MOVIEMAGIC - SMART MOVIE TICKET BOOKING SYSTEM

**Submitted by:** Amasa Pooja Sree

**Department:** AI & Data Science

**Name of the Institution:** Annamacharya Institute of Technology and Sciences, Tirupati (AITS)

**Email:** [apsreddy10@gmail.com](mailto:apsreddy10@gmail.com)

**Running on http://127.0.0.1:5000**

**ABSTRACT**

In an era of digital transformation, the traditional method of purchasing movie tickets is increasingly being replaced by smarter, faster, and more reliable online systems. **Movie Magic** is a web-based smart movie ticket booking application designed to streamline the ticket reservation process. Built using Python Flask for backend logic and HTML/CSS for the frontend, this system enables users to register, log in, search for available movies, and securely book tickets in real-time.

The platform enhances user experience with a clean interface, session-based access control, and a password recovery feature utilizing OTP verification. Though data is handled in-memory for demonstration, the system architecture is designed to be cloud-deployable using **AWS EC2**, with future integration options for **DynamoDB** and **SNS** for real-time email notifications. An admin dashboard allows monitoring of all bookings, offering a complete and functional overview of the system’s activity.

This project provides a robust foundation for building a scalable and user-friendly online ticketing system that addresses the inefficiencies of manual booking while offering a modern, cloud-compatible alternative for cinemas and entertainment venues.

# TABLE OF CONTENTS

1. Introduction

2. Problem Statement

3. Objectives

4. System Architecture

5. Modules/Features

6. Technologies Used

7. Snapshots

8. Code Overview

9. Challenges Faced

10. Conclusion

11. References

**🔹 1. Introduction**

**Movie Magic** is a smart, web-based movie ticket booking system built using the Flask framework. It simplifies the traditional cinema ticketing process by enabling users to register, log in, browse available movies, and book tickets online. The system ensures secure session handling and includes a password recovery mechanism using OTP-style verification. This user-friendly platform enhances accessibility and digitization in cinema ticketing.

**🔹 2. Problem Statement**

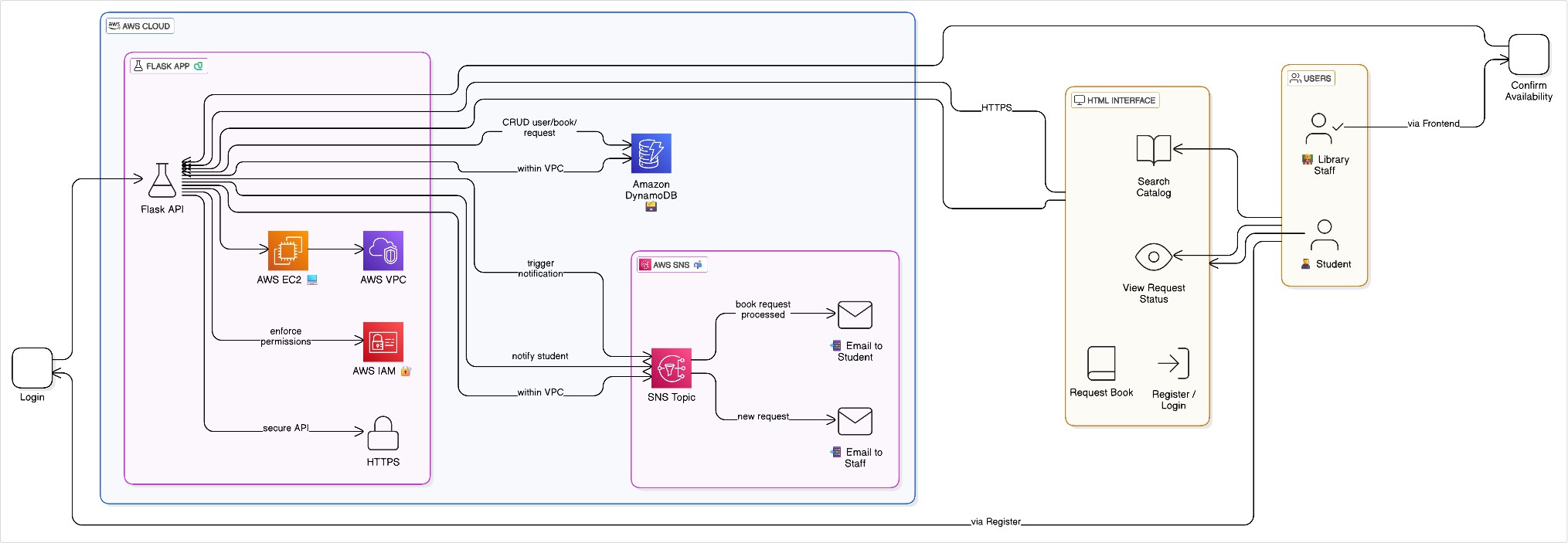
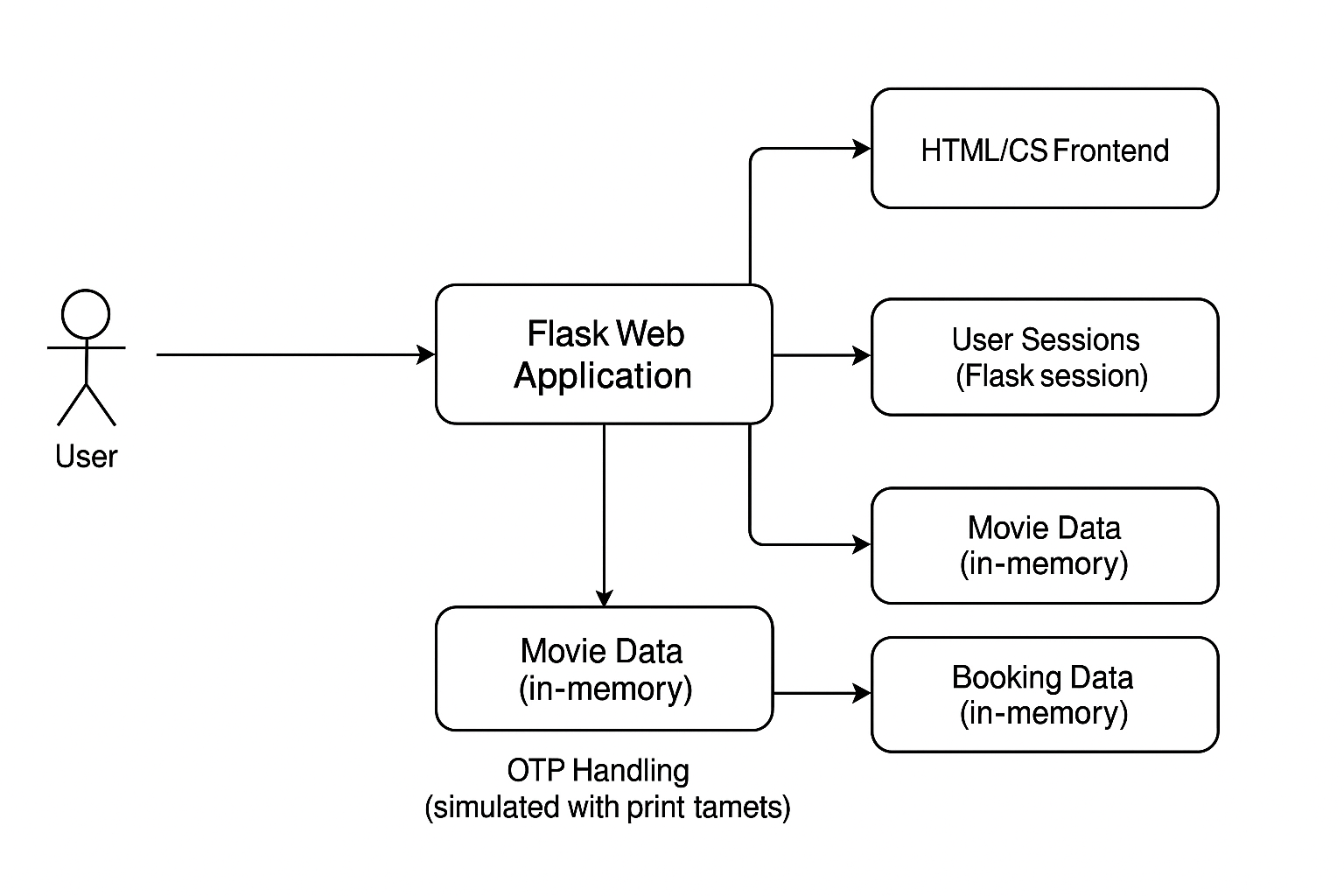
Traditional movie ticketing often involves standing in long queues, limited seat visibility, and poor management of user data. Users face delays, booking conflicts, and lack of convenience. This project addresses these inefficiencies by developing a fully digital ticket booking solution that is fast, secure, and easy to use, accessible from any device with internet access.

**🔹 3. Objectives**

* To create a user-friendly movie ticket booking platform.
* To support secure user registration, login, and session management.
* To allow users to view, search, and book movie tickets online.
* To provide admin visibility into booking records.
* To implement a forgot password system using OTP simulation.
* To offer a deployable and scalable backend using Flask.

🔹 4. **System Architecture**

User --> Flask Web App --> HTML/CSS Frontend  
 |--> DynamoDB (User, Booking data)  
 |--> AWS SNS (Email verification)  
 |--> EC2 Instance (Deployment)



**🔹 5. Modules / Features**

**🎟️ User Module**

* Register, login, logout functionality.
* Session management using Flask session.
* Password reset via OTP verification.

**🎬 Movie Module**

* Predefined list of movies with rating, time, price, and image.
* Search functionality based on movie name.

**🧾 Booking Module**

* Allows users to book tickets by submitting their name and ticket count.
* Displays success page after booking.

**🔐 Admin Module**

* Admin dashboard to view all bookings made across users.

**✉️ Forgot Password**

* User enters email → receives OTP via print → verifies → resets password.

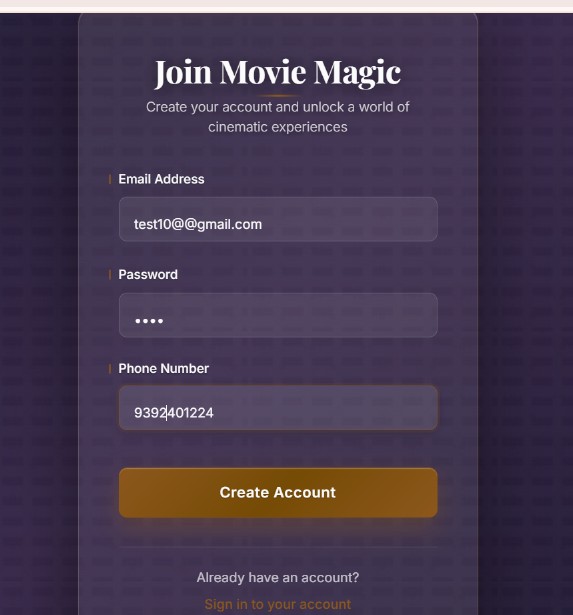
🔹 6. **Technologies Used**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | HTML, CSS, Bootstrap (in templates) |
| Backend | Python Flask |
| Session Mgmt | Flask session |
| OTP Simulation | Python random + print() |
| Deployment | Localhost (supports AWS EC2) |
| Database | In-memory lists (users, bookings) |

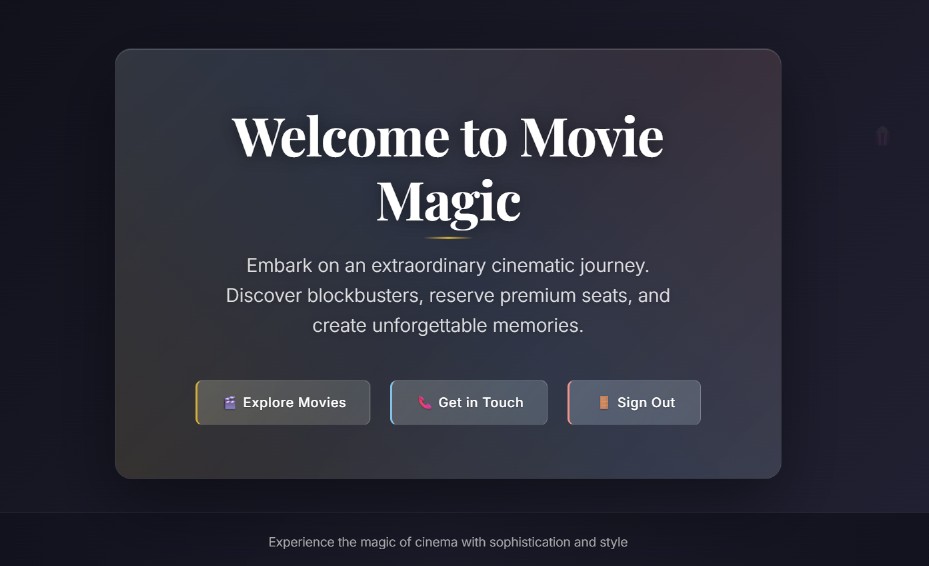
# 🔹 7. Snapshots Login Page:

# 

**Register Page:**

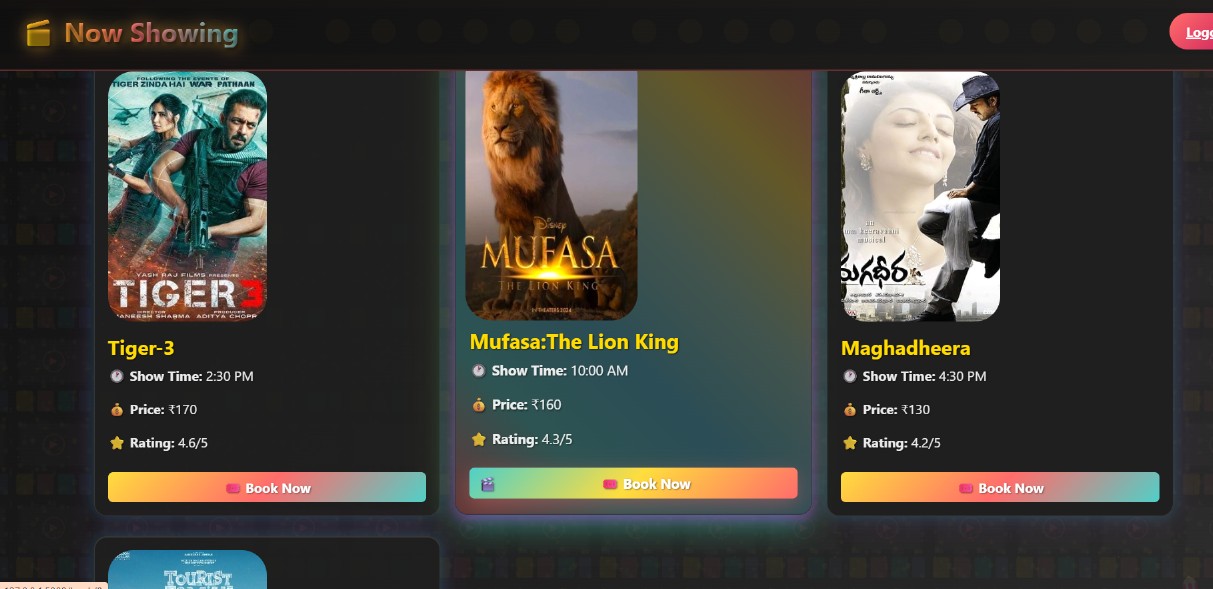


# Home Page:

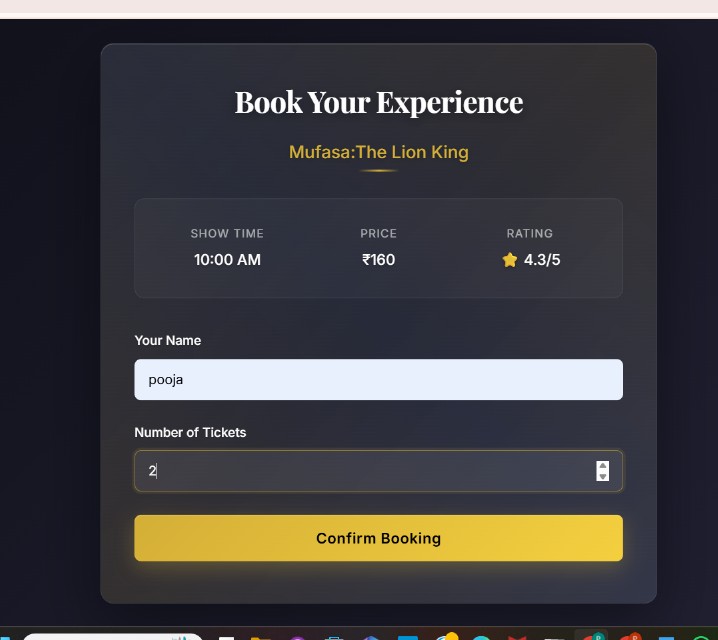


# Movies Page with ratings:

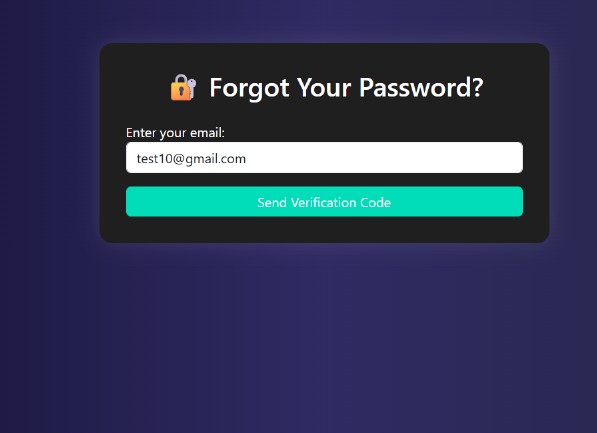
# 



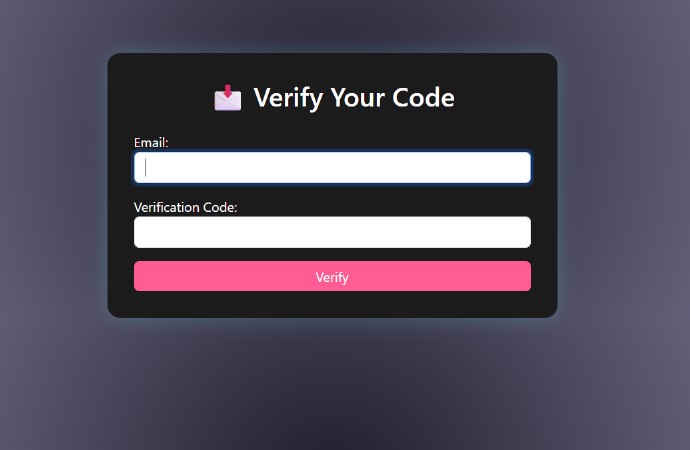
# Booking Page:



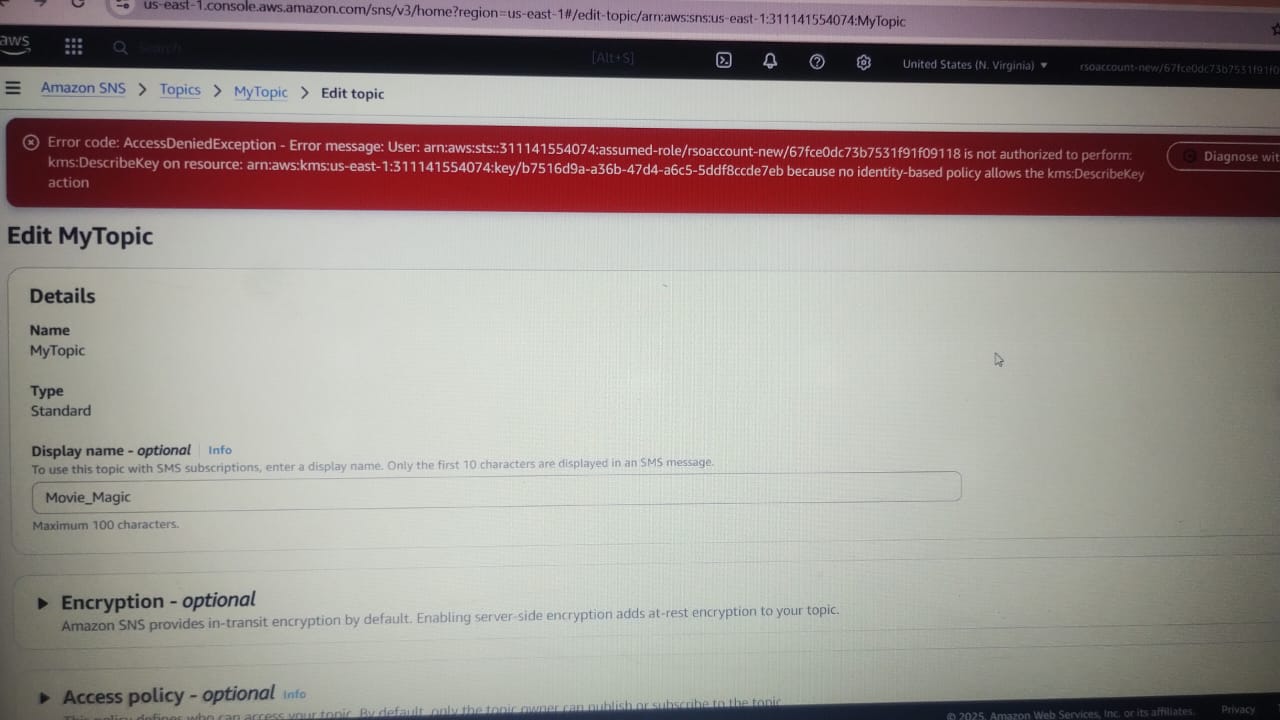
**Forgot Password OTP Screen:**

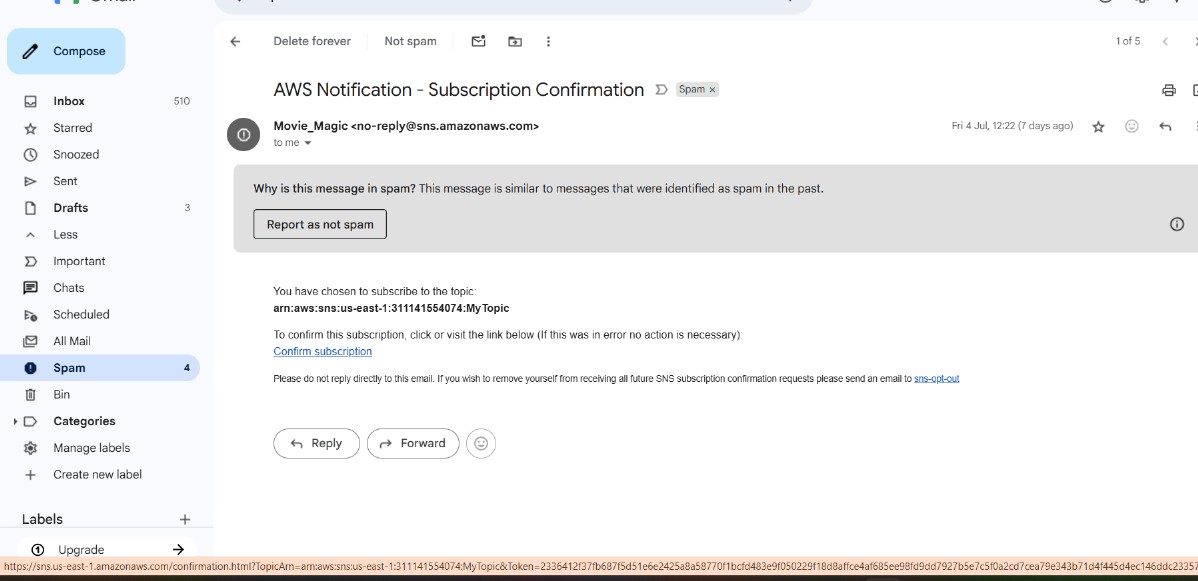


# Verify Page:

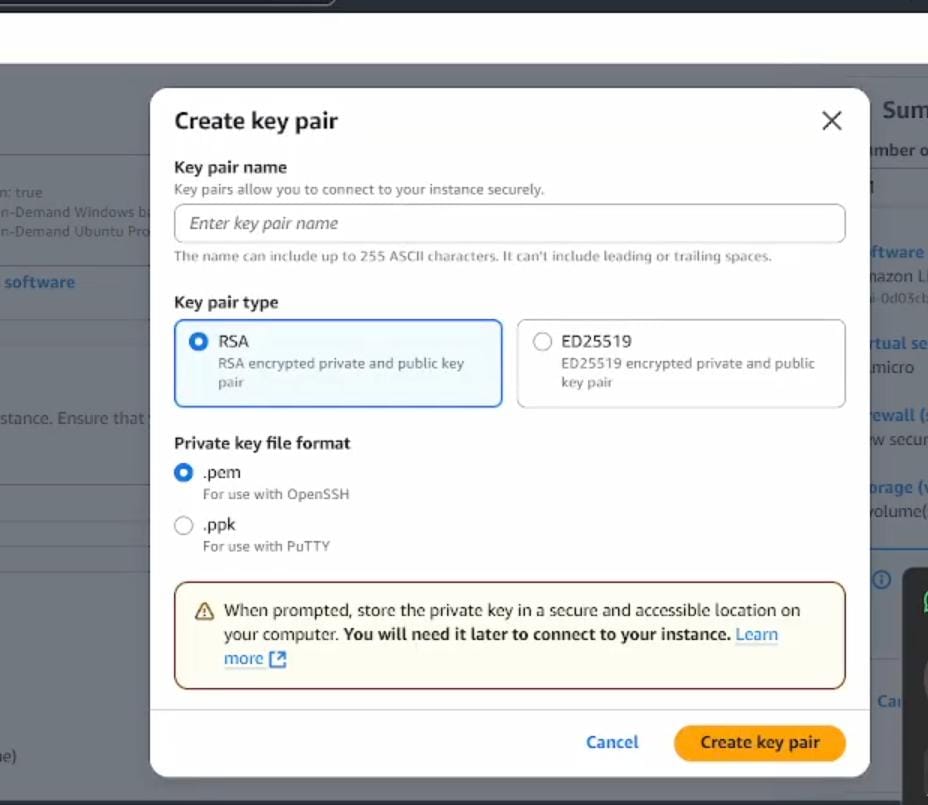


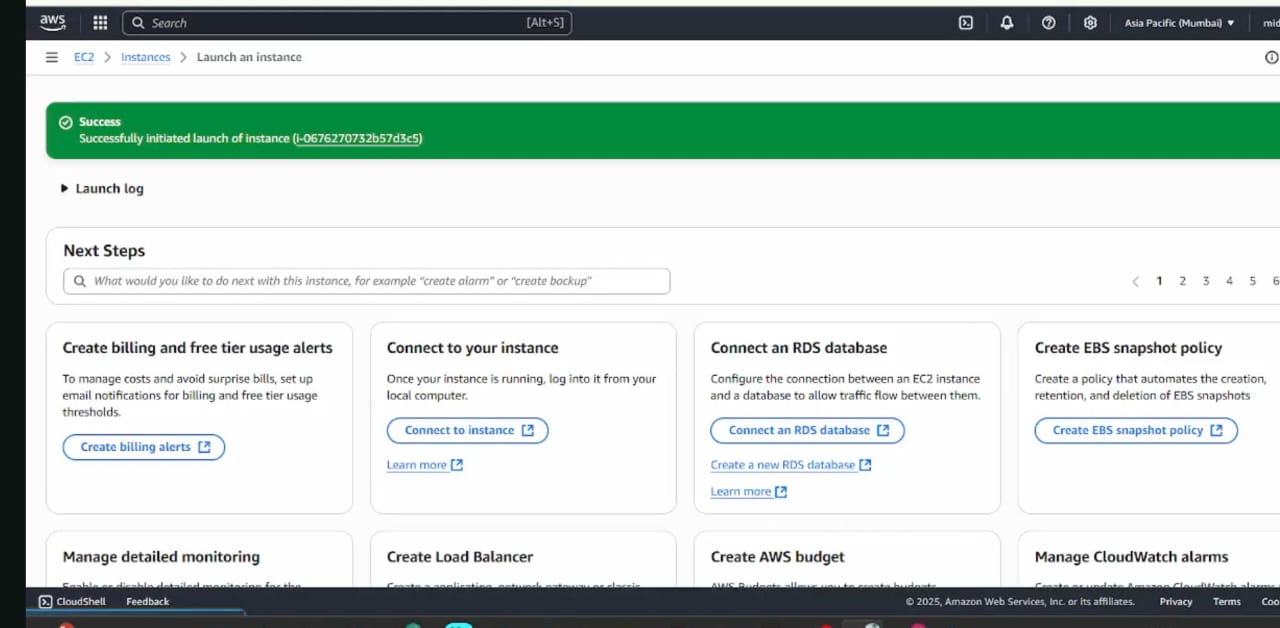
# SNS Email Confirmation:

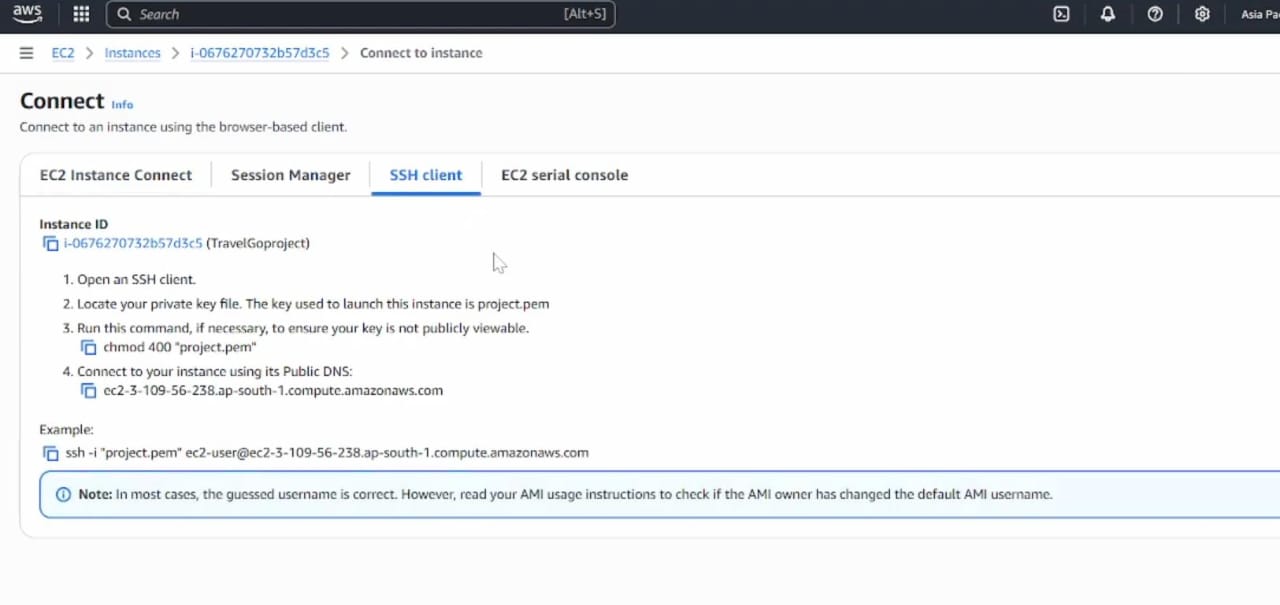




# EC2 Instance Running Flask App:

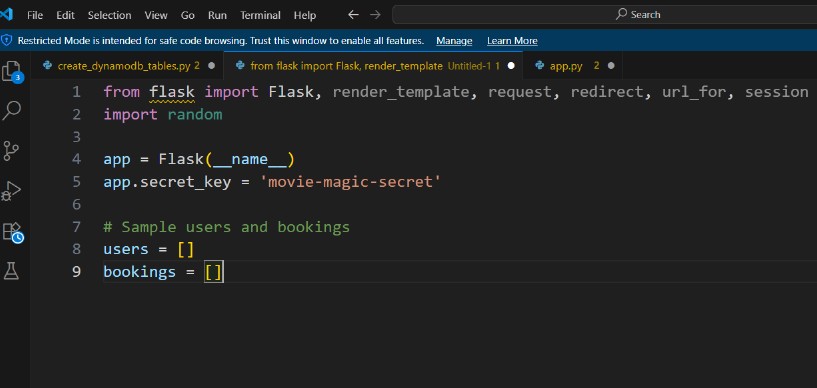






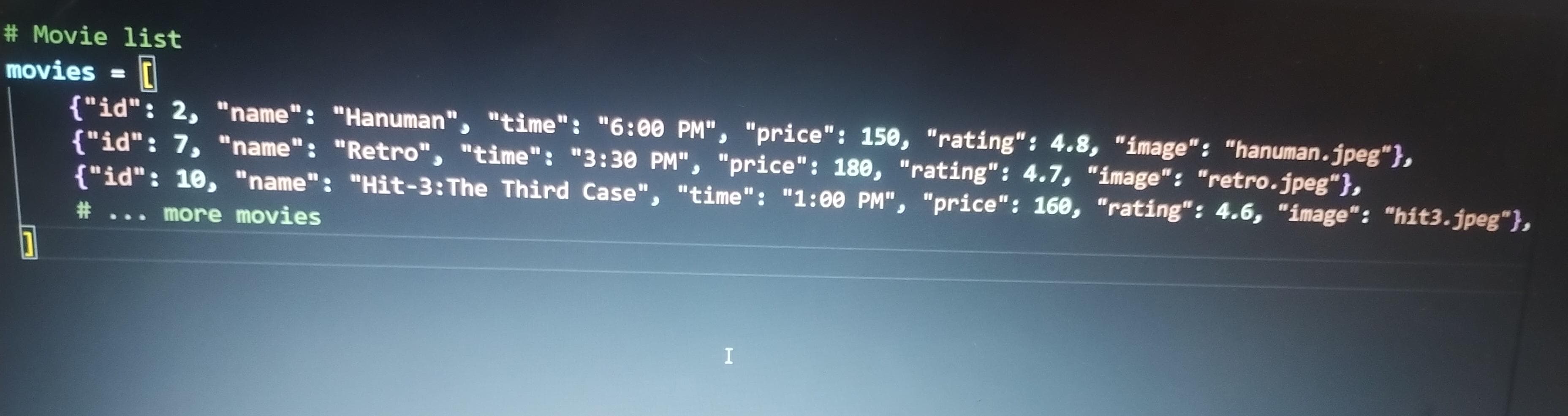
🔹 8. **Code Overview**

* **Flask App Initialization:**



**Description:** Import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification. Initialize the Flask application instance using **“Flask(\_\_name\_\_)”**to start building the web app.

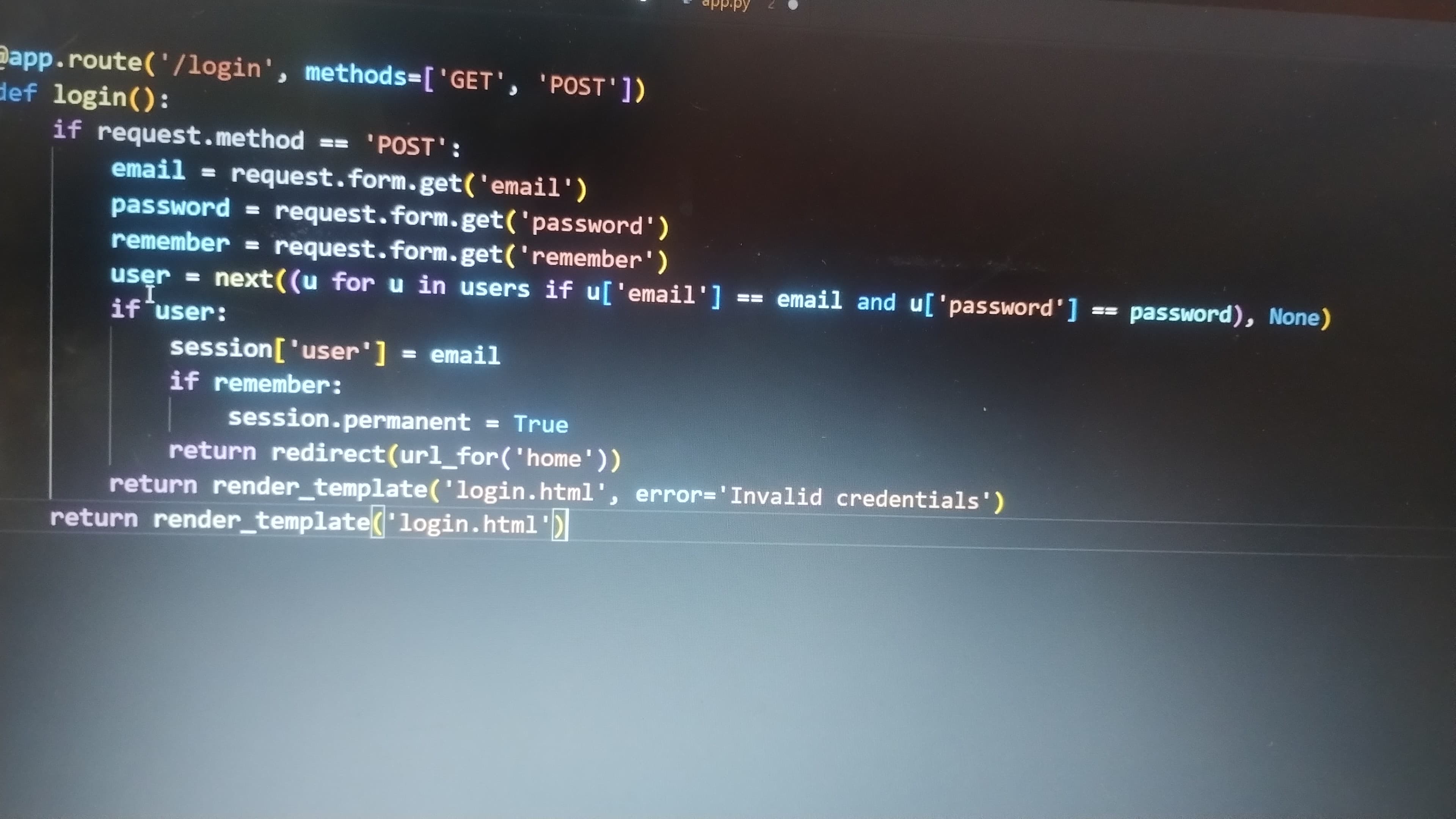
* **Data Storage Setup:**



**Description:** Initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests. Create static movie database with complete movie information including ID, name, showtime, pricing, ratings, and image references

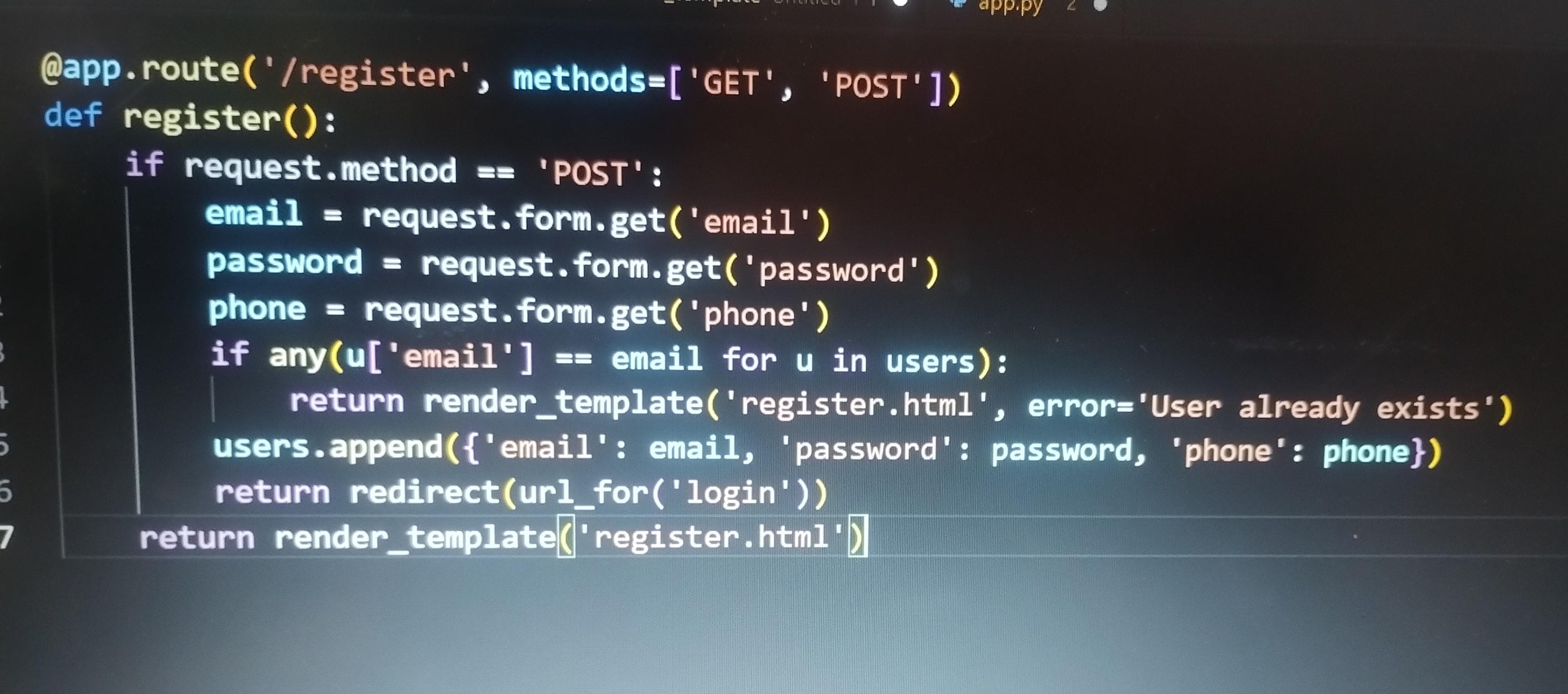
* **Authentication Routes:**

**1)Login Routes**



**Description**: Define the home route **`/`** to automatically redirect users to the register page when they access the base URL. Handle user authentication by validating email and password credentials against the user database, create secure sessions for authenticated users, and support "Remember Me" functionality for persistent login sessions.

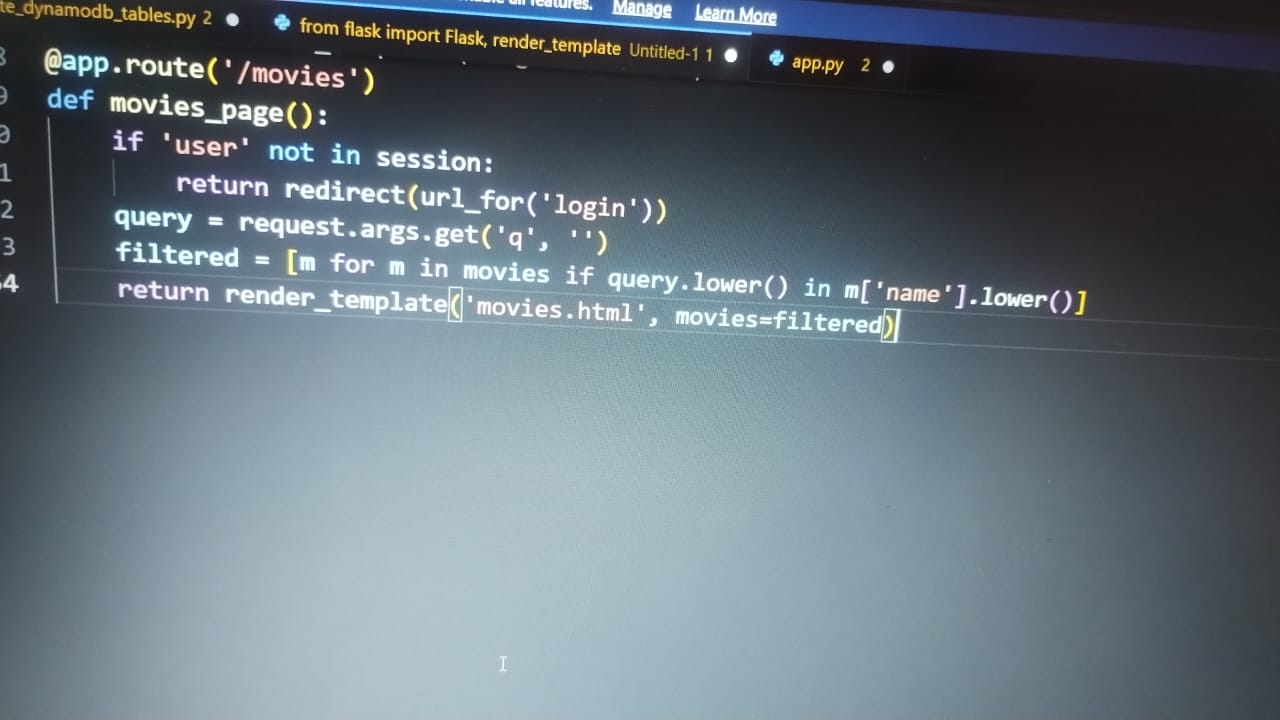
**2)Register Routes**



**Description:** Define **`/register`** route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count initialized to 0, and send an SNS notification for new user registration.

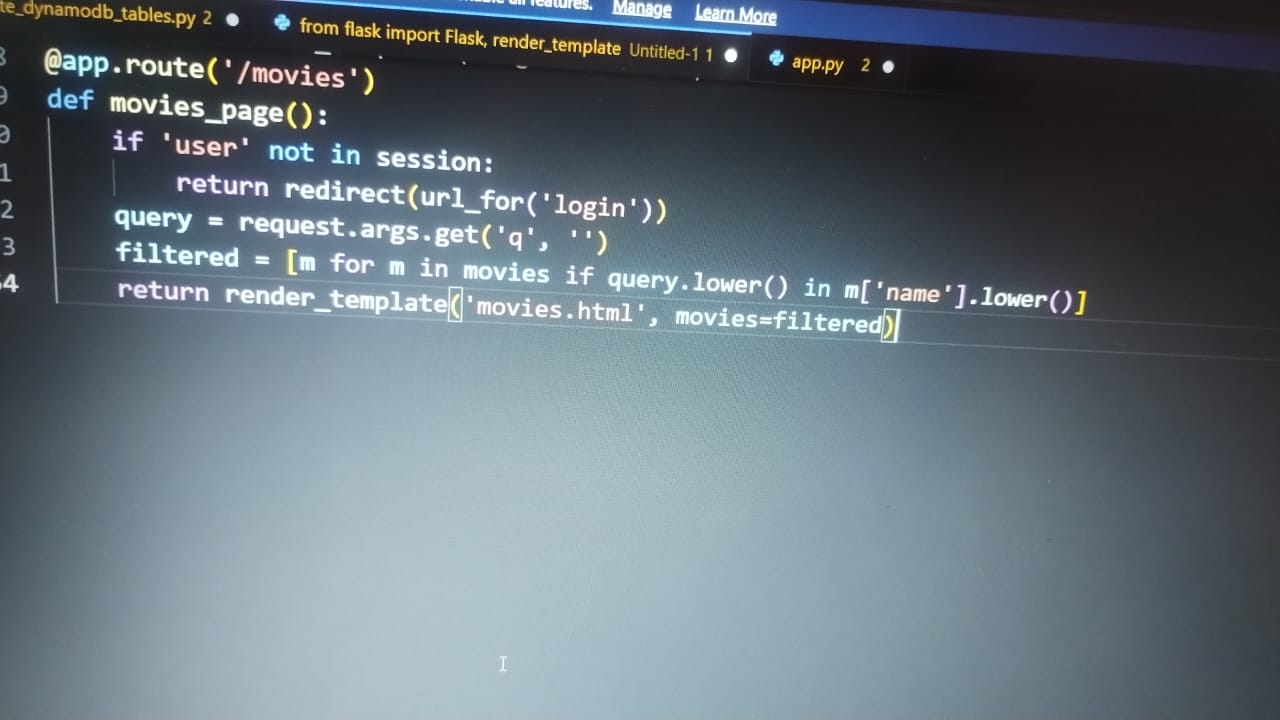
* **Movie Management Routes:**

**1)Movies Display Route**



**Description:** Implement authentication guard to ensure only logged-in users can access movies. Process search queries using case-insensitive filtering to match movie names. Return filtered movie list to the template for dynamic display with search functionality.

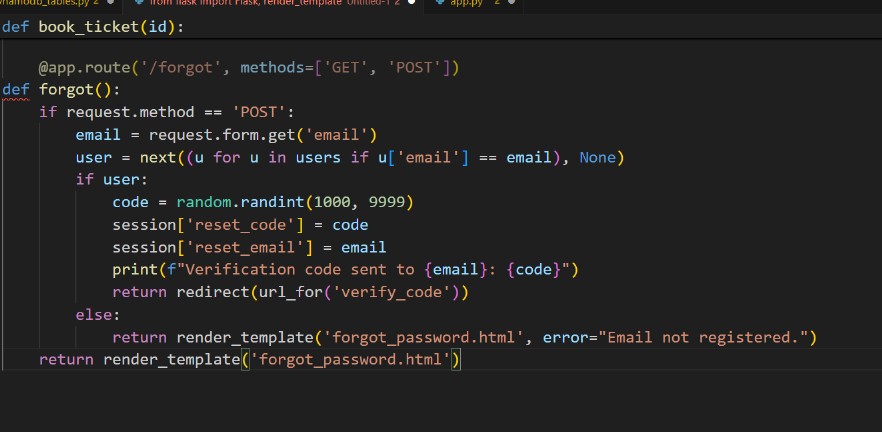
**2)Ticket Booking Route**

****

**Description:** Handle ticket booking process by validating movie existence using movie ID, processing booking form data including customer name and ticket quantity, storing booking information in the database, and displaying confirmation page with booking details and total price calculation.

* **Password Reset System:**

**1) Forgot Password Route**

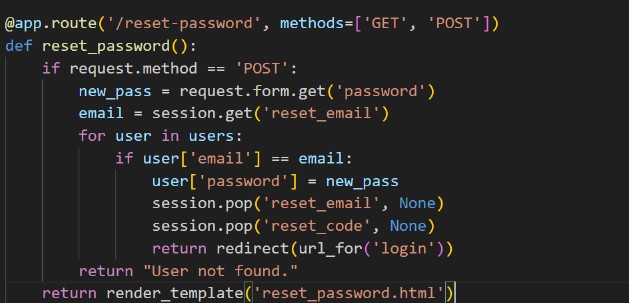


**Description:** Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the `sns\_topic\_arn` space, along with the region name where the SNS topic is created. Validate user email existence, generate 4-digit OTP code, store reset session data, and trigger SNS notification for password reset verification.

**2)OTP Verification Route**

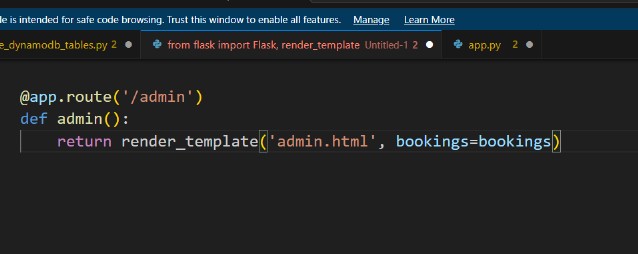


**3)Password Reset Route**



**Description:** Process the new password update by locating the user account using session-stored email, updating the password in the database, clearing all reset-related session data for security, and redirecting to login page for immediate authentication with new credentials.

* **Admin Management Route:**



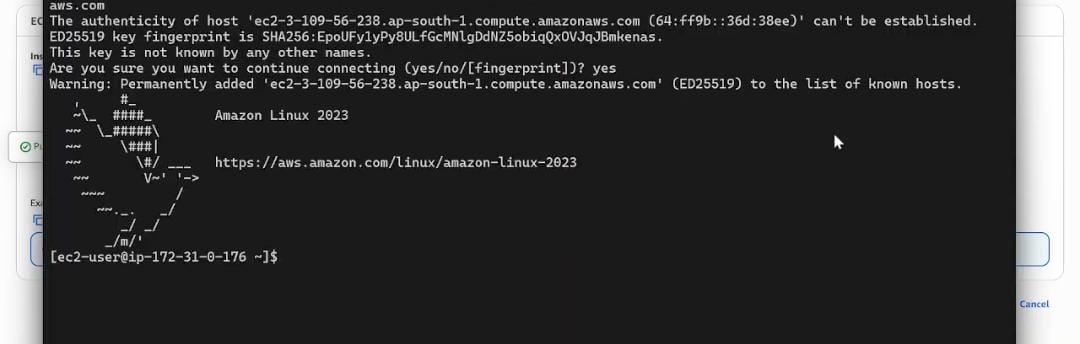
**Description**: Provide administrative access to view all booking records, display booking statistics including total bookings and revenue calculations, and enable management oversight of the ticket booking system operations.

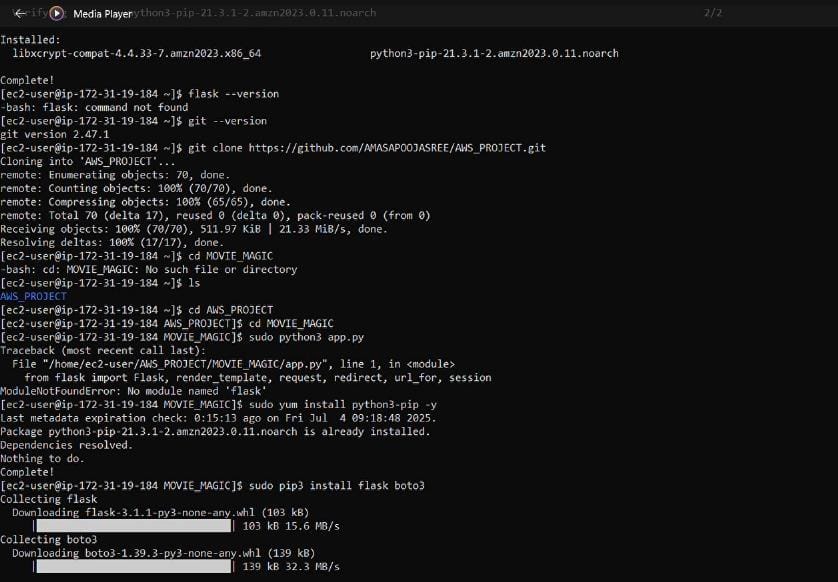
**🔹 9. Challenges Faced**

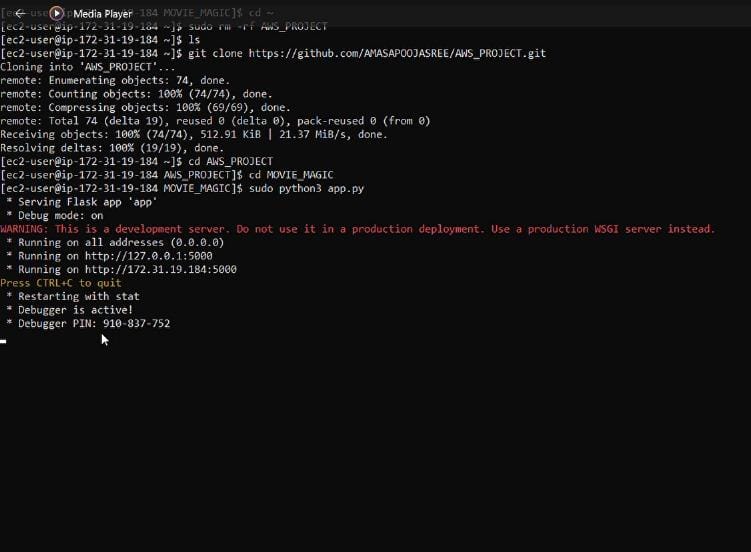
* Maintaining user sessions across multiple routes.
* Simulating OTP-based password reset without email services.
* Managing in-memory data without persistence.
* Form input validation and error handling.
* Preparing for real-time deployment or AWS integration.
* EC2 connectivity and permission errors
* DynamoDB data insertion and read issues
* Email delivery via AWS SNS during password reset
* PowerShell vs Git Bash permission differences
* Time constraints during cloud deployment and configuration

**🔹 10. Cloud Deployment**

**\*Open PowerShell:**

****

****



**🔹 11. Conclusion**

****

**🔹 12. References**

* Flask Documentation: <https://flask.palletsprojects.com/>
* W3Schools (HTML, CSS): <https://www.w3schools.com/>
* GitHub Docs: <https://docs.github.com/>
* AWS EC2: <https://docs.aws.amazon.com/ec2/>
* AWS DynamoDB: <https://docs.aws.amazon.com/dynamodb/>
* AWS SNS: <https://docs.aws.amazon.com/sns/>

Top of Form

* Bottom of Form