FAR EASTERN UNIVERSITY

Institute of Arts and Sciences

Department of Mathematics

Pusong Malaya Project: A Cloud-Native Movie Review Platform

FINAL PROJECT DOCUMENTATION

Prepared by:

Afundar, Audrie Lex L.
Aguila, Vera Frances A.
Cuerdo, Naomi Hannah A.
Dacanay, Jordan R.
Ducay, Harvey Lemuel O.
Gacasa, Ymanuel Josh R.
Masicat, Lindsy Rossel C.
Monfero, John Benedict A.
Rodillas, Christian Miguel T.

November 29, 2025

Contents

1 Project Introduction

1.1 Application Context and Goal

- **Application Name:** Pusong Malaya
- **Application Type:** Movies User Reviews & Ratings
- **Application Goal:** Pusong Malaya is a cloud-enabled initiative with a mobile application designed to empower everyone to freely express their genuine opinions and experiences through a unified review and rating system among Movies available within the Global Entertainment Market

  An application will be built using Flutter for the client interface and AWS Cloud Microservices (API Gateway, Lambda, DynamoDB) for the backend, mastering the system dynamics needed so that the application allows users to post, view, and evaluate reviews across various content or products.

2 Project Scope and Stakeholders

The success of the *Pusong Malaya* platform is assessed using the **Technology–Organization–Environment Lens** by Tornatzky & Fleischer (1990), which ensures the system meets the functional, policy, and infrastructure needs of all involved parties.

Stakeholders are essential to the project's viability, grouped by their relationship to the platform's core functions.

- **Task Stakeholders:**
  1. **Active Reviewers**: Generate content.
  2. **Review Browsers**: Consume content to make viewing decisions.
- **Content Oversight Stakeholders:**
  3. **Community Moderators**: Oversee quality and integrity.
  4. **Content Curators**: Ensure the catalogue remains accurate and civil.
- **Organizational Stakeholders:**
  5. **Pusong Malaya Core Team**: Dictates the product vision and feature roadmap.
  6. **Movie Organizations**: Assist with user recruitment and promotional outreach.
  7. **Data Privacy Officers**: Ensure adherence to institutional policies regarding user consent and data handling.
  8. **Potential Brand or Studio Partners**: Represent entities that may leverage anonymized market insights (future growth).
- **Technology/Infrastructure Stakeholders:**

9. **Development and DevOps Team**: Implements client and server code, managing deployment and security.
10. **Cloud and Platform Providers**: Provide essential infrastructure (e.g., AWS, app stores, analytics services).

These varied groups collectively define the complex operational environment of the project.

2.1 Technology Stack Overview

The project leverages a modern, serverless architecture:

- **Frontend:** Flutter (Mobile Client)
- **Backend:** AWS Cloud Microservices (Serverless: Lambda Functions + API Gateways)
- **Database:** Amazon DynamoDB (NoSQL)
- **Infrastructure/Hosting:** Amazon S3 and AWS IAM

2.2 Overall Architecture Diagram

Two figures below will be shown the simplified vs high-level view of the system components and their interaction.

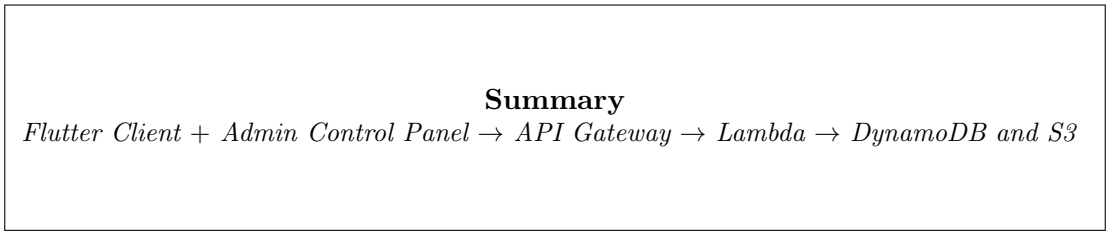


Figure 1: Oversimplified Architecture

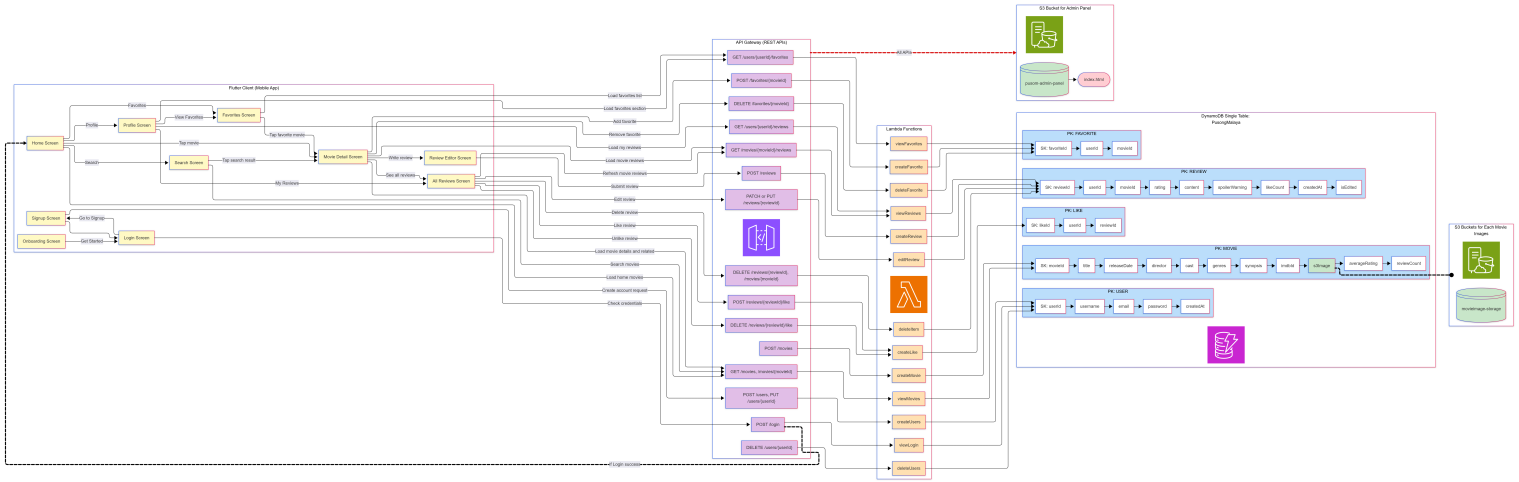


Figure 2: Pusong Malaya System Full Architecture

The Pusong Malaya's Full Architecture can be viewed with this link

`https://www.mermaidchart.com/d/65eacf10-60bc-4ad6-958b-1b43b9fe033d`

3 Mobile Development (UI and UX Development via Flutter)

3.1 Theme and Design Specification

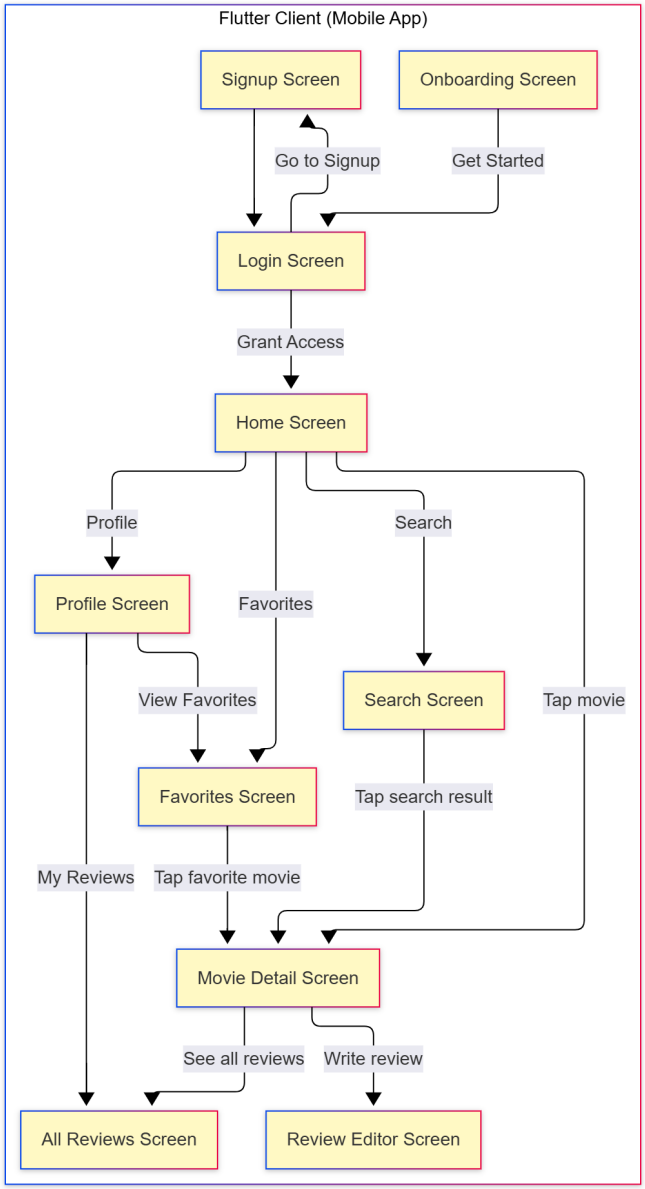


Figure 3: Frontend Architecture

3.1.1 Theme Proposal: Mulberry and Purple Minty

Theme Goal: vibrant, youthful, and eye-catching in evening feels, which is perfect for a "community reviews" vibe in dark mode already like a simulation of watching a movie in a closed theater vibes

3.1.2 Color Palette and Accessibility

A detailed breakdown of the primary, secondary, background, and surface colors for ensuring contrast standards.

Table 1: Single-Theme Color Distribution

Role	Color Value	Conceptual Use
Primary Base	`#061A2C`	Main screen background, top navigation, and major sections (The core dark color).
Accent / Highlight	`#FFEB3B`	Calls-to-action (CTAs), rating stars, active navigation links, and alerts.
Surface / Cards	`White (50% Opacity)`	Layered elements like review cards, modals, and tooltips (Semi-transparent over the dark base).
Text (Primary)	`#FFFFFF`	Main body copy and titles, ensuring maximum contrast against the dark background.
Text (Secondary)	`#B3B3B3`	Subtitles, helper text, and less emphasized information.

3.2 Typography Usage

The design uses a clean, modern sans-serif combination:

- **Main (Titles/Headlines): Khand** (Bold, geometric sans-serif for striking visual impact).
- **Supporting (Body/Labels): Poppins** (Highly readable and balanced sans-serif for all body text, forms, and labels).

3.3 Individual Component Breakdown

In this section, the document explains some important widgets or individual components created for each system pages.

3.3.1 User Authentication and Profile Management

The necessary components for these pages include input fields for email and password, utilizing robust form validation widgets to ensure data integrity before submission. A key component is the primary submission button (e.g., "Sign In" or "Register"), which triggers the authentication API call and navigates the user to the main application flow upon success.

Flutter Page Picture Slot
Signup Screen + Login Screen + Onboarding Screen

Figure 4: Login and Signup Pages

3.3.2 Movie Search and Discovery Interface

Flutter Page Picture Slot
Home Screen + Profile Screen + Favorite Screen + Search Screen + Movie Detail Screen

Figure 5: The Five Fundamental Pages of the Application System

The foundational components for discovery include a persistent search bar at the top of the Home and Search screens, enabling immediate text input for filtering. A responsive grid view is used to display movie posters, utilizing asynchronous image loading components for performance. On the Movie Detail screen, a floating action button allows users to quickly initiate a new review for that specific movie. Navigation is handled by a fixed bottom navigation bar, offering quick access to the Home, Profile, and Favorite screens. Finally, each movie card on the discovery page includes a small, interactive rating display component that summarizes the community's score.

3.3.3 Review Creation and Submission Flow

Flutter Page Picture Slot
All Review Screen + Review Editor Screen

Figure 6: The Review Page

The review editor screen must feature a large, multi-line text input field, allowing users sufficient space to write detailed critiques without excessive scrolling. A key component is the adjustable star rating widget, which uses touch gestures to capture the user's score from 1.0 to 10.0. A "Post Review" submission button, which is disabled until both the text and rating fields are populated, initiates the API call to the backend. The All Review Screen utilizes a list view builder to efficiently render individual review cards, each displaying the reviewer's name,

rating, and text snippet. Finally, a sorting/filtering dropdown is included at the top of the All Review Screen to allow users to organize reviews by score or date.

3.3.4 Review Viewing and Filtering

Flutter Page Code Simulation
Code about Review Viewing and Filtering

Figure 7: The Review Viewing and Filtering Functionality

Effective review viewing relies heavily on the ‘ListView.builder‘ widget to handle potentially thousands of review cards without memory issues. Each review card is a complex component featuring the reviewer's profile picture, username, the date of submission, and the full review text, which can optionally expand. The filtering component is typically implemented as a modal bottom sheet, allowing users to select multiple criteria such as minimum rating, language, or length. A central feature is the dynamic "Sort By" button, which changes its displayed label to reflect the current sort order (e.g., "Sort By: Highest Rated"). Furthermore, a local state management system ensures that the displayed list updates immediately when a filter or sort option is applied, providing a smooth user experience. This whole system relies on calling the GET reviews API endpoint with specific query parameters.

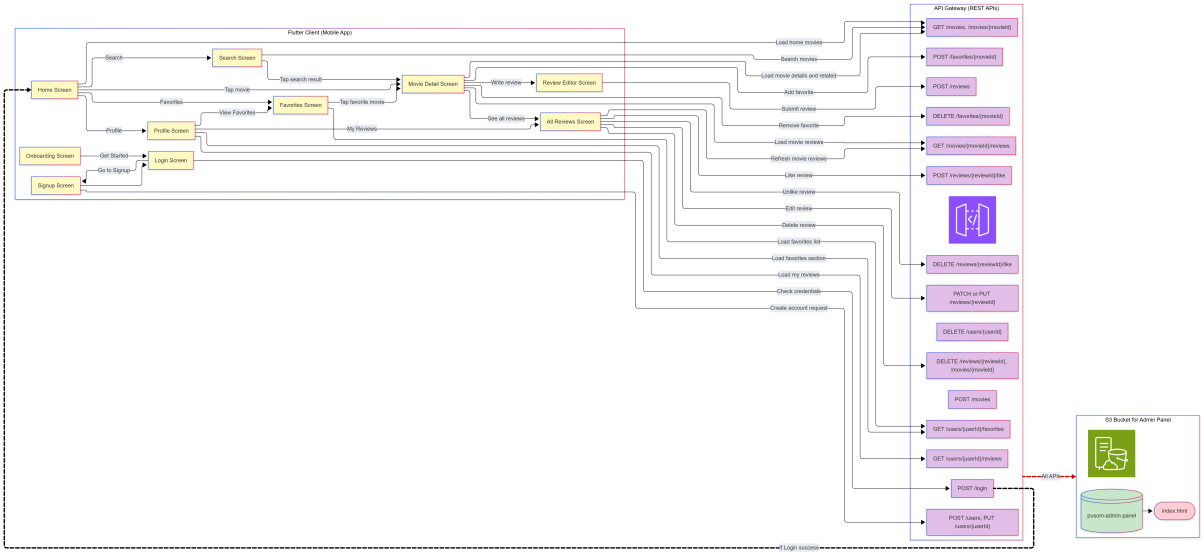
3.3.5 Like and Favorite Functionality

Flutter Page Code Simulation
Code about Like and Favorite Functionality

Figure 8: Other Important Functionality

The "Like" functionality is represented by a small, interactive heart or thumbs-up icon placed on every review card, utilizing an animated state change to provide visual feedback upon tap. Tapping this icon triggers a non-blocking API call (POST to the LIKE endpoint) and locally increments the displayed like count for a seamless user experience. The "Favorite" feature, used for saving movies, is typically a distinct bookmark icon found on the Movie Detail screen. This Favorite button toggles the movie's status and updates the user's list, accessible via the dedicated Favorite Screen. Both components use asynchronous state management to reflect the true status (liked or favorited) retrieved from the backend upon initial loading.

4 API Gateway



4.1 System Architecture Flow

The backend system operates using a Serverless pattern: API Gateway routes requests to an AWS Lambda function, which executes the business logic and interacts with the Database.

4.1.1 Base URL

All API requests should be made to:

`https://ccfh9odail.execute-api.ap-southeast-2.amazonaws.com/dev`

4.2 API Request Outcomes and HTTP Status Codes

After defining the context and goals of the backend, it is important to clarify how the API communicates success and failure to the frontend. The movie review app will follow standard HTTP status code conventions so that the Flutter client can reliably interpret responses.

4.2.1 Summary of Common Status Codes

4.3 Movies Endpoints

Code	Name	Typical use in this project
200	OK	Successful `GET` requests (home feed, movie detail, profile, search, lists).
201	Created	New resource created (user signup, new review, new favourite).
400	Bad Request	Invalid or missing fields in review submission, signup, or other forms.
401	Unauthorized	User not logged in or token invalid when calling protected endpoints.
403	Forbidden	User is logged in but not allowed to modify a resource (e.g. editing others' reviews).
404	Not Found	Movie, review, or user does not exist or has been deleted.
500	Internal Server Error	Unexpected backend error (exceptions, timeouts, configuration issues).

Table 2: Common HTTP status codes for the movie review API

Table 3: Movies API Endpoints

Method	Endpoint	Description
`POST`	`/movies`	Creates a new Movie entry. Requires authentication and administrative privileges (`createMovie`).
`GET`	`/movies`	Retrieves a list of all Movies. Supports optional query parameters for filtering/pagination (`viewMovies`).
`GET`	`/movies/{movieId}`	Retrieves a single Movie by its ID (`viewMovies`).
`PUT`	`/movies/{movieId}`	Updates an existing Movie. Accepts partial Movie object payload. Requires administrative privileges.
`DELETE`	`/movies/{movieId}`	Deletes a Movie entry (`deleteItem`). Requires administrative privileges.

4.4 Users and Authentication Endpoints

Table 4: Users and Authentication API Endpoints

Method	Endpoint	Description
`POST`	`/users`	Creates a new User account (`createUsers`). This is the sign-up function.
`POST`	`/login`	Authenticates a user with credentials and issues an access token (`viewLogin`).
`DELETE`	`/users/{userId}`	Deletes a User account (`deleteUsers`). Requires user or administrative authorization.
`PUT`	`/users/{userId}`	Updates User profile information (e.g., password, display name).

4.5 Reviews Endpoints

Table 5: Reviews API Endpoints

Method	Endpoint	Description
POST	/reviews	Posts a new Review for a Movie (createReview). Requires movieId and userId in the payload.
GET	/movies/{movieId}/reviews	Retrieves all Reviews for a specific Movie (viewReviews).
DELETE	/reviews/{reviewId}	Deletes a Review by its ID. Must be the owner or an administrator.

4.6 Favorites Endpoints

Table 6: User Favorite Movies Endpoints

Method	Endpoint	Description
POST	/favorites/{movieId}	Adds a Movie to the current authenticated user's favorites list (createFavorite).
GET	/users/{userId}/favorites	Retrieves the list of favorited movies for a specified user (viewFavorites).
DELETE	/favorites/{movieId}	Removes a Movie from the user's favorites list (deleteFavorite).

4.7 Likes Endpoints

Table 7: Review Likes Endpoints

Method	Endpoint	Description
POST	/reviews/{reviewId}/like	Adds a "like" to a specific review (createLike).
DELETE	/reviews/{reviewId}/like	Removes a "like" from a specific review (deleteLike).

4.8 Movie Model Structure

Movie objects are the core data structure.

```
{
  "movieId": "MOVIE#tt1375666",
```

```
  "title": "Inception",
  "releaseYear": "2010",
  "directors": ["Christopher Nolan"],
  "actors": ["Leonardo DiCaprio", "Joseph Gordon-Levitt"],
  "genres": ["Sci-Fi", "Thriller"],
  "synopsis": "A mind-bending thriller...",
  "averageRating": "8.8",
  "reviewCount": 1200
}
```

4.9 Review Model Structure

```
{
  "reviewId": "REVIEW#1700720400",
  "movieId": "MOVIE#tt1375666",
  "userId": "USER#johndoe",
  "rating": "9.5",
  "text": "Absolutely breathtaking cinematography and plot.",
  "timestamp": "2025-11-23T10:00:00Z"
}
```

4.10 Code Examples

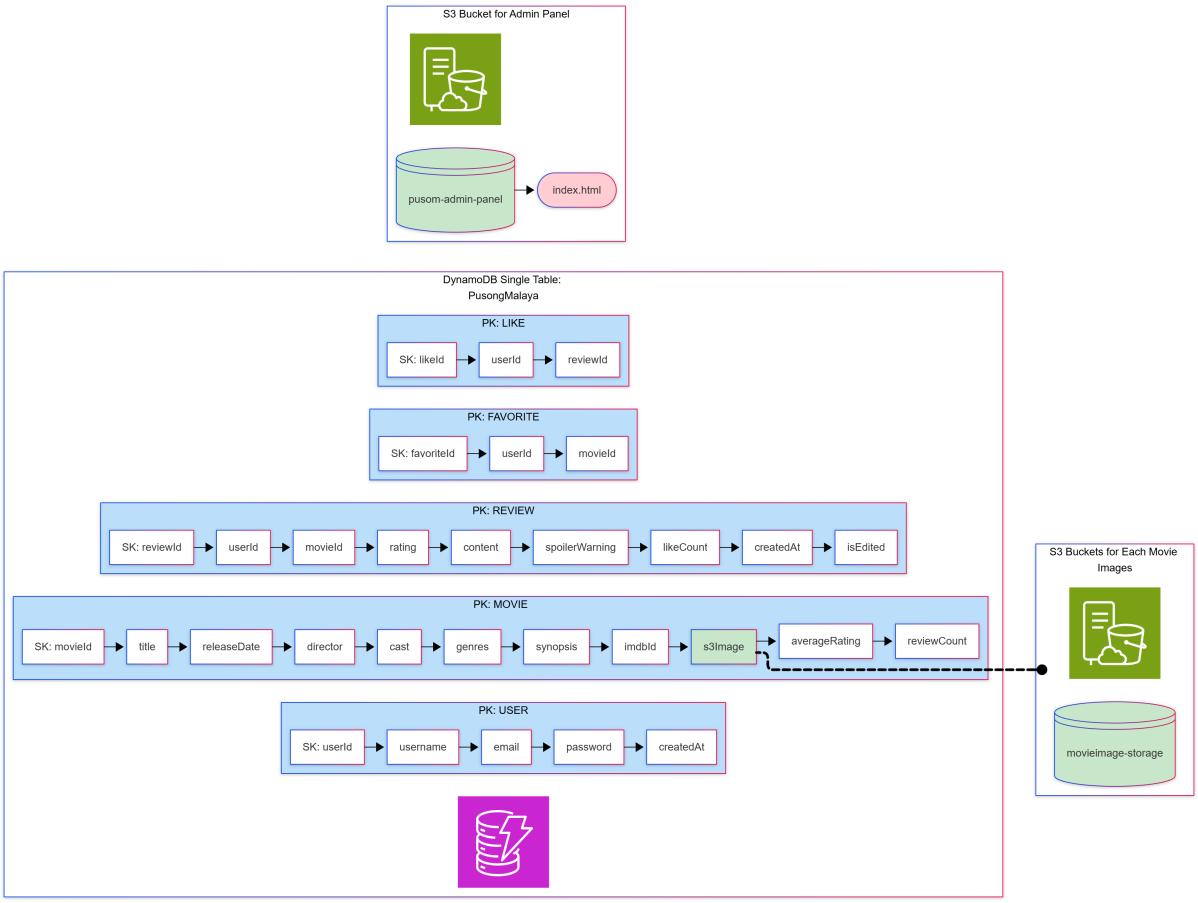
4.10.1 Node.js Client (Simplified)

```
const api = new MovieReviewApi(BASE_URL);

// Create a movie
const movie = await api.createMovie({
  title: "Inception",
  releaseYear: "2010",
  directors: ["Christopher Nolan"],
  actors: ["Leonardo DiCaprio"],
  genres: ["Sci-Fi"],
  synopsis: "A mind-bending thriller",
  averageRating: "8.8",
  reviewCount: 0
});
console.log(movie);

// Get all movies
const movies = await api.getMovies();
console.log(`Total movies: ${movies.count}`);
```

5 DynamoDB and S3 Buckets



The DynamoDB Single Table (PusongMalaya) stores all data entities of the application (such as `USER`, `MOVIE`, `REVIEW`, `FAVORITE`, `LIKE`). Each entity type (`USER`, `MOVIE`, `REVIEW`, `FAVORITE`, `LIKE`) has its own structured attributes for fast queries and efficient access patterns. The S3 buckets store static assets: one bucket holds admin panel files, and another holds movie images, linked directly from DynamoDB through the s3Image field.

6 IAM Identity Center

6.1 Objectives

The objective of this section is to define a role-based access model for all nine members of Puso Malaya using AWS IAM Identity Center. The policy aims to grant sufficient permissions while strictly adhering to the principle of least privilege.

- Sufficient permissions to perform with accordance to their assigned tasks.
- No excessive or unnecessary privileges beyond their role.
- A clearly named permission set that can be referenced in documentation and the AWS console.

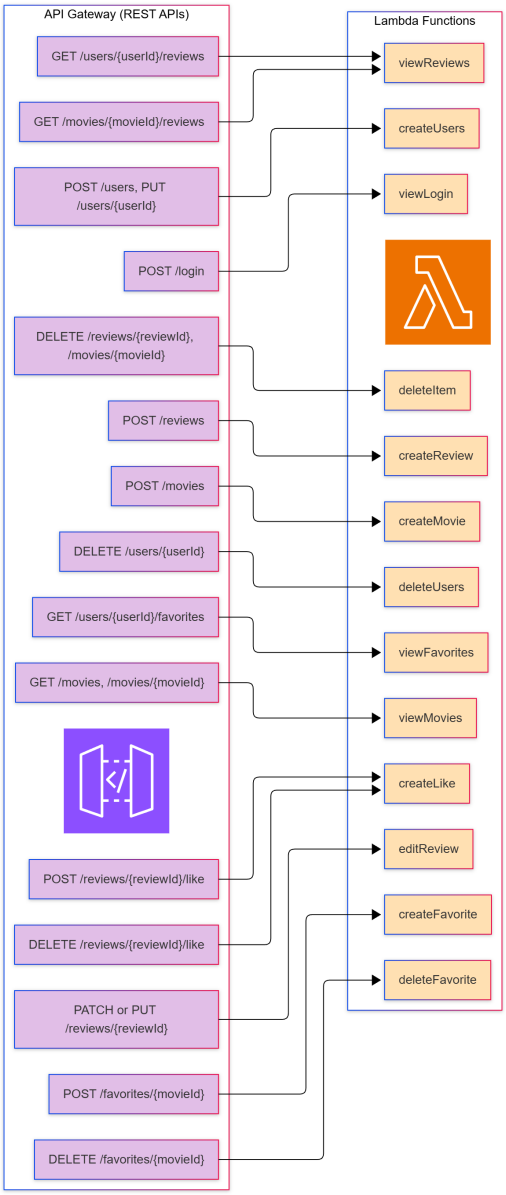
6.2 Team Members and Permission Sets

Table 8 below summarizes the final IAM Identity Center mapping based on the defined project needs.

Table 8: Puso Malaya IAM Identity Center Role Mapping

Name(s)	**Project Role**	**Permission Set Name**	**Core Service Access (S3, DB, Lambda, API Gateway, IAM)**
Monfero	Project Manager / Architect	`ProjectManager-Acess`	**Access Level:** Read-only to ALL services (IAM, S3, DynamoDB, Lambda, API Gateway).
Masicat, Vera	Mobile / Frontend Developer	`MobileDev-Access`	**Full Control:** API Gateway (for testing and deployment). **Read-only:** S3, DynamoDB (DB), Lambda.
Miguel, Naomi, Gacasa	Infrastructure Engineer	`InfastructureEng-Access`	**Full Control:** IAM and S3 (for security/hosting management). **Read-only:** DynamoDB (DB), Lambda, API Gateway.
Harvey, Lex, Dacanay	Backend Developer	`BackendDev-Access`	**Full Control:** S3, DynamoDB (DB), Lambda, and API Gateway (for implementation and deployment).

7 Lambda and IAM Resources: Authentication Roles for Lambda execution



7.1 Authentication Flow (Manual Made IAM)

The mobile client grants temporary credentials to interact with the API Gateway by declaring a specific policy grants for each lamdba function created for the Pusong Malaya.

Table 9: Deployed Lambda Function, Runtime Role, and API Mapping

Lambda Function Name	**HTTP Method (Inferred)**	**IAM Execution Role (Resource)**	**Primary Access Granted / DynamoDB Tables**
`viewFavorites`	`GET`	`viewFavorites-role-hplj2wmn`	**DB Read**: `Users` table to retrieve a user's list of favorited movies.
`viewReviews`	`GET`	`viewReviews-role-2ckj85tj`	**DB Read**: `Reviews` table. Retrieves reviews, typically filtered by Movie ID.
`viewLogin`	`GET`	`viewLogin-role-jgacd6kh`	**DB Read**: `Users` table. Likely performs credential check and session verification.
`viewMovies`	`GET`	`viewMovies-role-uhtycj0l`	**DB Read**: `Movies` table (for listing all or filtering).
`createUsers`	`POST`	`createUsers-role-gpsl9s6v`	**DB Write**: `Users` table. Handles new user registration/sign-up.
`deleteLike`	`DELETE`	`deleteLike-role-wpqgvrle`	**DB Write/Update**: `Reviews` table. Decrements the like count for a specific review.
`createLike`	`POST`	`createLike-role-getkx1qo`	**DB Write/Update**: `Reviews` table. Increments the like count for a specific review.

Table 9: Deployed Lambda Function, Runtime Role, and API Mapping (Continued)

Lambda Function Name	**HTTP Method (Inferred)**	**IAM Execution Role (Resource)**	**Primary Access Granted / DynamoDB Tables**
`deleteUsers`	`DELETE`	`deleteUsers-role-iry09nf6`	**DB Delete**: `Users` table. Handles account deletion.
`createMovie`	`POST`	`createMovie-role-q98ra62b`	**DB Write**: `Movies` table. Creation of new movie records (Admin/Curator function).
`createReview`	`POST`	`createReview-role-ufytw3uv`	**DB Write**: `Reviews` table. Also requires **DB Update** on `Movies` for aggregate rating.
`deleteFavorite`	`DELETE`	`deleteFavorite-role-w7zmqmqa`	**DB Write/Update**: `Users` table. Removes a movie from the user's favorite list.
`createFavorite`	`POST`	`createFavorite-role-vv1wq0d4d`	**DB Write/Update**: `Users` table. Adds a movie to the user's favorite list.
`deleteItem`	`DELETE`	`deleteItem-role-q57udjth`	**DB Delete**: General purpose function, likely targets `Movies` or `Reviews` based on context.

8 Cloud Infrastructure and Admin Hosting

8.0.1 Static Assets Hosting (S3)

Unlike Flutter, the Admin Panel will be deployed and accessble via S3 assets (`pusom-admin-panel` while in DynamoDB, all movie images shall be saved in another S3 bucket (`movieimage-storage`) for low-latency delivery.

8.1 Admin Control Panel Development (Static HTML)

8.1.1 Interface Design and Purpose

- A data management view that focuses to have CRUD operations, the assets itself shall have strong similarity of UI and UX vibes made within the `Pusong Malaya` Flutter Application
- Focused on content moderation (deleting abusive reviews).
- User management (tracking accounts).
- More convenience way on how to add new movies within the DynamoDB management (instead of performing test entry with lambda or injecting `JSON` files through `AWS Cloudshell`).

8.1.2 Administrative Page Mockups

This section provides mockups for the administrative interface, demonstrating key dashboard views and content management screens.

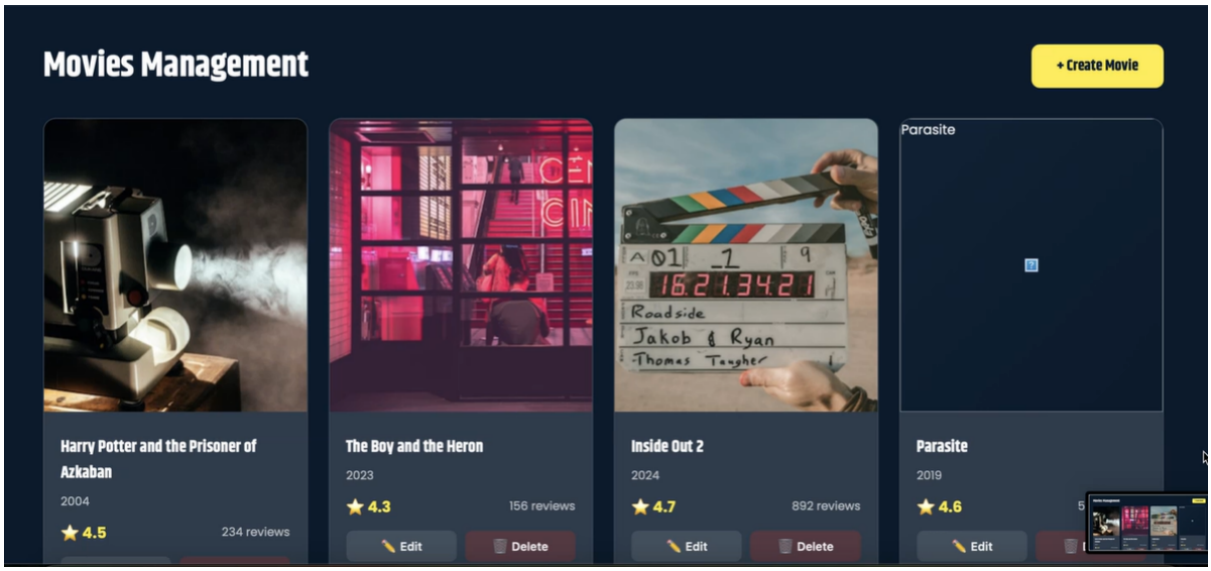


Figure 9: Movie Management

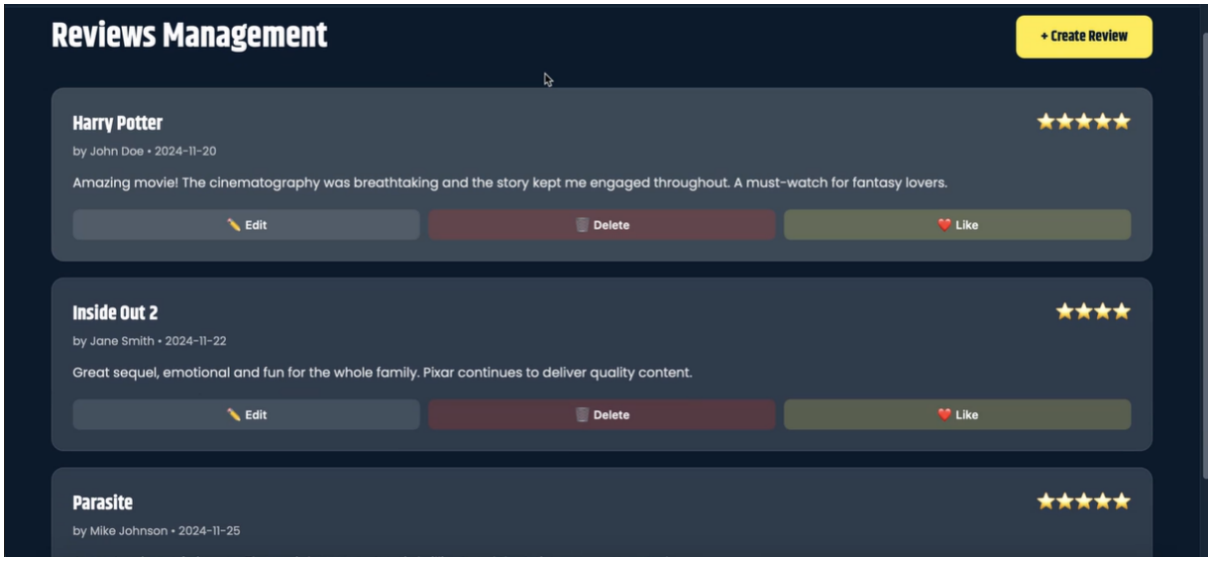


Figure 10: Reviews Management

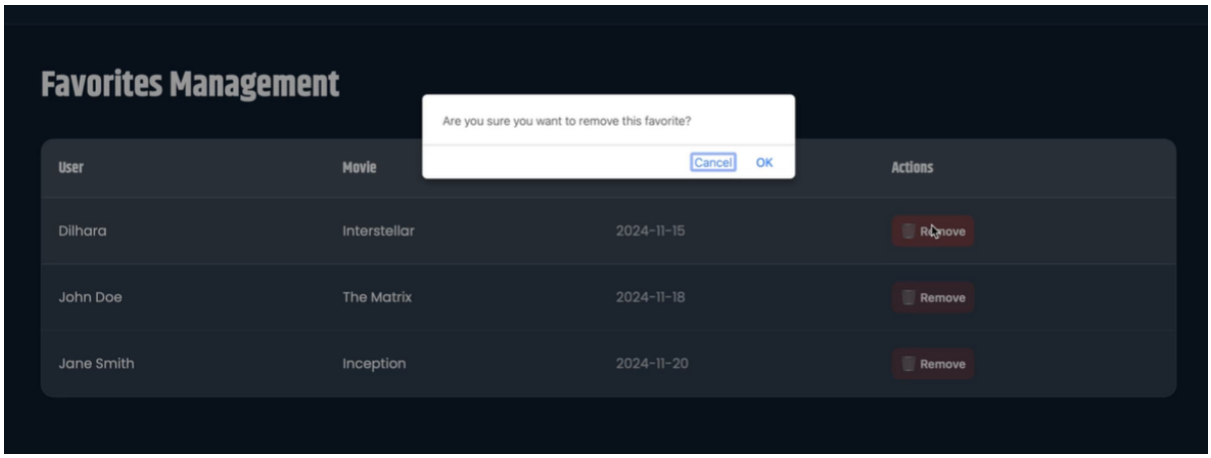


Figure 11: Favorites Management

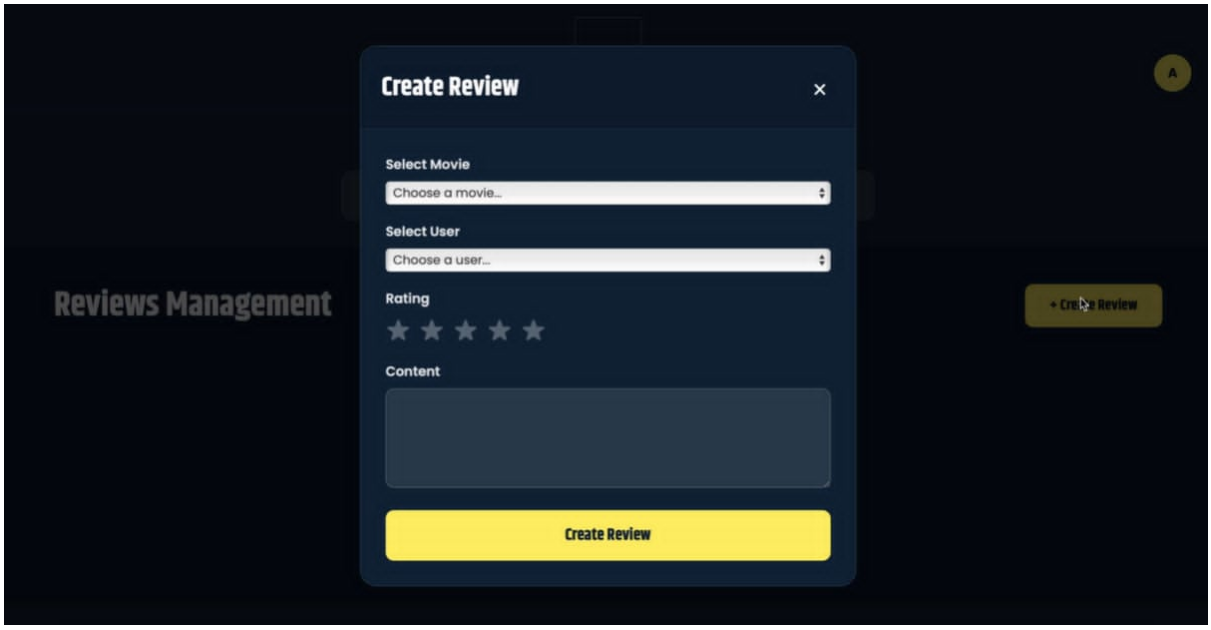


Figure 12: Adding Reviews

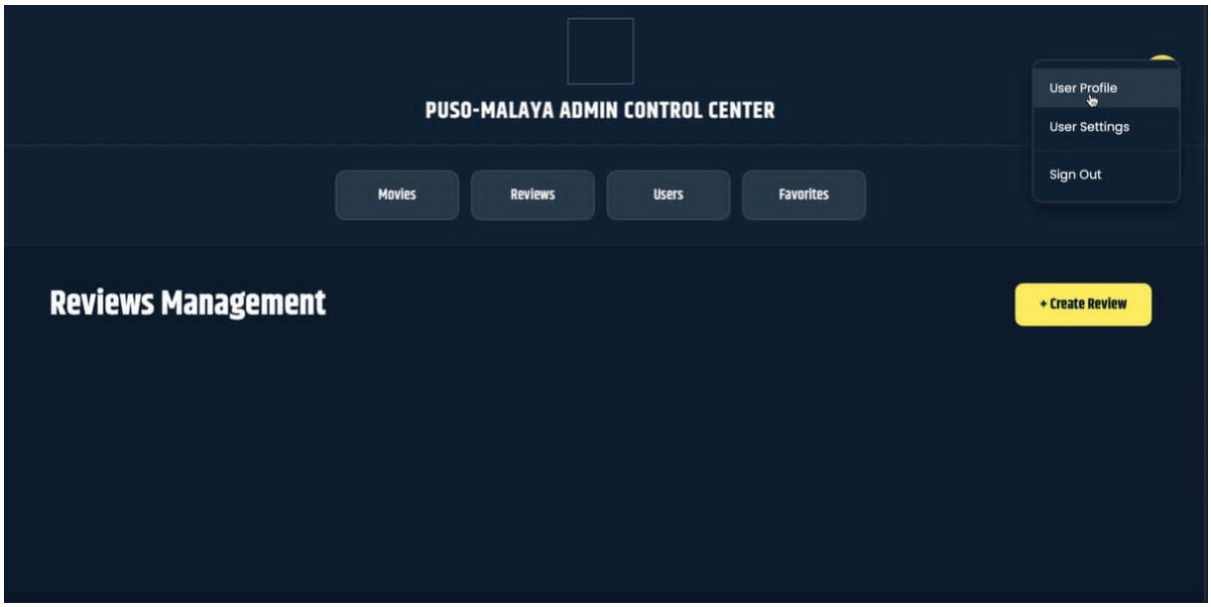


Figure 13: Admin's Homescreen

9 Appendices

9.1 Team Distribution and Task Description

Role Proposal	Proponent	Comprehensive Responsibility
Product Manager	Monfero, John Benedict A.	1. Track progress on each department and its members' learning curve 2. Oversees the architecture development 3. Ensure design consistency 4. Suggests online discussion from time to time 5. Handles team grievances 6. Prepare final documentation
AWS Backend Engineer	Afundar, Audrie Lex L. Ducay, Harvey Lemuel O. Dacanay, Jordan R.	1. Build Lambda functions with Python 2. Configure API Gateway routes 3. Database Schema Proposal 4. Creation of DynamoDB
Cloud Infrastructure and Admin Deployment	Cuerdo, Naomi Hannah A. Rodillas, Christian Miguel T. Gacasa, Ymanuel Josh R.	1. Handling IAM Roles 2. Deployment of Static Websites 3. S3 Buckets 4. Testing the endpoints and vulnerability of the Database
Mobile Developers	Aguila, Vera Frances A. Masicat, Lindsy Rossel C.	1. Building with the Flutter app 2. Design and Propose the UI and UX 3. Integrate APIs through HTTP requests 4. APK outputs (client + admin)

9.2 Github Reference and Codes

https://github.com/Monferium/Pusong-Malaya-Project