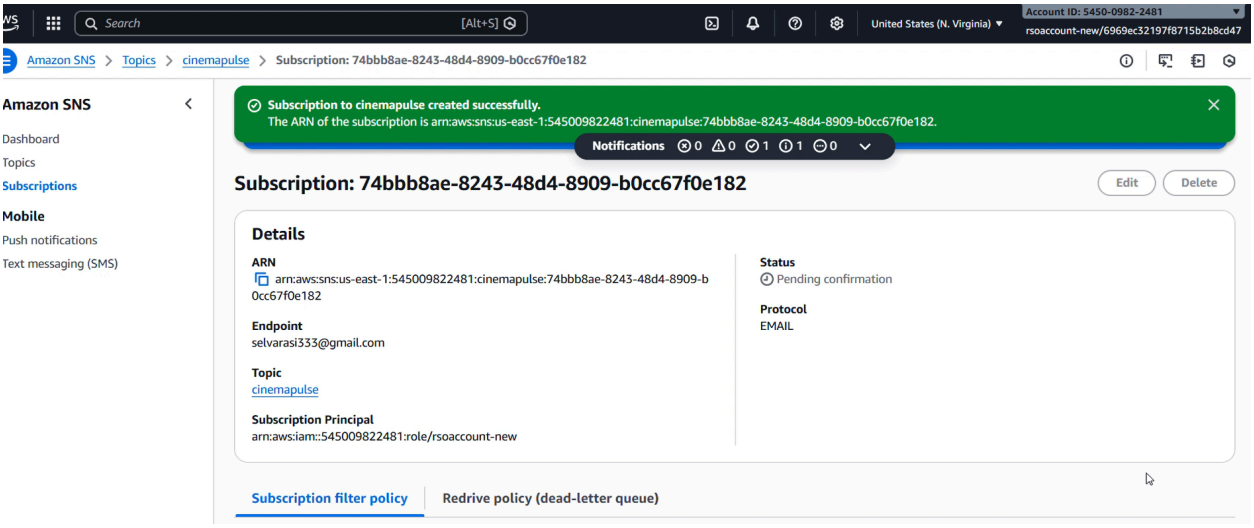
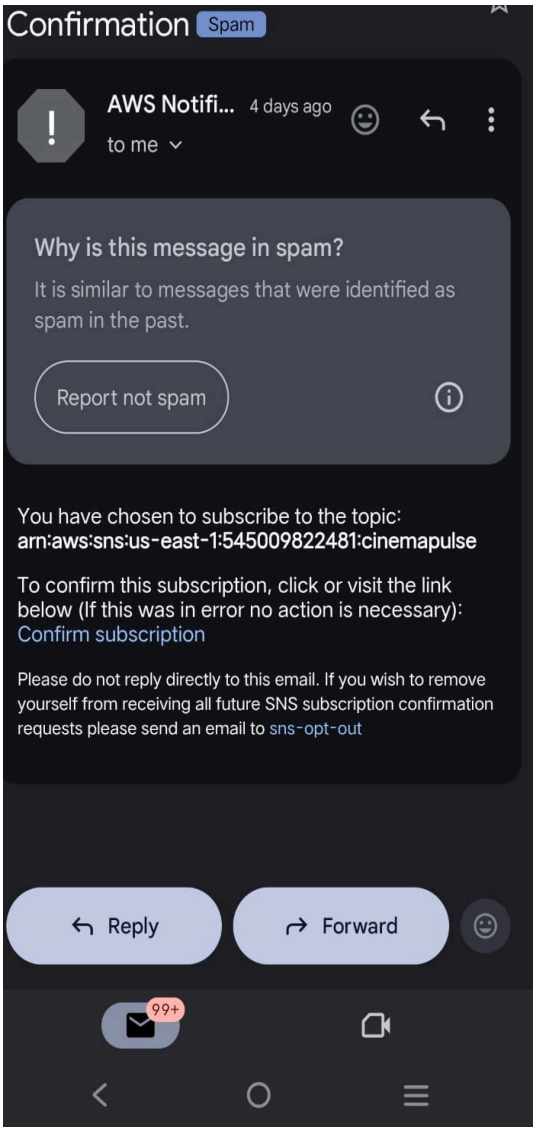


Figure: AWS SNS Email Notification for Real-Time Feedback Alerts
This screenshot shows an email notification received through AWS Simple Notification Service (SNS) as part of the CinemaPulse system. SNS is integrated to deliver real-time alerts for system events such as feedback submissions, analytics updates, or system notifications. This demonstrates cloud-based event-driven communication and real-time notification architecture within the AWS environment.


Confirmation
Spam
AWS Notifi...
4 days ago
to me
Why is this message in spam?
It is similar to messages that were identified as spam in the past.
Report not spam
You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:545009822481:cinemapulse
To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out
Reply
Forward
99+

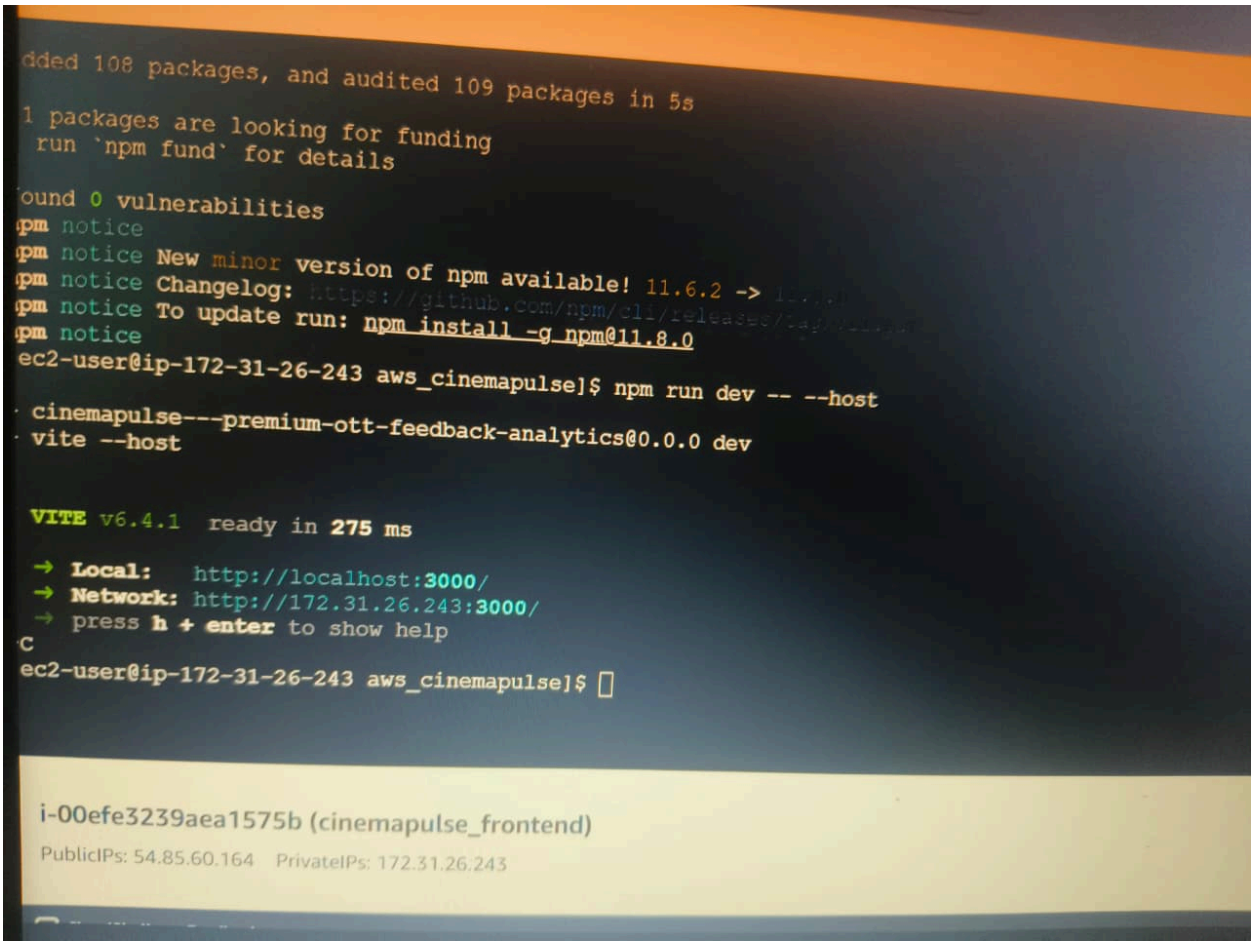
Frontend Public Access Documentation Content

Public Frontend Deployment

Description

The CinemaPulse frontend application is deployed as a publicly accessible web interface hosted on AWS infrastructure. The application is accessible using the EC2 public IP address, enabling users to interact with the system through a web browser. This confirms successful cloud deployment and public availability of the frontend application.

The frontend is fully integrated with the Flask backend APIs, enabling real-time feedback submission, movie analytics visualization, and role-based dashboard access.


dded 108 packages, and audited 109 packages in 5s
1 packages are looking for funding
run `npm fund` for details
ound 0 vulnerabilities
pm notice
pm notice New minor version of npm available! 11.6.2 ->
pm notice Changelog:
pm notice To update run: npm install -g npm@11.8.0
pm notice
ec2-user@ip-172-31-26-243 aws_cinemapulse]$ npm run dev -- --host
cinemapulse---premium-ott-feedback-analytics@0.0.0 dev
vite --host
VITE v6.4.1 ready in 275 ms
Local: http://localhost:3000/
Network: http://172.31.26.243:3000/
press h + enter to show help
c
ec2-user@ip-172-31-26-243 aws_cinemapulse]$
i-00efe3239aea1575b (cinemapulse_frontend)
PublicIPs: 54.85.60.164 PrivateIPs: 172.31.26.243

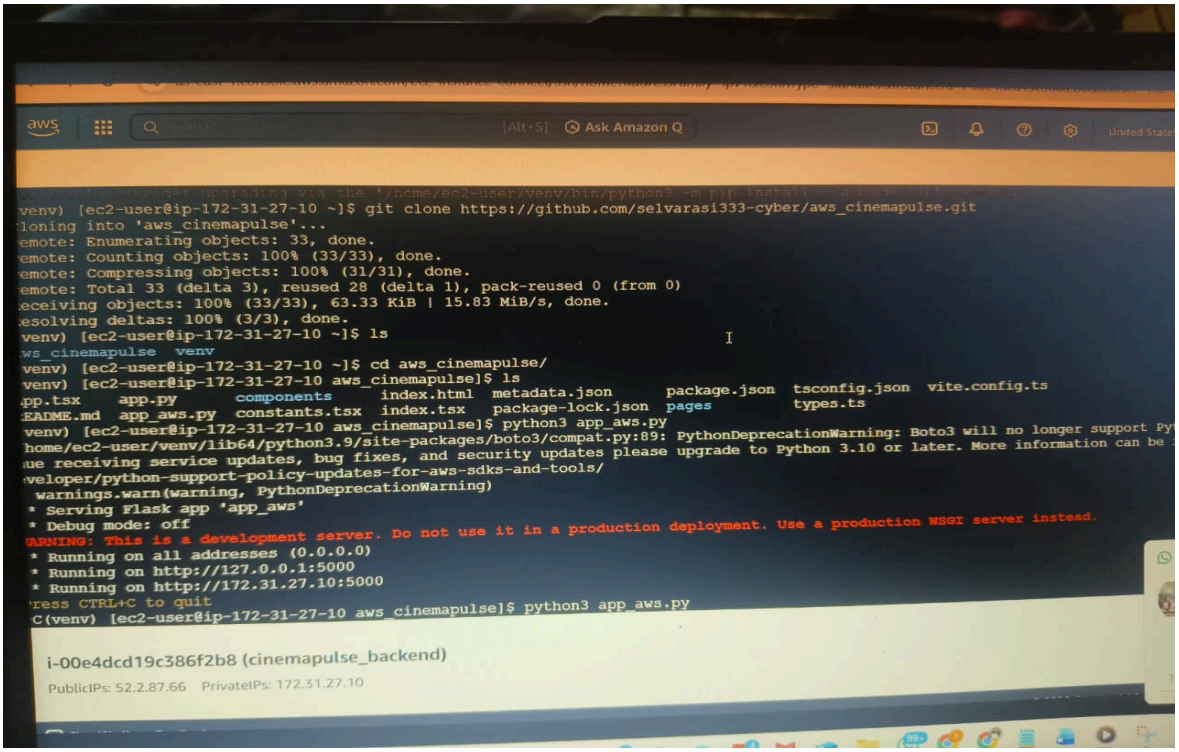
Backend Public Access

Public Backend Deployment

Description

The CinemaPulse backend application is deployed on an AWS EC2 instance and is publicly accessible through the EC2 public IP address. The Flask backend server exposes RESTful APIs for authentication, feedback management, movie data processing, and analytics services. This confirms successful cloud deployment and real-time backend availability.

The backend is securely integrated with AWS services including DynamoDB, SNS, and IAM, enabling scalable, secure, and cloud-native backend operations.



System Setup on AWS EC2

🔹 Step 1: System Update

sudo yum update -y

Description:
This command updates the EC2 Linux system packages to the latest versions. It ensures system security, stability, and readiness for deploying the CinemaPulse application.

🔹 Step 2: Install Required Software

sudo yum install -y python3 python3-pip git

Description:
Installs essential tools for CinemaPulse deployment:

- Python3 for running the Flask backend
- pip3 for installing Python dependencies
- Git for downloading the project source code

Project File Management

🔹 Step 3: Clone Project Repository

git clone <https://github.com/selvarasi333-cyber/aws_cinemapulse.git>

Description:

Downloads the CinemaPulse project source code from GitHub into the EC2 server environment.

🔹 Step 4: Enter Project Directory

cd CINI-PULSE-CAPSTONE-JAN

Description:
Moves into the main CinemaPulse project directory containing frontend, backend, and database files.

- **Step 5: View Project Files**

ls

Description:
Displays the project structure including Flask backend files, frontend folders, database, and configuration files.

Configuration & Preparation

- **Step 6: Edit Configuration Files**

```
nano app.py
```

or

```
nano .env.local
```

Description:
Used to configure Flask settings, AWS integrations, environment variables, and system parameters.

Dependency Installation

- **Step 7: Install Python Dependencies**

pip3 install -r requirements.txt

Description:
Installs all required Python libraries such as Flask, Flask-CORS, boto3, SQLite connectors, and other backend dependencies.

Application Execution

- **Step 8: Run Flask Backend Server**

python3 app.py

Description:

Starts the CinemaPulse Flask backend server on the EC2 instance.

Public Access Configuration

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

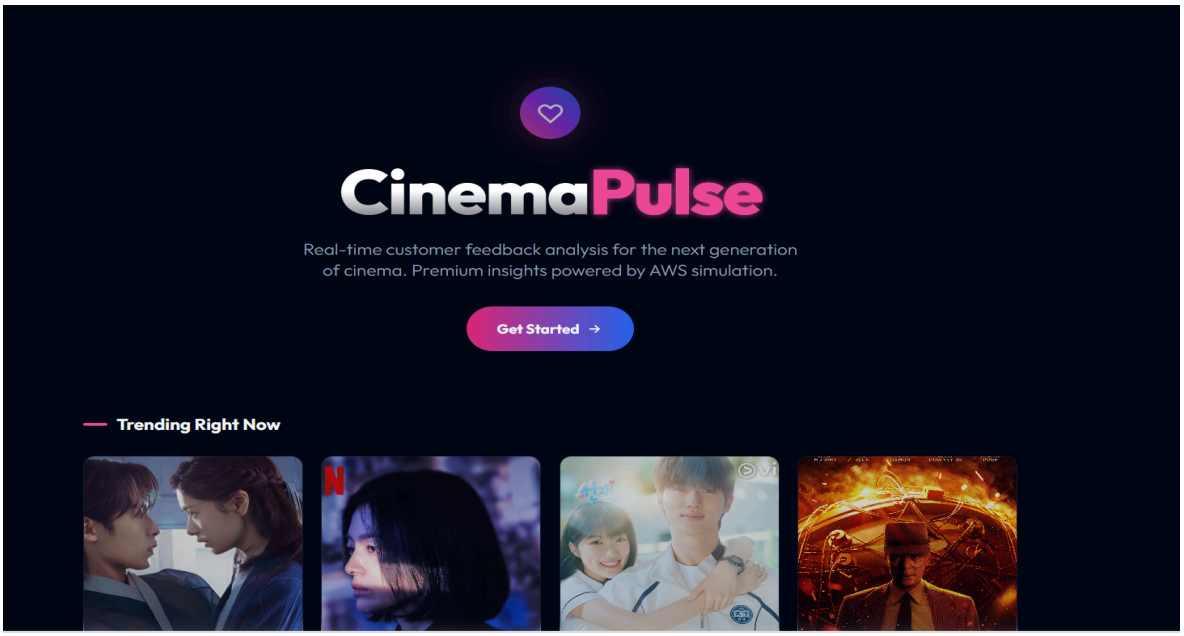
Description:

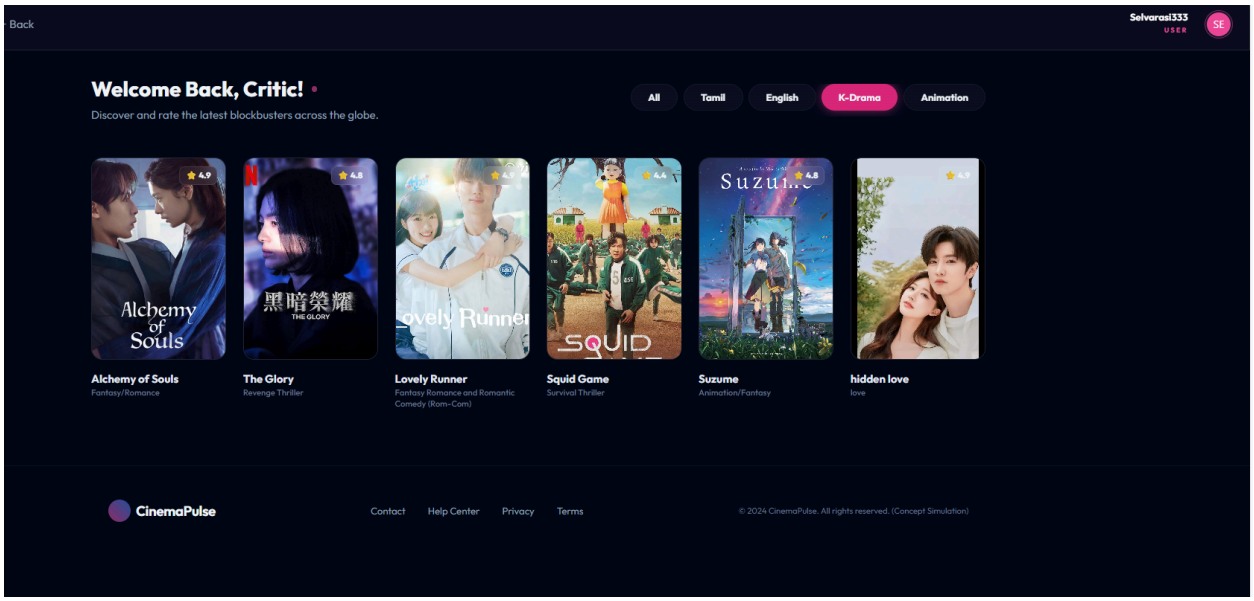
This configuration allows the Flask backend server to accept external requests using the EC2 public IP address, enabling frontend–backend communication over the internet.

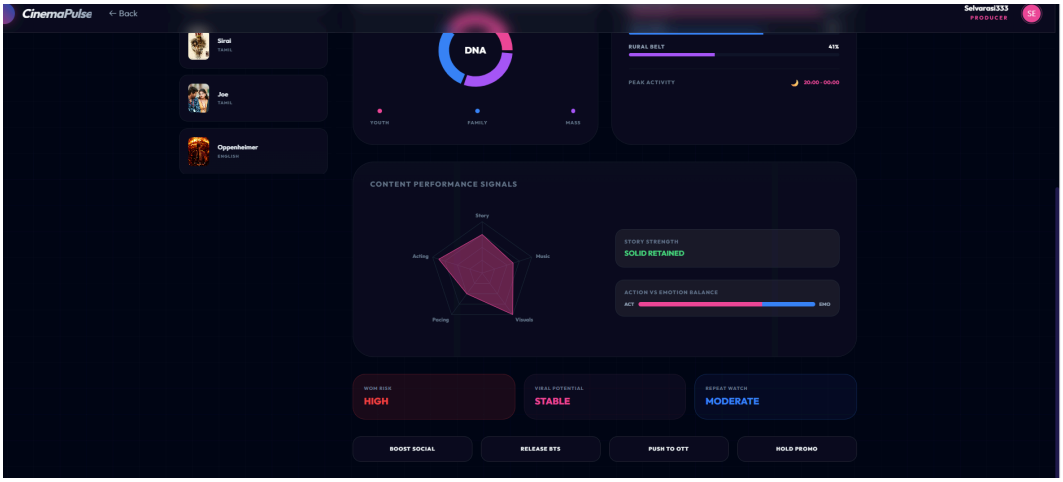
ACCESS THROUGH THE WEBSITE:

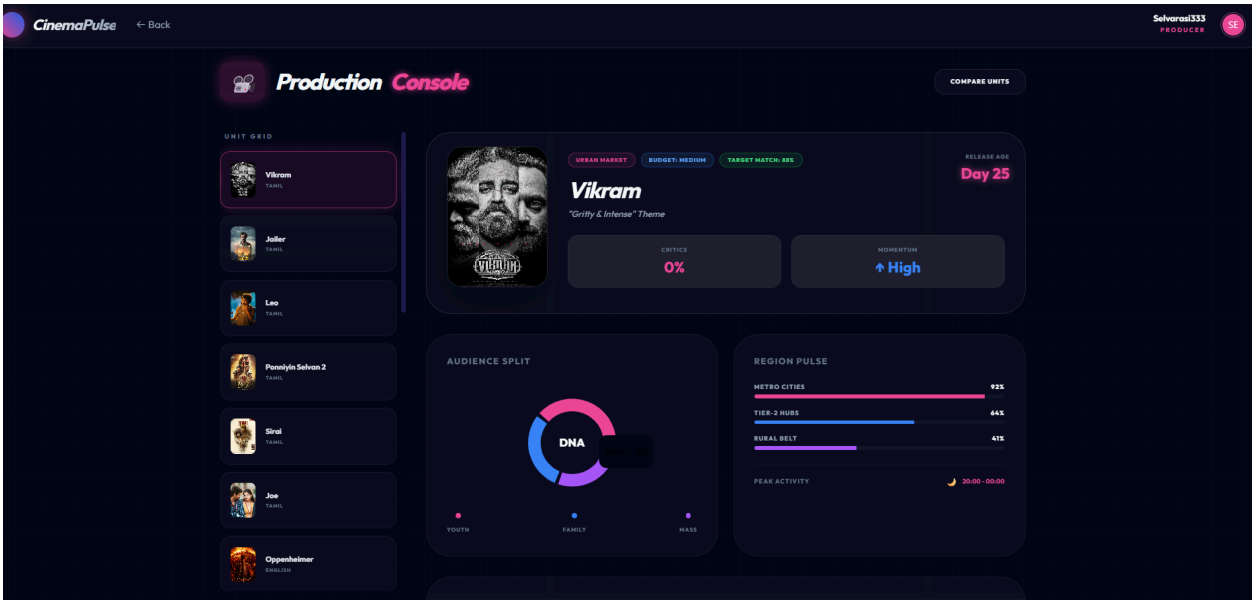
http://54.85.60.164:3000

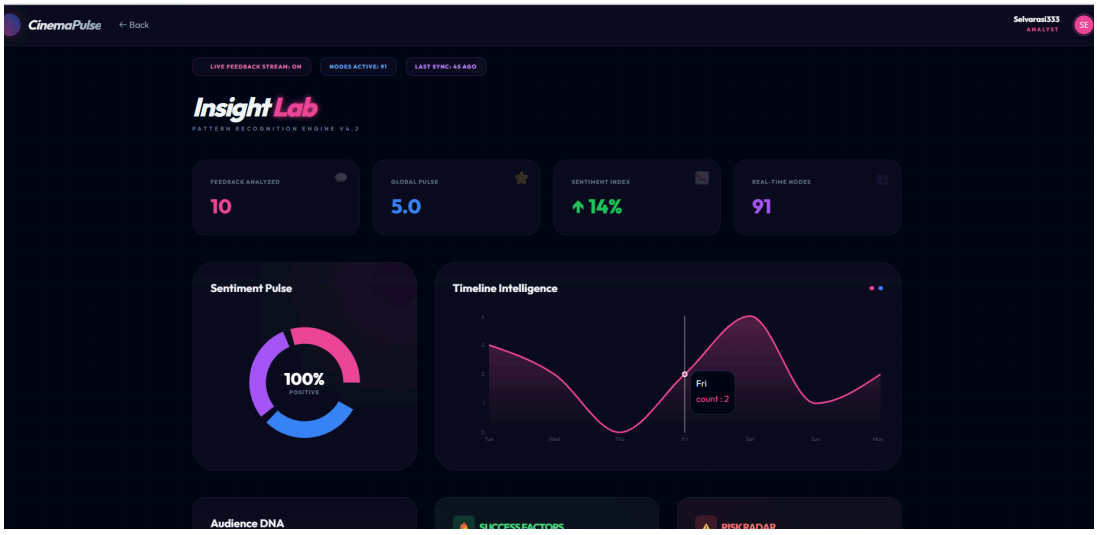
CinemaPulse is a cloud-based web application developed as an AWS Capstone Project to provide real-time customer feedback analysis for cinemas. The system features a user-friendly interface with a modern landing page, trending movie display, and interactive navigation. Users can submit feedback, which is processed through a Flask backend and analyzed using AWS services to extract sentiment insights. The application leverages AWS infrastructure for scalability, reliability, and performance, making it suitable for real-time data processing. Overall, CinemaPulse demonstrates the effective integration of frontend technologies, backend development, and AWS cloud services to deliver meaningful insights for the cinema industry.

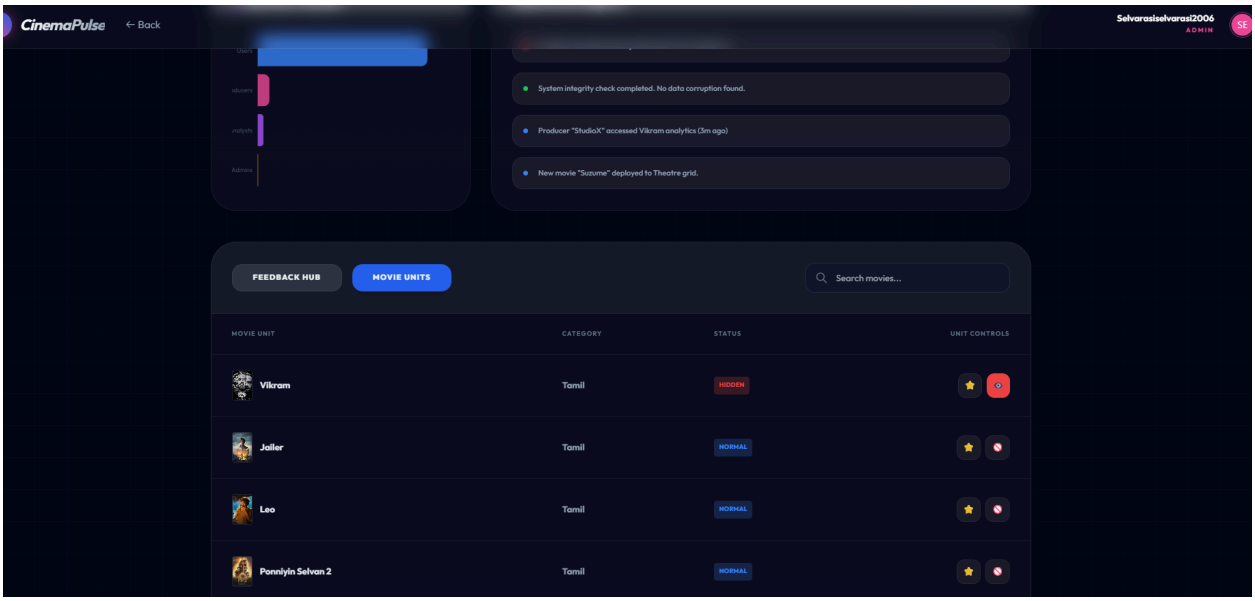


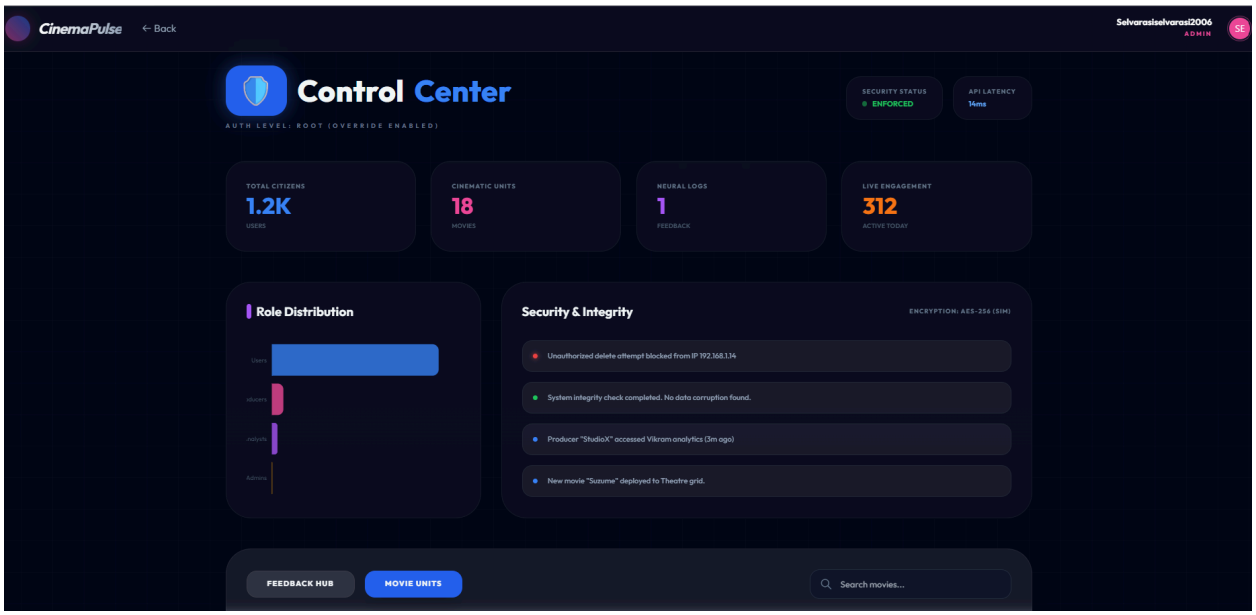


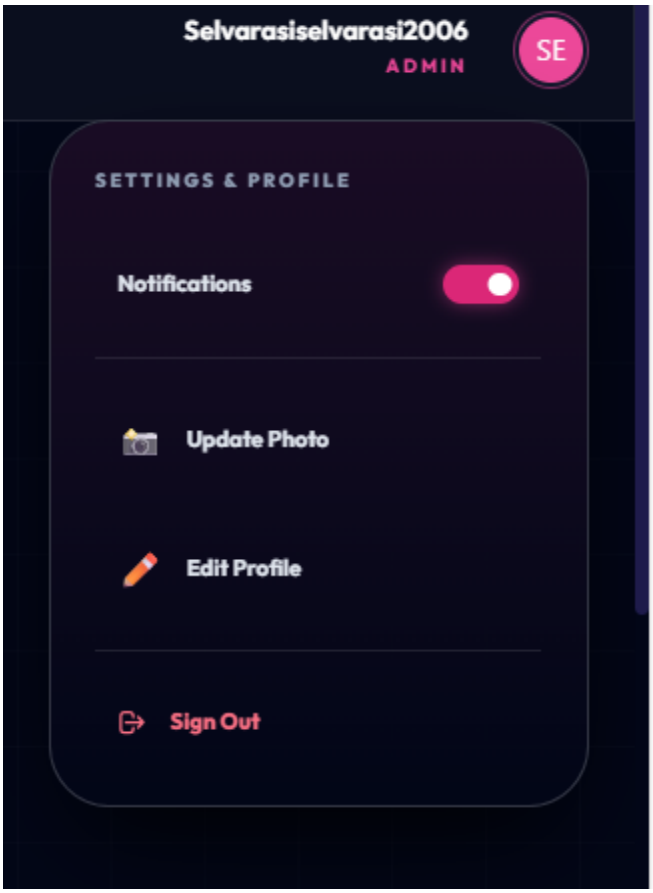

CinemaPulse
← Back
Selvarasi333
PRODUCER
Siral
TAMIL
Joe
TAMIL
Oppenheimer
ENGLISH
DNA
YOUTH
FAMILY
MASS
RURAL BELT
41%
PEAK ACTIVITY
20:00 - 00:00
CONTENT PERFORMANCE SIGNALS
Story
Music
Visuals
Pacing
Acting
STORY STRENGTH
SOLID RETAINED
ACTION VS EMOTION BALANCE
ACT
EMO
WOM RISK
HIGH
VIRAL POTENTIAL
STABLE
REPEAT WATCH
MODERATE
BOOST SOCIAL
RELEASE BTS
PUSH TO OTT
HOLD PROMO

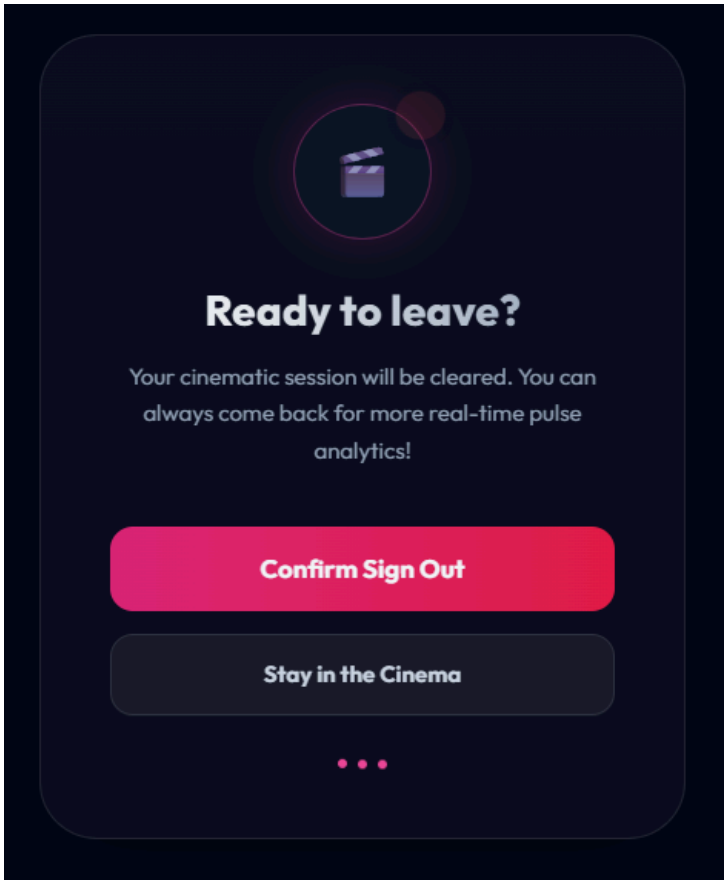

CinemaPulse
← Back
Selvarasi333
PRODUCER
Production Console
COMPARE UNITS
UNIT GRID
Vikram
TAMIL
Jailer
TAMIL
Leo
TAMIL
Ponniyin Selvan 2
TAMIL
Siral
TAMIL
Joe
TAMIL
Oppenheimer
ENGLISH
URBAN MARKET
BUDGET: MEDIUM
TARGET MATCH: 88%
RELEASE AGE
Day 25
Vikram
"Gritty & Intense" Theme
CRITICS
0%
MOMENTUM
High
AUDIENCE SPLIT
DNA
YOUTH
FAMILY
MASS
REGION PULSE
METRO CITIES
92%
TIER-2 HUBS
64%
RURAL BELT
41%
PEAK ACTIVITY
20:00 - 00:00


CinemaPulse
← Back
Selvarasi333
ANALYST
LIVE FEEDBACK STREAM: ON
NODES ACTIVE: 91
LAST SYNC: 45 AGO
Insight Lab
PATTERN RECOGNITION ENGINE V4.2
FEEDBACK ANALYZED
10
GLOBAL PULSE
5.0
SENTIMENT INDEX
14%
REAL-TIME NODES
91
Sentiment Pulse
100%
POSITIVE
Timeline Intelligence
Fri
count : 2
Audience DNA
SUCCESS FACTORS
RISK RADAR


CinemaPulse
← Back
Selvarasiselvarasi2006
ADMIN
System integrity check completed. No data corruption found.
Producer "StudioX" accessed Vikram analytics (3m ago)
New movie "Suzume" deployed to Theatre grid.
FEEDBACK HUB
MOVIE UNITS
Search movies...
MOVIE UNIT
CATEGORY
STATUS
UNIT CONTROLS
Vikram
Tamil
HIDDEN
Jailer
Tamil
NORMAL
Leo
Tamil
NORMAL
Ponniyin Selvan 2
Tamil
NORMAL


CinemaPulse
← Back
Selvarasiselvarasi2006
ADMIN
Control Center
AUTH LEVEL: ROOT (OVERRIDE ENABLED)
SECURITY STATUS
ENFORCED
API LATENCY
14ms
TOTAL CITIZENS
1.2K
USERS
CINEMATIC UNITS
18
MOVIES
NEURAL LOGS
1
FEEDBACK
LIVE ENGAGEMENT
312
ACTIVE TODAY
Role Distribution
Security & Integrity
ENCRYPTION: AES-256 (SIM)
Unauthorized delete attempt blocked from IP 192.168.1.14
System integrity check completed. No data corruption found.
Producer "StudioX" accessed Vikram analytics (3m ago)
New movie "Suzume" deployed to Theatre grid.
FEEDBACK HUB
MOVIE UNITS
Search movies...


Selvarasiselvarasi2006
ADMIN
SE
SETTINGS & PROFILE
Notifications
Update Photo
Edit Profile
Sign Out


Ready to leave?
Your cinematic session will be cleared. You can always come back for more real-time pulse analytics!
Confirm Sign Out
Stay in the Cinema

Conclusion

The project **CinemaPulse – Real-Time Customer Feedback Analysis (Concept Simulation on AWS)** was successfully designed, developed, and deployed as a complete cloud-integrated web application. The system effectively demonstrates how modern cloud infrastructure and full-stack technologies can be combined to build a scalable, secure, and intelligent feedback analysis platform for the entertainment industry.

The application integrates a **Flask-based backend**, **HTML/CSS/Vanilla JavaScript frontend**, and a **SQLite database** for local data management, along with **AWS cloud services** such as **EC2 for hosting**, **IAM for secure access control**, **DynamoDB for cloud-based data storage**, and **SNS for notification services**. This architecture ensures reliability, scalability, and secure role-based access across all user types including **Viewers, Producers, Analysts, and Administrators**.

Through rule-based analytics and structured data processing, CinemaPulse provides real-time insights such as sentiment analysis, audience trends, rating patterns, and performance indicators without relying on external AI APIs. The system successfully simulates a production-grade OTT analytics platform, offering decision-support features for producers and analysts while maintaining strong security and system stability.

The deployment on **AWS EC2** with public IP access confirms the project's real-world cloud readiness, demonstrating successful backend hosting, frontend integration, and end-to-end system connectivity. The implementation of **AWS IAM (Scenario 3: Secure Access to Movie Analytics)** ensures that sensitive data and analytics are protected through role-based permissions and controlled access.

Overall, CinemaPulse stands as a **robust, scalable, secure, and professionally structured cloud application**, fulfilling all project objectives and serving as a strong foundation for future enhancements such as real-time streaming analytics, advanced data visualization, AI-driven insights, and large-scale production deployment. This project successfully bridges theoretical cloud concepts with practical full-stack implementation, making it a complete and industry-relevant solution.