

Movie Magic–Smart Movie Ticket Booking System

Project Overview :

Movie Magic is a cloud-enabled web application that allows users to browse movies, select show timings, and book movie tickets online. The system is powered by Python Flask for backend logic, AWS DynamoDB for storing booking data, AWS SNS for sending real-time booking confirmations, and AWS EC2 for deployment. The project demonstrates how cloud services can be used to build scalable, reliable, and real-time web applications.

Scenario 1: User Browses and Books a Ticket

- User visits the Movie Magic homepage.
- The system displays a list of available movies with show timings.
- User fills in their name and email, selects a movie and timing, and clicks “Book Now.”
- Flask backend saves the booking in AWS DynamoDB.
- The user receives a confirmation email via AWS SNS.
- The confirmation page is displayed with their booking details.

Scenario 2: User Enters Invalid Email Address

- User inputs an invalid or misspelled email.
- Form submission fails validation or AWS SNS can't deliver the email.
- Flask may still save the booking, but the email notification does not reach the user.

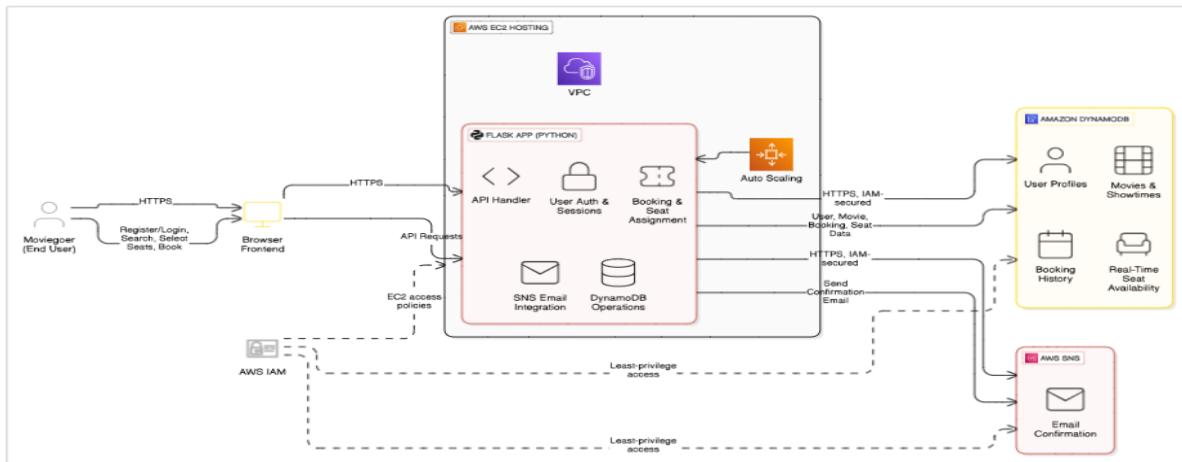
Scenario 3: Admin Views Booking Records (Future Enhancement)

- Admin logs in to a protected route (not implemented yet).
- Flask app fetches all booking records from DynamoDB.
- Admin can view or export booking data.

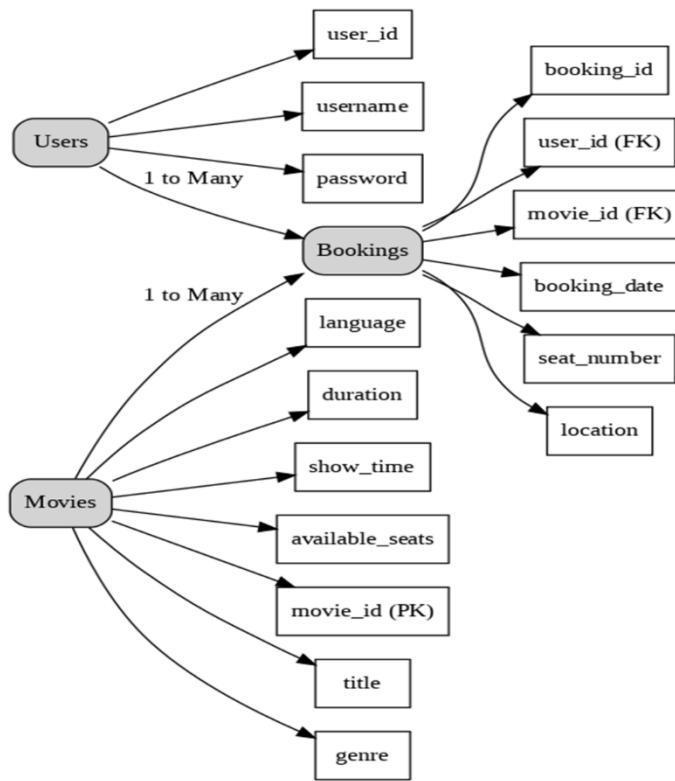
Scenario 4: SMS Booking Confirmation (Optional via SNS)

- User selects phone number option and enters mobile number.
- AWS SNS sends SMS confirmation instead of email.

AWS ARCHITECTURE



Entity Relationship (ER) Diagram:



Pre-requisites:

1. **AWS Account Setup:** [AWS Account Setup](#)
2. **Understanding IAM:** [IAM Overview](#)
3. **Amazon EC2 Basics:** [EC2 Tutorial](#)
4. **DynamoDB Basics:** [DynamoDB Introduction](#)
5. **SNS Overview:** [SNS Documentation](#)
6. **Git Version Control:** [Git Documentation](#)

Project WorkFlow:

1. AWS Account Setup and Login

Activity 1.1: Set up an AWS account if not already done.

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

4. Backend Development and Application Setup

Activity 4.1: Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

6. EC2 Instance Setup

Activity 6.1: Launch an EC2 instance to host the Flask application.

Activity 6.2: Configure security groups for HTTP, and SSH access.

7. Deployment on EC2

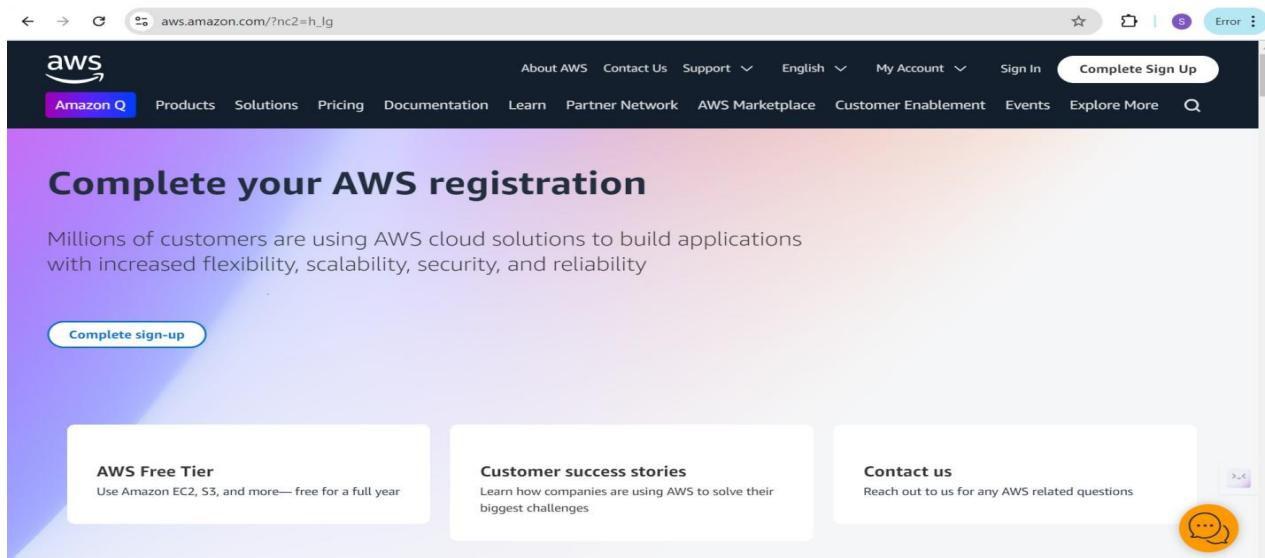
Activity 7.1: Upload Flask Files

Activity 7.2: Run the Flask App

8. Testing and Deployment

Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

1: AWS Account Setup and Login



2: Log in to the AWS Management Console.

A screenshot showing two side-by-side pages. On the left is the AWS "Sign in" page. It features the AWS logo at the top, followed by a "Sign in" section. Inside, there are two radio button options: "Root user" (selected) and "IAM user". The "Root user" option is described as an account owner performing tasks requiring unrestricted access. Below this is a "Root user email address" input field containing "username@example.com" and a "Next" button. At the bottom, a small note states: "By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information." On the right is the "AI Use Case Explorer" page. It has a dark purple gradient background with white text. The title "AI Use Case Explorer" is at the top, followed by the subtext "Discover AI use cases, customer success stories, and expert-curated implementation plans". At the bottom is a "Explore now >" button.

After logging into the AWS Console:

Once you're logged in, you can access a wide range of AWS services from the dashboard. Use the search bar at the top to quickly navigate to services like EC2, DynamoDB, SNS, or IAM as needed for your project setup.

3. IAM Roles Creation

The screenshot shows the AWS IAM Dashboard. On the left, a sidebar navigation includes 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'User groups', 'Users', 'Roles', 'Policies', 'Identity providers', 'Account settings', and 'Root access management'), 'Access reports' (with 'Access Analyzer', 'Resource analysis', 'Unused access', 'Analyzer settings', 'Credential report', and 'Organization activity'), and 'CloudShell' and 'Feedback' buttons. The main content area has a blue header bar with a message: 'New access analyzers available' and 'Create new analyzer'. Below this is the 'IAM Dashboard' section with tabs for 'Info' and 'C'. It contains three main boxes: 'IAM resources' (with an 'Access denied' error message for 'iam:GetAccountSummary' action), 'AWS Account' (with an 'Access denied' message for 'iam>ListAccountAliases' action), and 'What's new' (with a 'View all' link). At the bottom, there are links for 'Policy simulator', '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

The screenshot shows the 'Create role' wizard, Step 1: Use case. The title is 'Use case' with the sub-instruction 'Allow an AWS service like EC2, Lambda, or others to perform actions in this account.' A 'Service or use case' dropdown is set to 'EC2'. Below it, a list of options is shown, each with a radio button and a description. The 'EC2' option is selected. Other options include: 'EC2 Role for AWS Systems Manager', 'EC2 Spot Fleet Role', 'EC2 - Spot Fleet Auto Scaling', 'EC2 - Spot Fleet Tagging', 'EC2 - Spot Instances', 'EC2 - Spot Fleet', and 'EC2 - Scheduled Instances'. At the bottom right are 'Cancel' and 'Next' buttons.

Before assigning permissions, ensure that the IAM role is created and ready to be attached with appropriate AWS-managed policies for each required service.

The screenshot shows the 'Add permissions' step of creating a new IAM role. The left sidebar indicates the current step is 'Add permissions'. The main area displays a search interface for AWS managed policies. A search bar at the top contains the query 'ec2'. Below it, a table lists policies filtered by type ('AWS managed'). One policy, 'AmazonEC2FullAccess', is selected and highlighted with a blue border. Other visible policies include 'AmazonEC2ContainerRegistryFullAccess', 'AmazonEC2ContainerRegistryPowerUser', 'AmazonEC2ContainerRegistryPullOnly', 'AmazonEC2ContainerRegistryReadOnly', 'AmazonEC2ContainerServiceAutoscaleRole', 'AmazonEC2ContainerServiceEventsRole', 'AmazonEC2ContainerServiceforEC2Role', 'AmazonEC2ContainerServiceRole', and 'AmazonEC2ReadOnlyAccess'.

The screenshot shows the 'Add permissions' step of creating a new IAM role. The left sidebar indicates the current step is 'Add permissions'. The main area displays a search interface for AWS managed policies. A search bar at the top contains the query 'Dynamo'. Below it, a table lists policies filtered by type ('AWS managed'). One policy, 'AmazonDynamoDBFullAccess', is selected and highlighted with a blue border. Other visible policies include 'AmazonDynamoDBFullAccess_v2', 'AmazonDynamoDBFullAccesswithDataPipeline', 'AmazonDynamoDBReadOnlyAccess', 'AWSLambdaforDynamoDBExecutionRole', and 'AWSLambdaInvocation-DynamoDB'.

Roles > Create role

Add permissions

Permissions policies (2/1058) info

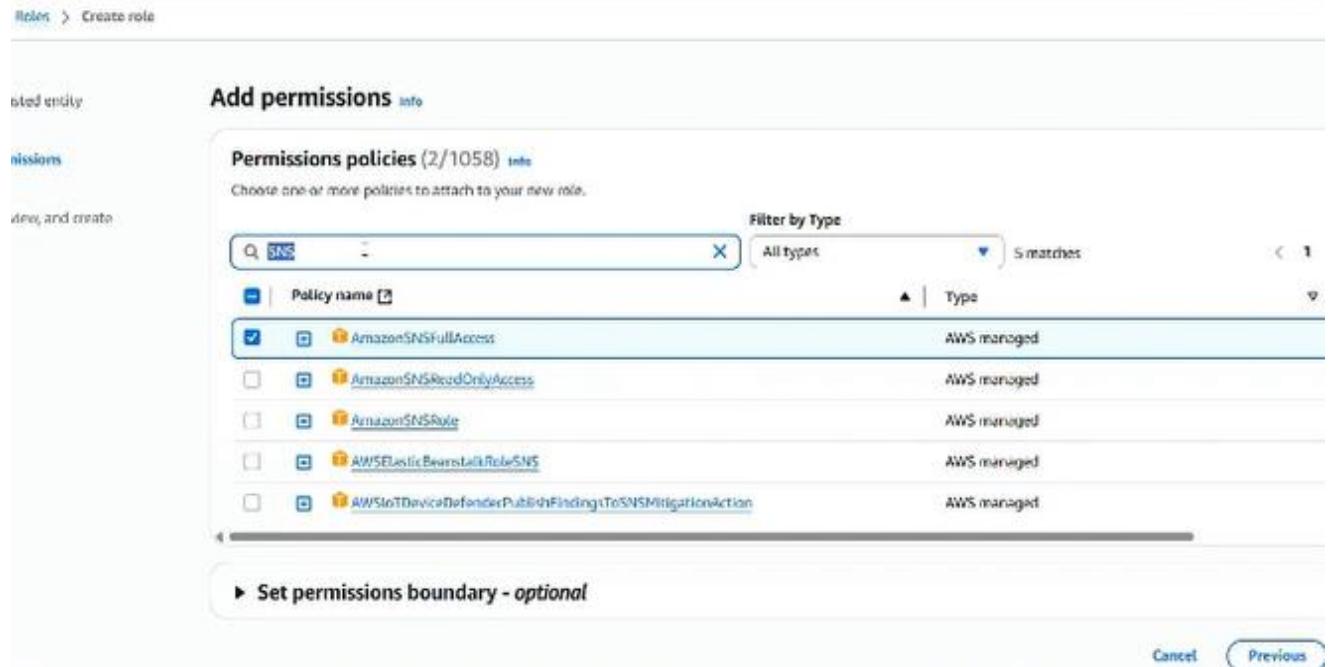
Choose one or more policies to attach to your new role.

Filter by Type

Policy name	Type
<input checked="" type="checkbox"/> AmazonSNSFullAccess	AWS managed
<input type="checkbox"/> AmazonSNSReadOnlyAccess	AWS managed
<input type="checkbox"/> AmazonSNSRole	AWS managed
<input type="checkbox"/> AWSLambdaBasicExecutionRoleSNS	AWS managed
<input type="checkbox"/> AWSIoTDeviceDefaultPublishingToSNSSignatureAction	AWS managed

▶ Set permissions boundary - optional

Cancel Previous



DynamoDB Database Creation and Setup

- In the AWS Console, navigate to DynamoDB and click on create tables

aws Services Search results for 'dyn'

Services

Features

Resources New

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

DynamoDB ☆
Managed NoSQL Database

Top features

Tables Imports from S3 Explore Items Clusters Reserved Capacity

Amazon DocumentDB ☆
Fully-managed MongoDB-compatible database service

CloudFront ☆
Global Content Delivery Network

Athena ☆
Serverless interactive analytics service

Features

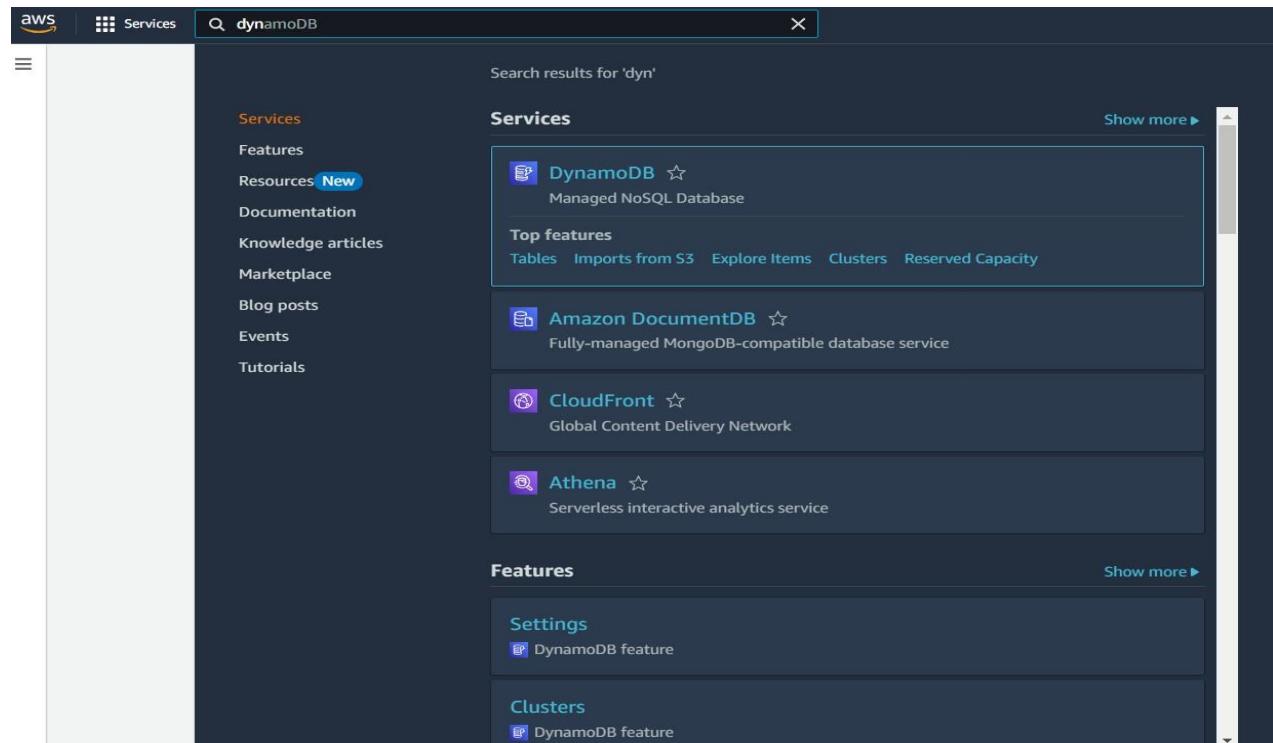
Show more ▶

Settings

DynamoDB feature

Clusters

DynamoDB feature



Create Users table with partition key "Email" with type String and click on create tables.

DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

UserID	String
--------	--------

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name	String
-------------------------	--------

1 to 255 characters and case sensitive.

Table settings

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

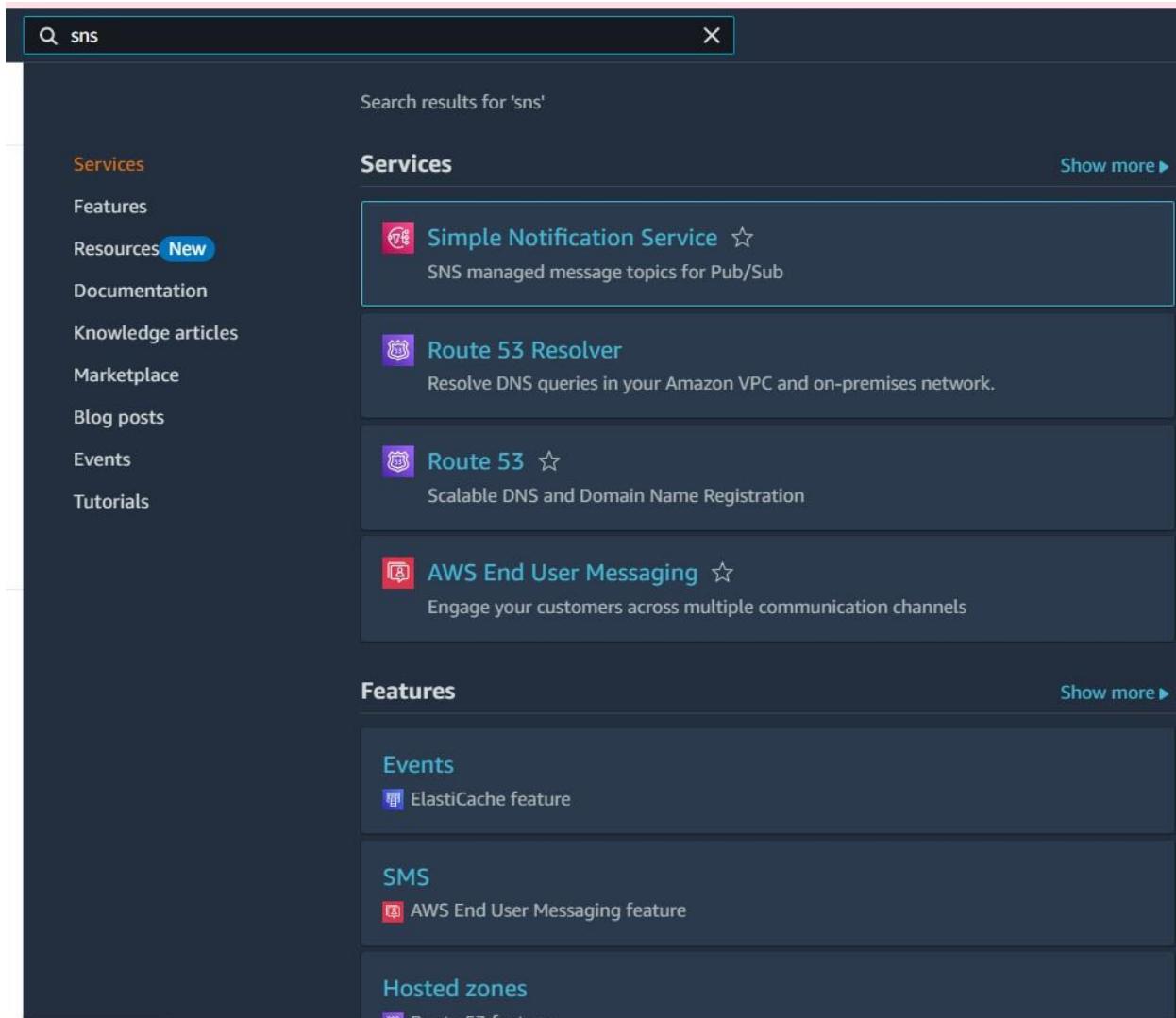
Add new tag

You can add 50 more tags.

Cancel Create table

SNS Notification Setup

1. Create SNS topics for sending email notifications to users.



The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are categorized under 'Services' and 'Features'.

Services

- Simple Notification Service (SNS) - Described as 'SNS managed message topics for Pub/Sub'.
- Route 53 Resolver - Described as 'Resolve DNS queries in your Amazon VPC and on-premises network.'
- Route 53 - Described as 'Scalable DNS and Domain Name Registration'.
- AWS End User Messaging - Described as 'Engage your customers across multiple communication channels'.

Features

- Events - Includes the 'ElastiCache feature'.
- SMS - Includes the 'AWS End User Messaging feature'.
- Hosted zones - Includes the 'Route 53 feature'.

The screenshot shows the Amazon SNS landing page. On the left, there's a sidebar with links: Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)). The main content area has a banner for 'New Feature' about FIFO topics. Below it, there's a section titled 'Application Integration' with the heading 'Amazon Simple Notification Service' and a sub-section 'Pub/sub messaging for microservices and serverless applications.' A descriptive paragraph follows. To the right, there's a 'Create topic' form with a 'Topic name' field containing 'MyTopic', a 'Next step' button, and a link to 'Start with an overview'. At the bottom right is a 'Pricing' section.

- Click on **Create Topic** and choose a name for the topic.

The screenshot shows the 'Topics' page under the 'Amazon SNS' navigation. The sidebar is identical to the previous screenshot. The main area shows a table header for 'Topics (0)' with columns for Name, Type, and ARN. Below the table, a message says 'No topics' and 'To get started, create a topic.' with a 'Create topic' button.

- Choose Standard type for general notification use cases and Click on Create Topic.

Name, review, and create

Role details

Role name

Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and '+,-,@,-' characters.

Description

Add a short explanation for this role.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/[\{\}]#\$%^&`~;"'

Step 1: Select trusted entities

Trust policy

```
1 - [ {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Action": [  
7                 "sts:AssumeRole"  
8             ],  
9             "Principal": {  
10                "Service": [  
11                    "ec2.amazonaws.com"  
12                ]  
13            }  
14        }  
15    ]  
16 }]
```

➤ Click on create topic

To begin configuring notifications, navigate to the SNS dashboard and click on “CreateTopic.” Choose the topic type (Standard or FIFO) based on your requirements. Provide a meaningful name for the topic that reflects its purpose (e.g., MovieMagic). This topic will serve as the communication channel for sending notifications to subscribed users

The screenshot shows the 'Create Topic' configuration page with several optional settings sections:

- Access policy - optional**: This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
- Data protection policy - optional**: This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
- Delivery policy (HTTP/S) - optional**: The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
- Delivery status logging - optional**: These settings configure the logging of message delivery status to CloudWatch Logs.
- Tags - optional**: A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)
- Active tracing - optional**: Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

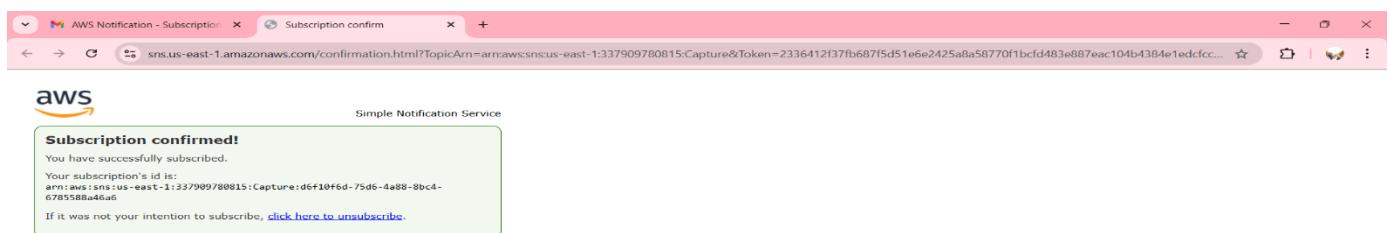
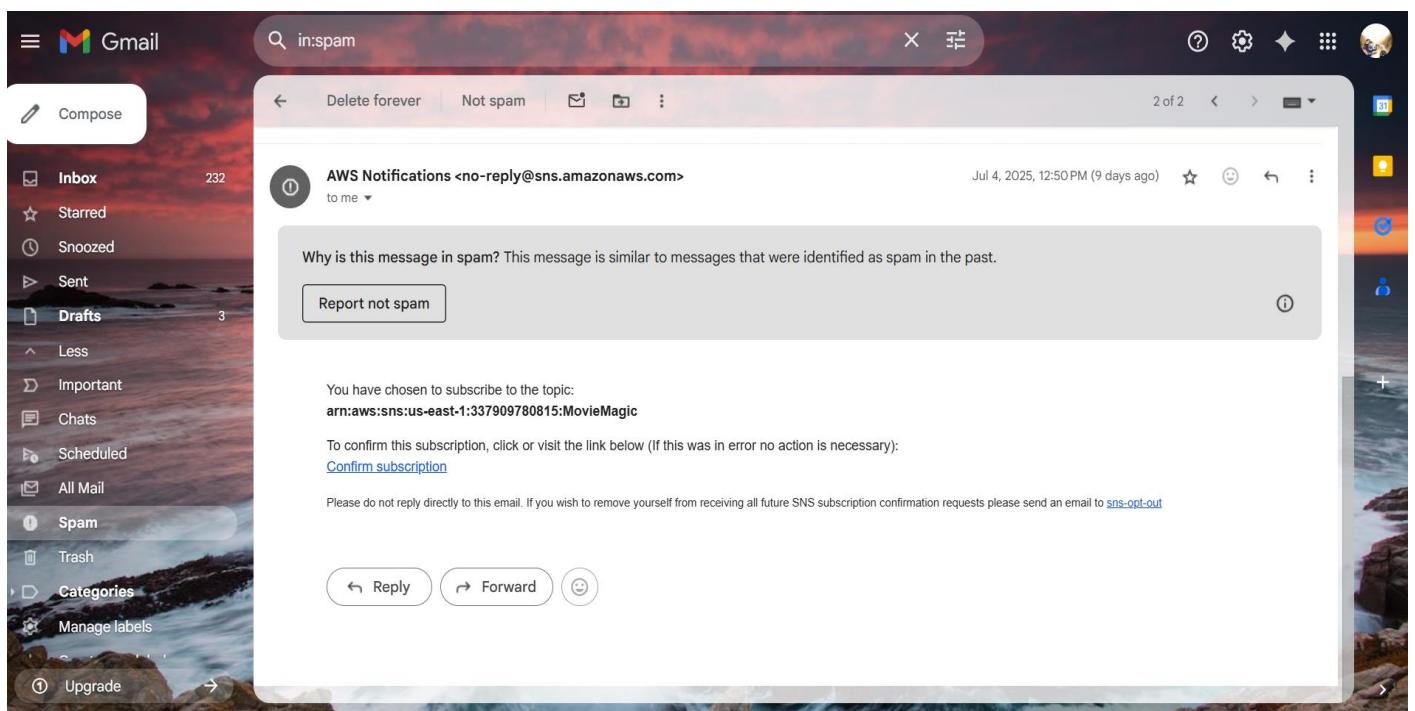
At the bottom right, there are 'Cancel' and 'Create topic' buttons.

Click on create Subscription:

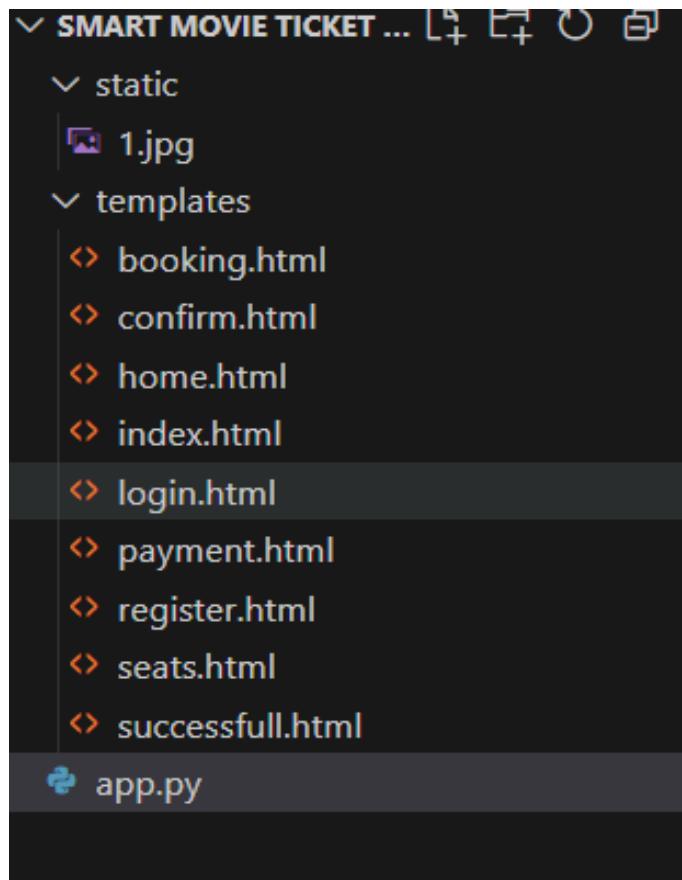
Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail

Once the email subscription is confirmed, the endpoint becomes active for receiving notifications. This ensures that users will instantly get booking confirmations or alerts.

You can manage or remove subscriptions anytime via the SNS dashboard. It's recommended to test the setup by publishing a sample message to verify successful delivery.



Backend Development and Application Setup



Develop the backend using Flask

Initialize the Flask application instance using `Flask(__name__)` to start building the web app.

```
app.py > ...
1  from flask import Flask, render_template, request, redirect, url_for
2  import uuid
3
4  app = Flask(__name__)
5
6  users = {}
7
8  @app.route('/')
9  def index():
10     return render_template('index.html')
11
12 @app.route('/register', methods=['GET', 'POST'])
13 def register():
14     if request.method == 'POST':
15         name = request.form['name']
16         email = request.form['email']
17         password = request.form['password']
18         if email in users:
19             return "User already exists!"
20         users[email] = {'name': name, 'password': password}
21         return redirect(url_for('login'))
22     return render_template('register.html')
23
```

- Flask: Framework used to build the web server
- render_template: Renders HTML templates from the templates/ folder
- request: Gets form data from user input
- redirect, url_for: Redirects users between pages
- uuid: Generates unique IDs for ticket confirmation

```

23
24     @app.route('/login', methods=['GET', 'POST'])
25     def login():
26         if request.method == 'POST':
27             email = request.form['email']
28             password = request.form['password']
29             user = users.get(email)
30             if user and user['password'] == password:
31                 return redirect(url_for('home'))
32             return "Invalid credentials!"
33         return render_template('login.html')
34
35     @app.route('/home')
36     def home():
37         return render_template('home.html')
38
39     @app.route('/booking')
40     def booking():
41         return render_template('booking.html')
42
43     @app.route('/seats', methods=['GET'])
44     def seats():
45         movie = request.args.get('movie')
46         price = request.args.get('price')
47         time = request.args.get('time')
48         location = request.args.get('location')
49         return render_template('seats.html', movie=movie, price=price, time=time, location=location)
50
51     @app.route('/confirm', methods=['POST'])
52     def confirm():
53         movie = request.form.get('movie')
54         location = request.form.get('location')
55         time = request.form.get('time')
56         seats = request.form.get('seats')
57         price = request.form.get('price')
58         print("DEBUG:", movie, seats, price, time, location)
59         return render_template('confirm.html', movie=movie, location=location, time=time, seats=seats, price=price)

59         return render_template('confirm.html', movie=movie, location=location, time=time, seats=seats, price=price)
60
61     @app.route('/payment', methods=['POST'])
62     def payment():
63         movie = request.form.get('movie')
64         seats = request.form.get('seats')
65         price = request.form.get('price')
66         time = request.form.get('time')
67         location = request.form.get('location')
68         if not all([movie, seats, price, time, location]):
69             return "Missing required parameters", 400
70         return render_template('payment.html', movie=movie, seats=seats, price=price, time=time, location=location)
71
72     @app.route('/successfull', methods=['GET'])
73     def successfull():
74         movie = request.args.get('movie')
75         seats = request.args.get('seats')
76         showtime = request.args.get('showtime')
77         location = request.args.get('location')
78         booking_id = str(uuid.uuid4())[:8].upper()
79         return render_template([
80             'successfull.html',
81             movie=movie,
82             showtime=showtime,
83             seats=seats,
84             location=location,
85             booking_id=booking_id
86         ])
87
88     if __name__ == '__main__':
89         app.run(host='0.0.0.0', port=5000, debug=True)

```

Deployment Code:

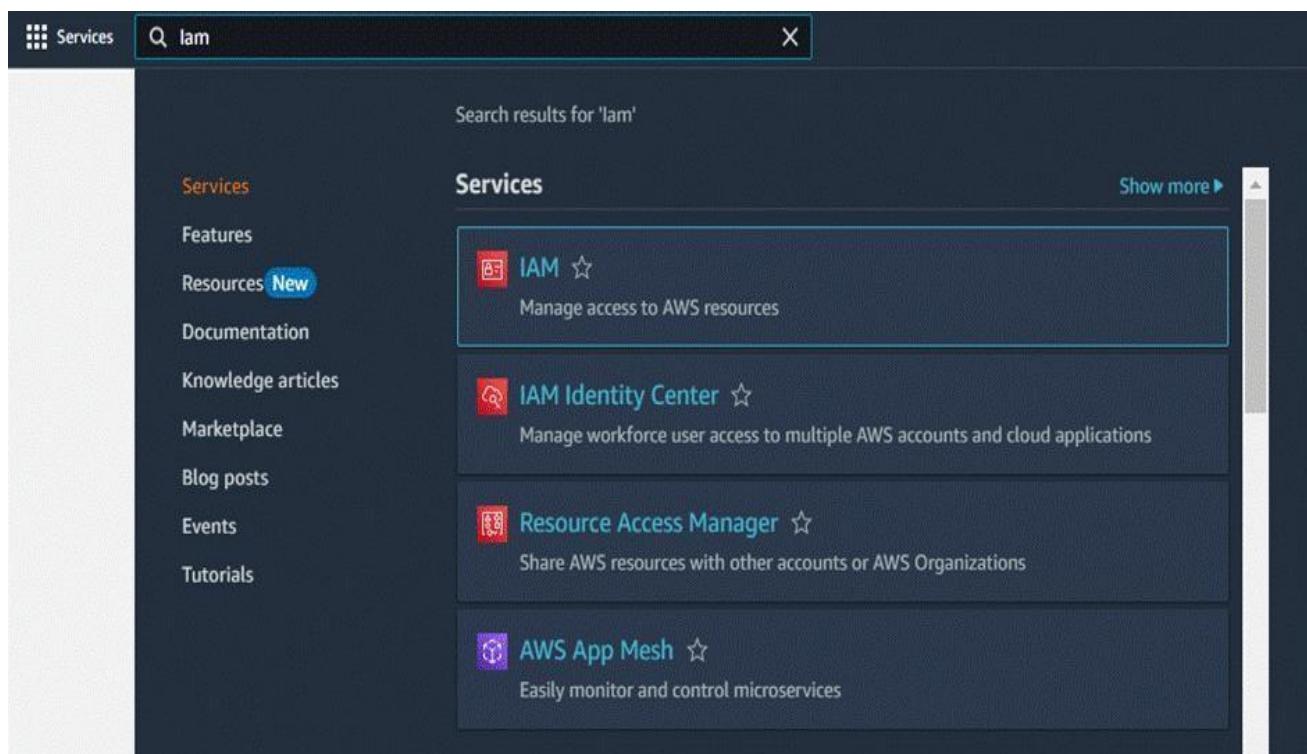
```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

IAM Role Setup:

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

Create IAM Role:

- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



Identity and Access Management (IAM)

IAM > Roles

Roles (6) [Info](#)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Search

Role name Trusted entities Last activity

Create rule

Step 1: Select trusted entity

Trusted entity type

- AWS service
Allow AWS services like S3, Lambda, or others to perform actions in this account.
- AWS account
Allow users in other AWS accounts belonging to one or a third party to perform actions in this account.
- Web identity
Allow users identified by your authentication web identity provider to assume the role to perform actions in this account.

User case

Select an AWS service for the role to identify or allow to perform actions in this account.

Service or use case

RID

Select a use case for the specified service.

Role use

- RID Role for AWS Systems Manager
Allows the role to call AWS services like CloudWatch and Systems Manager on your behalf.
- EC2 Role for Amazon RDS
Allows the role to call AWS services like Amazon RDS on your behalf.
- EC2 Role for Amazon Redshift
Allows the role to call AWS services like Amazon Redshift on your behalf.
- EC2 Role for Amazon Auto Scaling
Allows the role to call AWS services like Amazon Auto Scaling on your behalf.
- EC2 Role Tagging
Allows the role to create, view, manage, and delete tags on the specified resources on your behalf.
- EC2 - Spot Instances
Allows the role to request and manage spot instances on your behalf.
- EC2 - Spot Fleet
Allows the role to request and manage spot fleet instances on your behalf.
- EC2 - Scheduled Instances
Allows the role to request and manage scheduled instances on your behalf.

Cancel Next

New Feature

Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)

Create topic

Details

Type | [Info](#)

Topic type cannot be modified after topic is created

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name

MovieTicketNotifications

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional | [Info](#)

To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

Load your Project Files to GitHub

- Load your Flask app and Html files into GitHub repository.



static

Initial commit



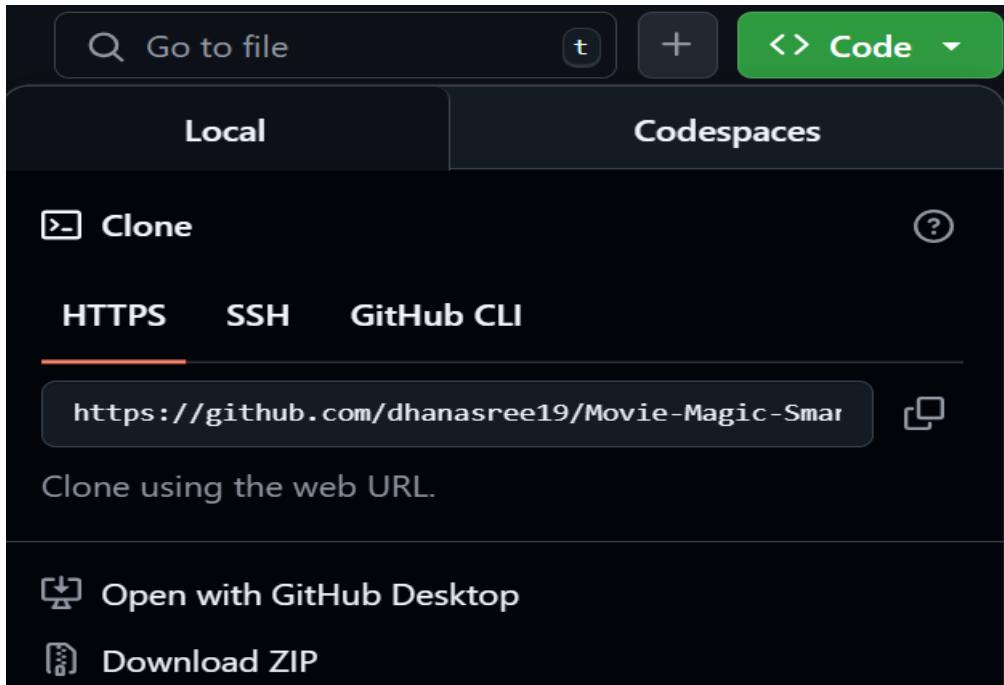
templates

Update statistics.html



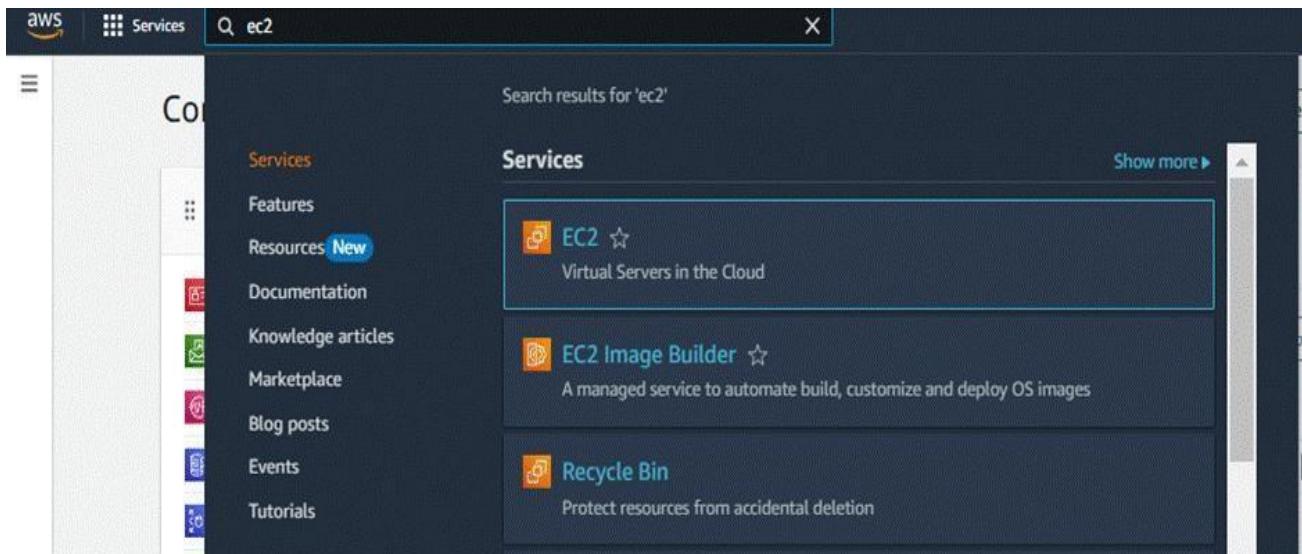
app.py

Update app.py



Launch an EC2 instance to host the Flask

- Launch EC2 Instance
- In the AWS Console, navigate to EC2 and launch a new instance.



- Click on Launch instance to launch EC2 instance

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main area has a header with tabs for Instances (selected) and Info, and a search bar. Below the header, there are filters for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. A message says 'No instances' and 'You do not have any instances in this region'. At the bottom right of the main area is a large orange 'Launch instances' button.

ⓘ It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

The screenshot shows the 'Launch an instance' wizard. The first step, 'Name and tags', is selected. It has a 'Name' field containing 'MovieMagid' and a 'Add additional tags' link. Below this is a section titled 'Application and OS Images (Amazon Machine Image)' with a 'Quick Start' tab selected. It shows recent AMIs: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. There's also a 'Search our full catalog including 1000s of application and OS images' bar and a 'Browse more AMIs' link.

- Create and download the key pair for Server access.

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro	Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true	
On-Demand Linux base pricing: 0.0124 USD per Hour	
On-Demand Windows base pricing: 0.017 USD per Hour	
On-Demand RHEL base pricing: 0.0268 USD per Hour	
On-Demand SUSE base pricing: 0.0124 USD per Hour	

All additional costs apply for AMIs with pre-installed software

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select [Create new key pair](#)

Create key pair

Key pair name
Key pairs allow you to connect to your instance securely.

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

RSA
RSA encrypted private and public key pair

ED25519
ED25519 encrypted private and public key pair

Private key file format

.pem
For use with OpenSSH

.ppk
For use with PuTTY

⚠️ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

[Cancel](#) [Create key pair](#)

Edit Inbound Rules:

Select the EC2 instance you just launched and ensure it's in the "running" state. Navigate to the "Security" tab, then click "Edit inbound rules." Add a new rule with the following settings: Type – Custom TCP, Protocol – TCP, Port Range – 5000, Source – Anywhere (IPv4) 0.0.0.0/0. This allows external access to your Flask application running on port 5000.

Inbound Security Group Rules

▼ Security group rule 1 (TCP: 22, 0.0.0.0/0)

Type Info ssh	Protocol Info TCP	Port range Info 22	Remove
Source type Info Anywhere	Source Info Add CIDR, prefix list or security 0.0.0.0/0 X	Description - optional Info e.g. SSH for admin desktop	

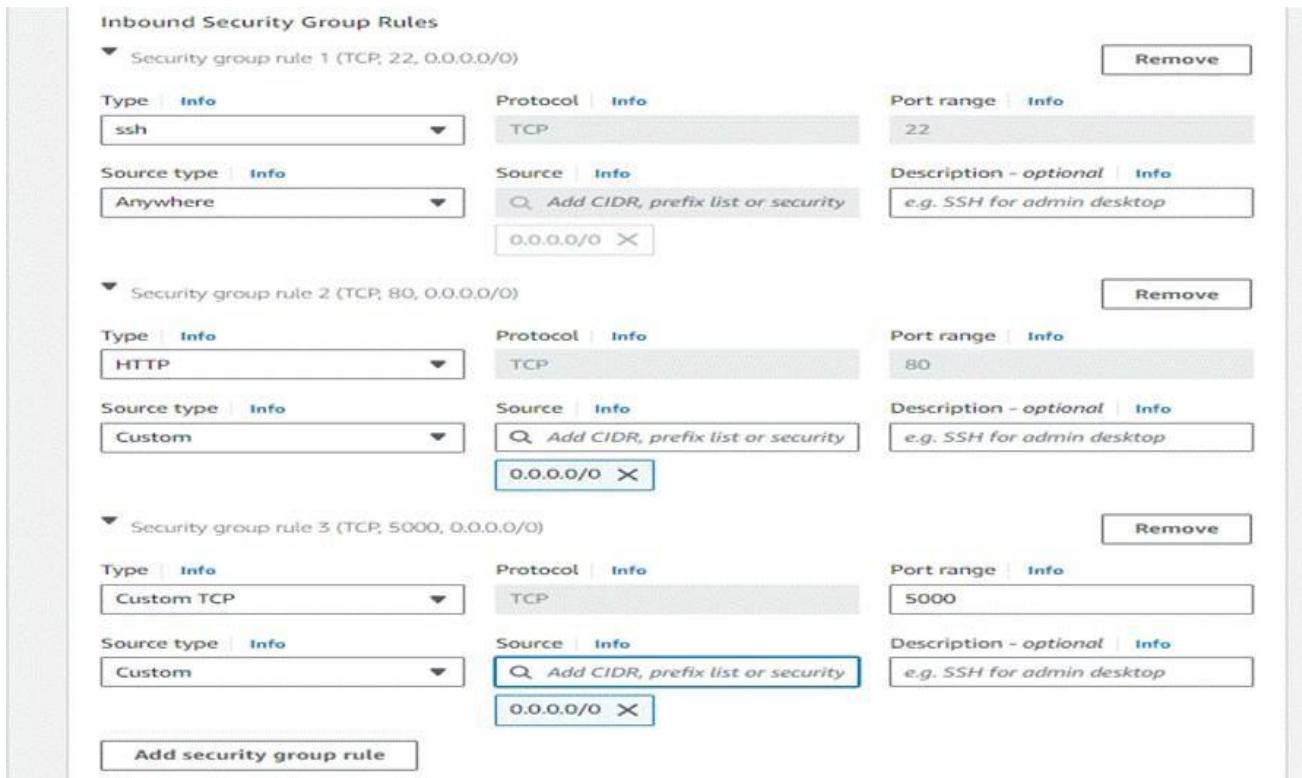
▼ Security group rule 2 (TCP: 80, 0.0.0.0/0)

Type Info HTTP	Protocol Info TCP	Port range Info 80	Remove
Source type Info Custom	Source Info Add CIDR, prefix list or security 0.0.0.0/0 X	Description - optional Info e.g. SSH for admin desktop	

▼ Security group rule 3 (TCP: 5000, 0.0.0.0/0)

Type Info Custom TCP	Protocol Info TCP	Port range Info 5000	Remove
Source type Info Custom	Source Info Add CIDR, prefix list or security 0.0.0.0/0 X	Description - optional Info e.g. SSH for admin desktop	

Add security group rule



Instances (1/1) Info		Last updated less than a minute ago	Connect	Instance state ▾	Actions ▾	Launch	
<input type="text"/> Find Instance by attribute or tag (case-sensitive)		All states ▾					
<input checked="" type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input checked="" type="checkbox"/>	MovieMagic	i-047dbc81fe3dcad56	Stopped	t2.micro	-	View alarms +	ap-south-1b

- Now connect the EC2 with the files

Connect to instance [Info](#)

Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

[EC2 Instance Connect](#) [Session Manager](#) [SSH client](#) [EC2 serial console](#)

Port 22 (SSH) is open to all IPv4 addresses
 Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: **13.233.177.0/29**. [Learn more](#).

Instance ID
 i-001861022fbcac290 (InstantLibraryApp)

Connection Type

Connect using EC2 Instance Connect
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

Connect using EC2 Instance Connect Endpoint
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
 13.200.229.59

IPv6 address
 -

Username
 Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#) [Connect](#)

```
A newer release of "Amazon Linux" is available.  
Version 2023.6.20241010:  
Run "/usr/bin/dnf check-release-update" for full release and version update info  
Amazon Linux 2023  
https://aws.amazon.com/linux/amazon-linux-2023  
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3  
[ec2-user@ip-172-31-3-5 ~]$ █
```

i-001861022fbcac290 (InstantLibraryApp)
PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

Deployment Using EC2 :

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

Install Software on the EC2 Instance

Install Python3, Flask, and Git

- On Amazon Linux 2:

```
sudo yum update -y  
sudo yum install python3 git  
sudo pip3 install flask boto3
```

- Verify Installations:

```
flask --version  
git --version
```

Clone Your Flask Project from GitHub

git clone: <https://github.com/dhanasree19/Movie-Magic-Smart-Movie-Ticket-Booking-System.git>

Clone your project repository from GitHub into the EC2 instance using Git.

This will download your project to the EC2 instance.

- To navigate to the project directory, run the following command: cd MovieMagic
- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application: sudo flask run --host=0.0.0.0 --port=5000

Verify the Flask app is running: <http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```
Windows PowerShell

Verifying      : perl-Git-2.47.1-1.amzn2023.0.3.noarch          6/8
Verifying      : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64       7/8
Verifying      : perl-lib-0.65-477.amzn2023.0.7.x86_64          8/8

Installed:
git-2.47.1-1.amzn2023.0.3.x86_64           git-core-2.47.1-1.amzn2023.0.3.x86_64          git-core-doc-2.47.1-1.amzn2023.0.3.noarch
perl-Error-1:0.17029-5.amzn2023.0.2.noarch   perl-File-Find-1.37-477.amzn2023.0.7.noarch    perl-Git-2.47.1-1.amzn2023.0.3.noarch
perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64   perl-lib-0.65-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-23-177 ~]$ git clone https://github.com/dhanasree19/Movie-Magic-Smart-Movie-Ticket-Booking-System.git
Cloning into 'Movie-Magic-Smart-Movie-Ticket-Booking-System'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 35 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (35/35), 161.76 KiB | 16.18 MiB/s, done.
Resolving deltas: 100% (5/5), done.
[ec2-user@ip-172-31-23-177 ~]$ cd Movie-Magic-Smart-Movie-Ticket-Booking-System
[ec2-user@ip-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ sudo yum install python3 -y
Last metadata expiration check: 0:07:24 ago on Fri Jul  4 08:29:14 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ sudo yum install python3-pip -y
Last metadata expiration check: 0:10:57 ago on Fri Jul  4 08:29:14 2025.
Dependencies resolved.
=====
Package      Arch      Version       Repository      Size
=====
Installing:
python3-pip   noarch   21.3.1-2.amzn2023.0.11   amazonlinux   1.8 M
Installing weak dependencies:
libCRYPT-compat x86_64  4.4.33-7.amzn2023      amazonlinux   92 k

Transaction Summary
=====
Install 2 Packages

Total download size: 1.9 M
```

```
ec2-user@ip-172-31-23-177: ~ + v
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting zipp>=3.2.0
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3.23.0
[ec2-user@ip-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 MB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.3->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.3 botocore-1.39.3 s3transfer-0.13.0
[ec2-user@ip-172-31-23-177 Movie-Magic-Smart-Movie-Ticket-Booking-System]$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 142-432-607
```

➤ Access the website through: **your-ec2-public-ip**

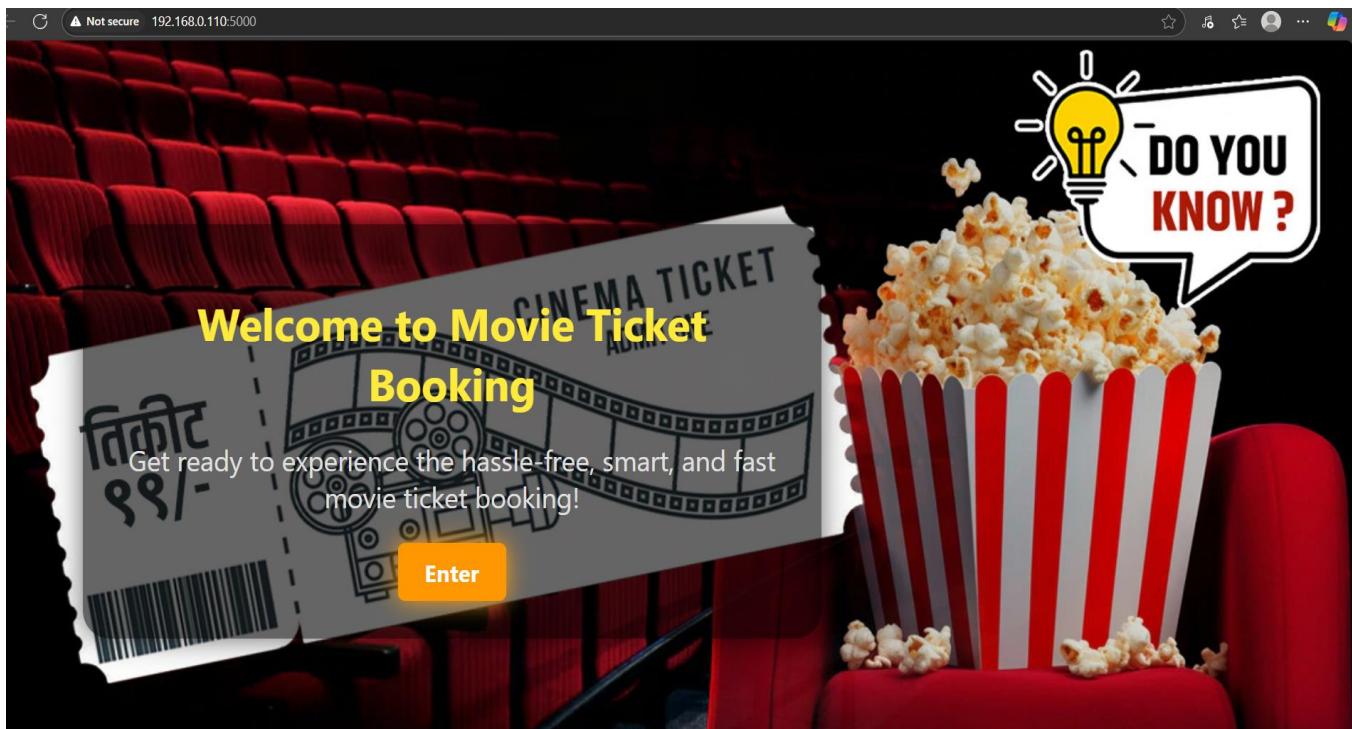
Public IP's: <http://54.197.161.11:5000>

Testing and Deployment

Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional testing to verify the Project

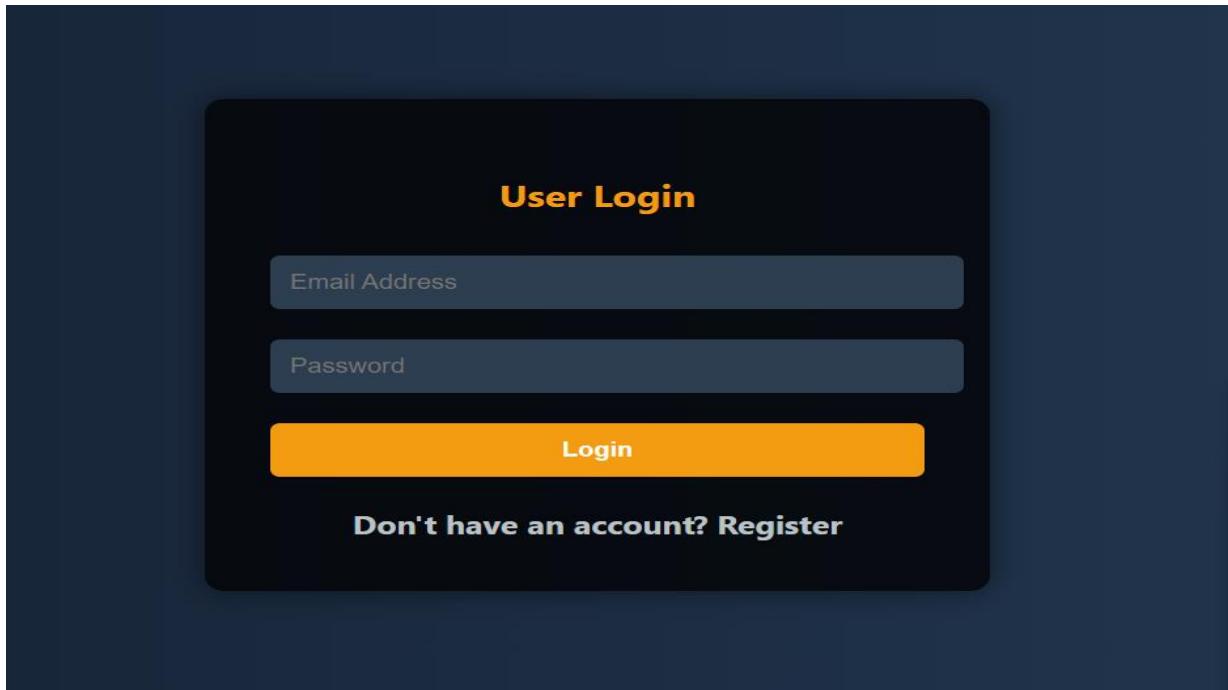
Index Page:



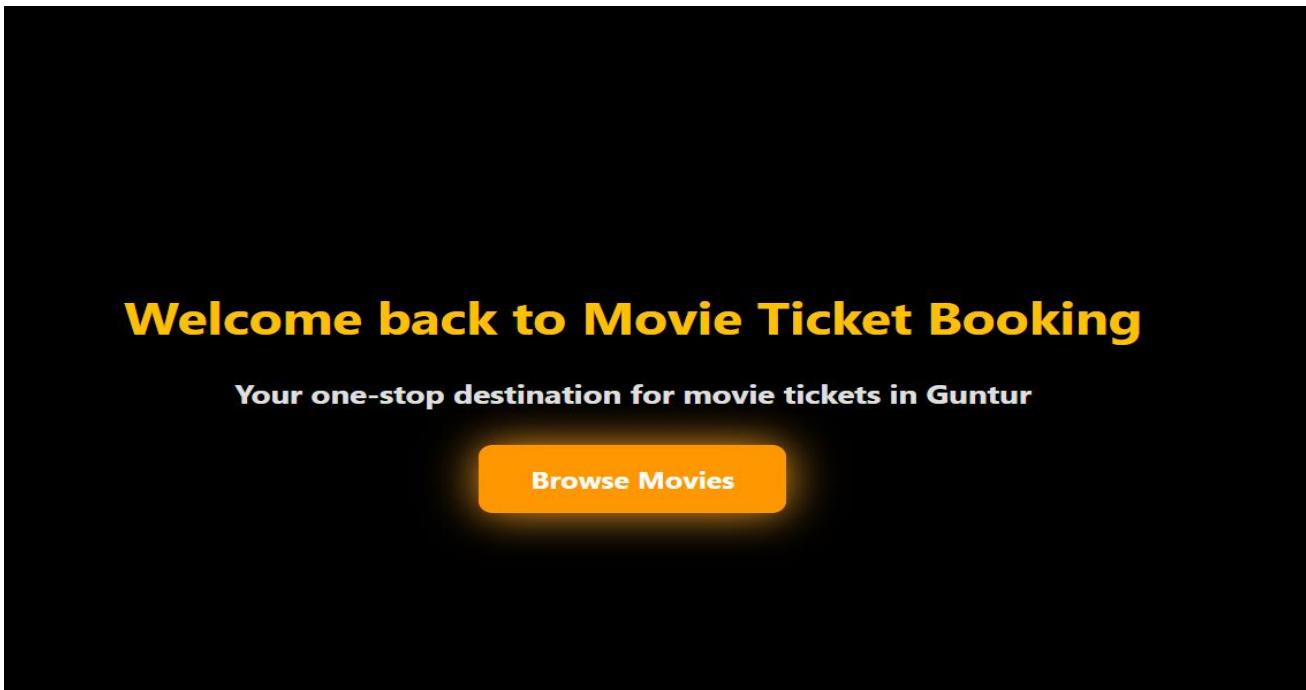
Register Page:

A screenshot of the 'Create an Account' registration form. The form is contained within a dark rectangular box. At the top, the text 'Create an Account' is centered in yellow. Below it are three input fields: 'Full Name' (blue placeholder), 'Email Address' (blue placeholder), and 'Password' (blue placeholder). At the bottom of the form is a large yellow 'Register' button. Below the button, the text 'Already have an account? Login' is displayed in orange.

Login Page:



Home page:



Select a Theater and Screen

Phoenix Mall

Grand Trunk Road, Nagarampalem, Guntur

Screen 1

Kubera (250rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

Screen 2

Salaar (200rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

Screen 3

Lucky Bhaskar (200rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

V Plateno Cinemas

Chandramouli Nagar, Guntur

Screen 1

Kubera (250rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

Screen 2

Daaku Maharaj (250rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

Screen 3

Mad 2 (175rs)

9:00 AM 1:00 PM 5:00 PM 9:00 PM

Select Your Seats

SCREEN

1	2	3	4	33	34	35	36	37	38	81	82	83	84
5	6	7	8	39	40	41	42	43	44	85	86	87	88
9	10	11	12	45	46	47	48	49	50	89	90	91	92
13	14	15	16	51	52	53	54	55	56	93	94	95	96
17	18	19	20	57	58	59	60	61	62	97	98	99	100
21	22	23	24	63	64	65	66	67	68	101	102	103	104
25	26	27	28	69	70	71	72	73	74	105	106	107	108
29	30	31	32	75	76	77	78	79	80	109	110	111	112

Available Booked Selected

Confirm Booking

Payment

Payment Details

₹ {{ seats.split(',')|length }} seat(s) for "{{ movie }}"
Total Amount to Pay: ₹{{ seats.split(',')|length * price|int }}

Select Payment Method:

-- Choose --

Pay Now

Booking Confirmed!

Movie: {{ movie }}
Theater Location: {{ location }}
Show Time: {{ time }}
Seats: {{ seats }}
Price per Ticket: ₹{{ price }}
Total Price: ₹{{ (seats.split(',')|length) * (price|int) }}

Proceed to Payment

Back to Home



Final Conclusion:

Your Flask-based backend system is **successfully set up** and ready to manage the full movie ticket booking flow. Here's a summary of what you've built and achieved:

What the Project Does

- ❖ **User Management:**
 - Users can register and log in.
 - User sessions are handled in memory (basic dictionary for now).
- ❖ **Movie Browsing & Booking:**
 - Theaters and movies are listed for users.
 - Movie details are passed through routes to show available seats.
- ❖ **Seat Selection & Payment:**
 - Users can select seats, view booking summary, and proceed to a mock payment form.
 - Input validation ensures correct card information (as a demo).
- ❖ **Booking Confirmation:**
 - A unique booking ID is generated using uuid.
 - The booking confirmation (with ticket details) is shown.
- ❖ **Modular Code Structure:**
 - Cleanly separated routes and templates.
 - HTML files are stored in templates/ and assets like images in static