# Movie Magic–Smart Movie Ticket Booking System

## Project Description:

With the increasing demand for a seamless and modern movie-watching experience, traditional ticket booking methods often fall short due to long queues, limited availability, and inconsistent service. To address this, the development team introduced Movie Magic—a smart, cloud-based movie ticket booking system. Built using Flask for backend development, hosted on AWS EC2, and integrated with DynamoDB for dynamic data management, the platform allows users to register, log in, and book movie tickets online with ease. Users can search for movies and events based on location, view real-time seat availability, and complete their bookings in just a few clicks. Upon booking, AWS SNS sends instant email notifications confirming ticket details, enhancing user engagement and trust. This cloud-native solution streamlines the entire movie ticketing process, ensuring fast, scalable, and user-friendly access to entertainment for all.

## Scenario 1: Efficient Ticket Booking System for Users

In the Movie Magic System, AWS EC2 provides a reliable infrastructure capable of handling multiple users accessing the platform simultaneously. For example, a user can log in, navigate to the movie selection page, and seamlessly browse available shows and events in their city. They can then select a showtime, pick their preferred seats using an interactive layout, and confirm the booking—all in real-time. Flask manages backend processes, ensuring smooth data flow and quick response times even during high-traffic periods such as weekends or blockbuster releases.

## Scenario 2: Seamless Booking Confirmation Notifications

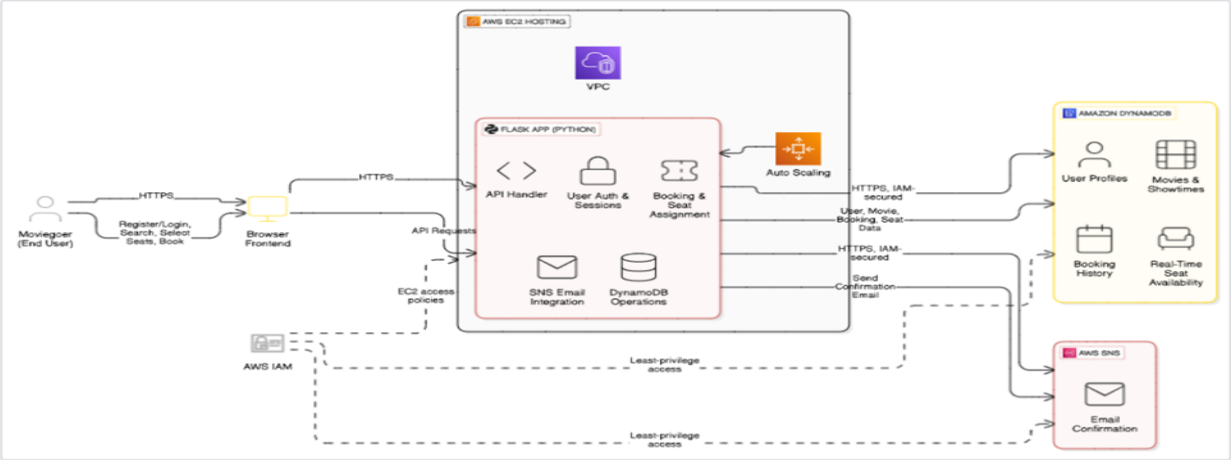
When a user completes a ticket booking, the Movie Magic System leverages AWS SNS to send instant email notifications to confirm the booking. For instance, once the booking is submitted, Flask processes the transaction, and SNS sends a customized email to the

user with all ticket details, including movie name, date, time, and seat numbers. This real-time notification system enhances the customer experience and reduces uncertainty, while DynamoDB securely stores the booking records for both users and admins to manage and track.

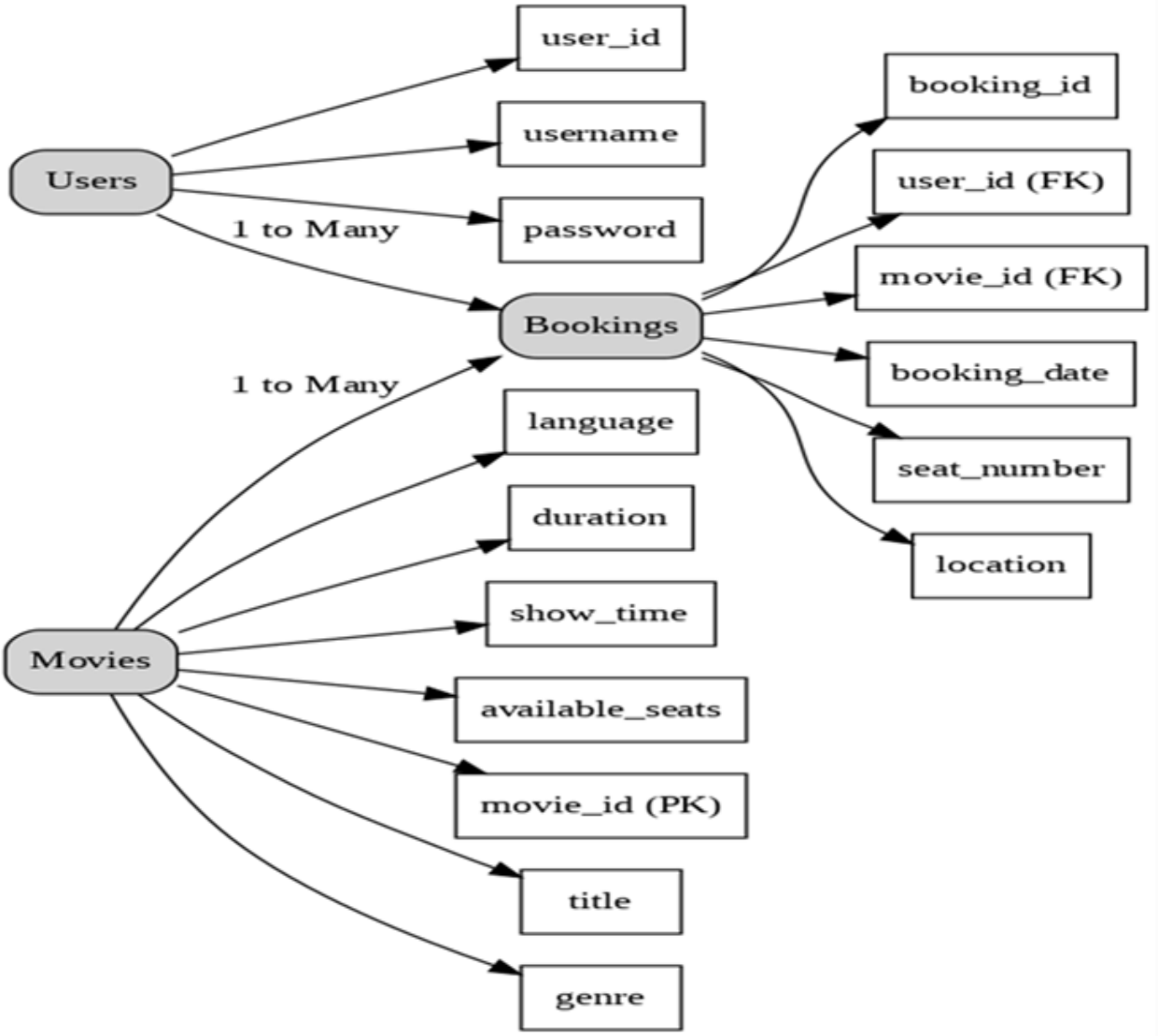
## Scenario 3: Easy Access to Movies and Events

The Movie Magic platform offers users a seamless interface to explore currently running movies and upcoming live events. After logging in, a user can search by location or genre and instantly view listings with showtimes, ratings, and available seats. Flask dynamically fetches this data from DynamoDB, ensuring real-time updates on seat availability and event information. Meanwhile, the EC2-hosted application remains stable and responsive, even during traffic spikes, providing users with an uninterrupted and enjoyable booking experience.

## AWS ARCHITECTURE :



**Entity Relationship (ER)Diagram:**



## Pre-requisites:

1. **AWS Account Setup**: https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html

2. **AWS IAM (Identity and Access Management) :** https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

3. **AWS EC2 (Elastic Compute Cloud) :** https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

4. **AWS DynamoDB** :
5. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

6. **Amazon SNS :**

   https://docs.aws.amazon.com/sns/latest/dg/welcome.html

7. **Git Documentation :** https://git-scm.com/doc

8. **VS Code Installation : (download the VS Code using the below link or you can get that in Microsoft store):** https://code.visualstudio.com/download

# Project WorkFlow:

## 1. AWS AccountSetup and Login :

**Activity 1.1:** Set up an AWS accountif 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 requestnotifications.

**Activity 3.2**: Subscribe users and librarystaff to SNS email notifications.

## 4. Backend Development and Application Setup :

**Activity 4.1**:Develop the BackendUsing 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 instanceto host the Flask application.

**Activity 6.2**: Configure securitygroups for HTTP, and SSH access.

## 7. Deployment on EC2 :

**Activity 7.1**:Upload Flask Files

**Activity 7.2**: Run the Flask App

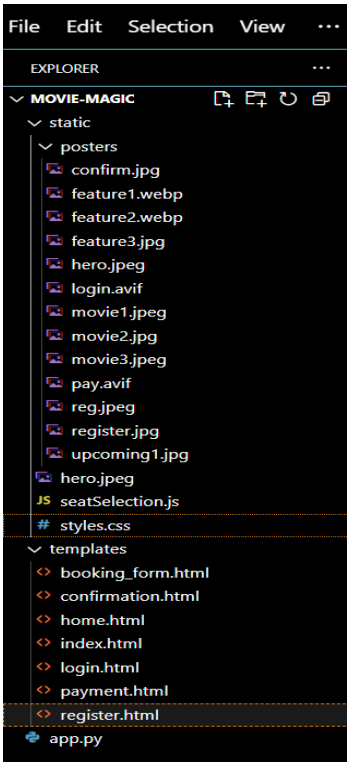
## 8. Testing and Deployment :

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

# Milestone 1: Web Application Development and Setup

- **Local Deployment :**

File Explorer Structure




## Description of the code :

- **Flask App Initialization**

  **Imports and Configuration:**

app.py > ...

```
from flask import Flask, render_template, request, redirect, session, flash, url_for
import hashlib
import uuid
import boto3
```

**Description**: This project uses Flask for routing, session management, and user authentication with secure password hashing. It integrates AWS services via Boto3 for handling data storage, notifications, and unique user operations.

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

**Description:**A new Flask application instance is initialized, and a secret key is set to securely manage user sessions and protect against cookie tampering.

- **Dynamodb and SNS Setup :**

```
app = Flask(__name__)
app.secret_key = 'super-secret-key'
dynamodb = boto3.resource('dynamodb',region_name='us-east-1') # e.g., 'ap-south-1'

users_table = dynamodb. Table("user")

bookings_table = dynamodb.Table("moviebooking")
```

**Description**: Use **boto3** to connect to **DynamoDB** for handling user registration, movie bookings database operations and also mention region_name where Dynamodb tables are created.

- **SNS Connection :**

**Description:** Configure **SNS** to send notifications when a movie ticket is booked. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

```
sns = boto3.client('sns', region_name='us-east-1')

sns_topic_arn = 'arn:aws:sns:us-east-1:216989138822:MovieTicketNotifications.fifo'
```

- **Function to send the Notifications:**

**Description:**This function sends a booking confirmation email using AWS SNS. It formats the booking details into a message and publishes it to a specified SNS topic, notifying the user via email about their successful movie ticket booking.

```
# -------- Helper Functions --------
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def send_mock_email(email, booking_info):
    print(f"[MOCK EMAIL] Sent to {email}:\nBooking confirmed for {booking_info['movie']}\n"
          f"Seat: {booking_info['seat']}, Date: {booking_info['date']}, Time: {booking_info['time']}\n"
          f"Booking ID: {booking_info['id']}\n")
```

- **Routes for Web Pages**

**Register User:** Collecting registration data, hashes the password, and stores user details in the database.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        if email in mock_users:
            flash("Account already exists.")
            return redirect(url_for('login'))

        mock_users[email] = hash_password(password)
        flash("Account created! Please login.")
        return redirect(url_for('login'))
    return render_template('register.html')
```

- **login Route (GET/POST)**: Verifies user credentials, increments login count, and redirects to the dashboard on success

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        hashed = hash_password(password)

        print("Submitted email:", email)
        print("Submitted hash:", hashed)
        print("Registered users:", mock_users)

        if email in mock_users and mock_users[email] == hashed:
            session['user'] = email
            print("Login successful!")
            return redirect(url_for('home'))
        else:
            flash("Invalid email or password.")
            print("Login failed.")
            return render_template('login.html')

    return render_template('login.html')
```

These Flask routes handle key navigation in the app: /logout logs out the user by clearing the session and showing a flash message; /home1 is a protected route accessible only to logged-in users.

```python
@app.route('/logout')
def logout():
    session.clear()
    flash("You have been logged out.")
    return redirect(url_for('index'))
```

```python
@app.route('/home')
def home():
    if 'user' not in session:
        return redirect(url_for('login'))

    now_showing = [
        {"title": "The Grand Premiere", "genre": "Drama", "poster": "posters/movie1.jpeg", "duration": "2h 10m", "rating": "4.5", "synopsis": "A hea
        {"title": "Engaging", "genre": "Drama", "poster": "posters/movie2.jpg", "duration": "1h 45m", "rating": "4.2", "synopsis": "A hilarious ride
    ]
    coming_soon = [
        {"title": "Future Flick", "genre": "Sci-Fi", "poster": "posters/upcoming1.jpg", "duration": "2h 20m", "rating": "N/A", "synopsis": "A mind-b
    ]
    top_rated = [
        {"title": "Edge of Tomorrow", "genre": "Action", "poster": "posters/movie3.jpeg", "duration": "2h", "rating": "4.8", "synopsis": "A soldier
    ]
    return render_template('home.html', now_showing=now_showing, coming_soon=coming_soon, top_rated=top_rated)
```

- **Booking Page Route:**

**Description:** This route displays the booking page (b1.html) with movie, theater, address, and

price details passed as query parameters. It ensures only logged-in users can access the page.

```python
@app.route('/booking', methods=['GET', 'POST'])
def booking():
    if 'user' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Temporarily store booking form in session
        session['pending_booking'] = {
            'movie': 'Example Movie',
            'seat': request.form['seat'],
            'date': request.form['date'],
            'time': request.form['time']
        }
        return redirect(url_for('payment'))

    return render_template('booking_form.html', movie='Example Movie')
```

- **Tickets Page Route:**

**Description:** This route processes movie ticket bookings by collecting form data, generating a unique booking ID, storing details in DynamoDB, and sending a confirmation email via AWS SNS. It then displays the booking details on the tickets page.

```python
@app.route('/payment', methods=['GET', 'POST'])
def payment():
    if 'user' not in session or 'pending_booking' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Pretend to process the fake payment
        booking_info = session['pending_booking']
        booking_info['user'] = session['user']
        booking_info['id'] = str(uuid.uuid4())[:8]

        mock_bookings.append(booking_info)
        session['last_booking'] = booking_info
        send_mock_email(session['user'], booking_info)
        session.pop('pending_booking', None)
        flash("Payment successful. Ticket booked!")

        return redirect(url_for('confirmation'))

    return render_template('payment.html')
```

- **Confirmation page for movie ticket booking :**

```python
@app.route('/confirmation')
def confirmation():
    if 'user' not in session or 'last_booking' not in session:
        return redirect(url_for('login'))

    booking = session['last_booking']
    return render_template('confirmation.html', booking=booking)
```

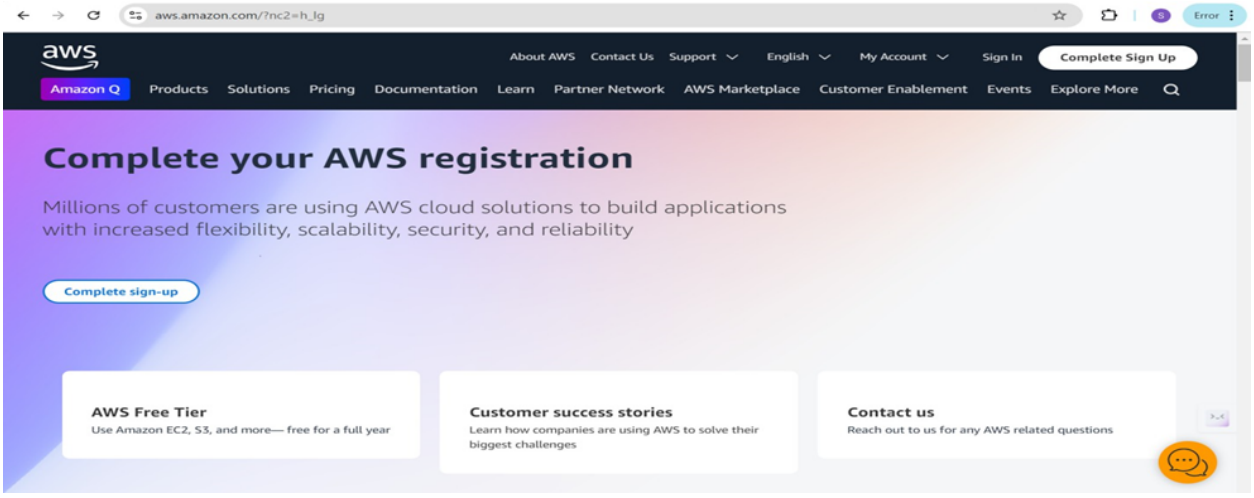
**Application Entry point:**

```python
if __name__ == '__main__':
    print("🚀 Mock MovieMagic running at http://127.0.0.1:5000")
    app.run(debug=True)
```

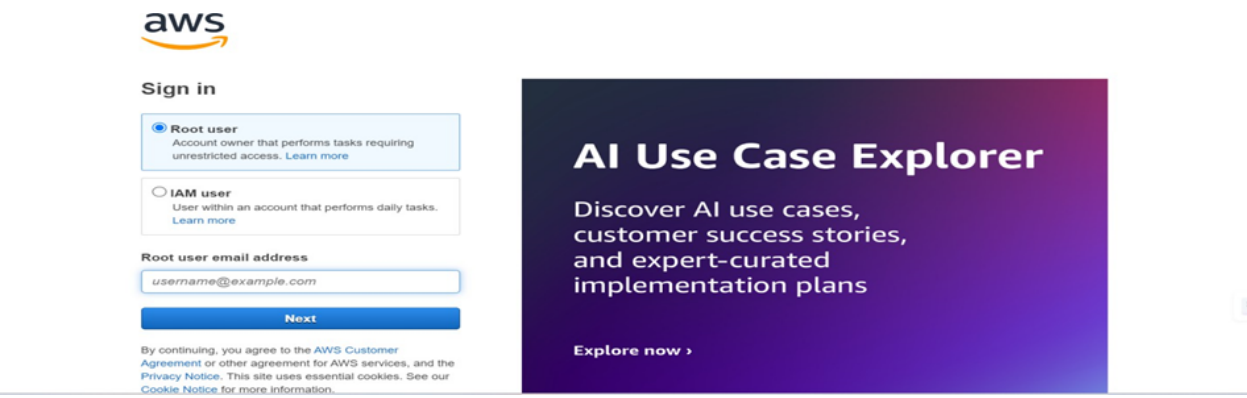
**Description**: This block starts the Flask application using the built-in development server, setting the host, port, and enabling debug mode for easier development and testing.

## Milestone 2: AWS AccountSetup and Login :

- **Activity 1.1: Set up an AWS account if not already done.**
  - ➤ Sign up for an AWS accountand configure billingsettings.

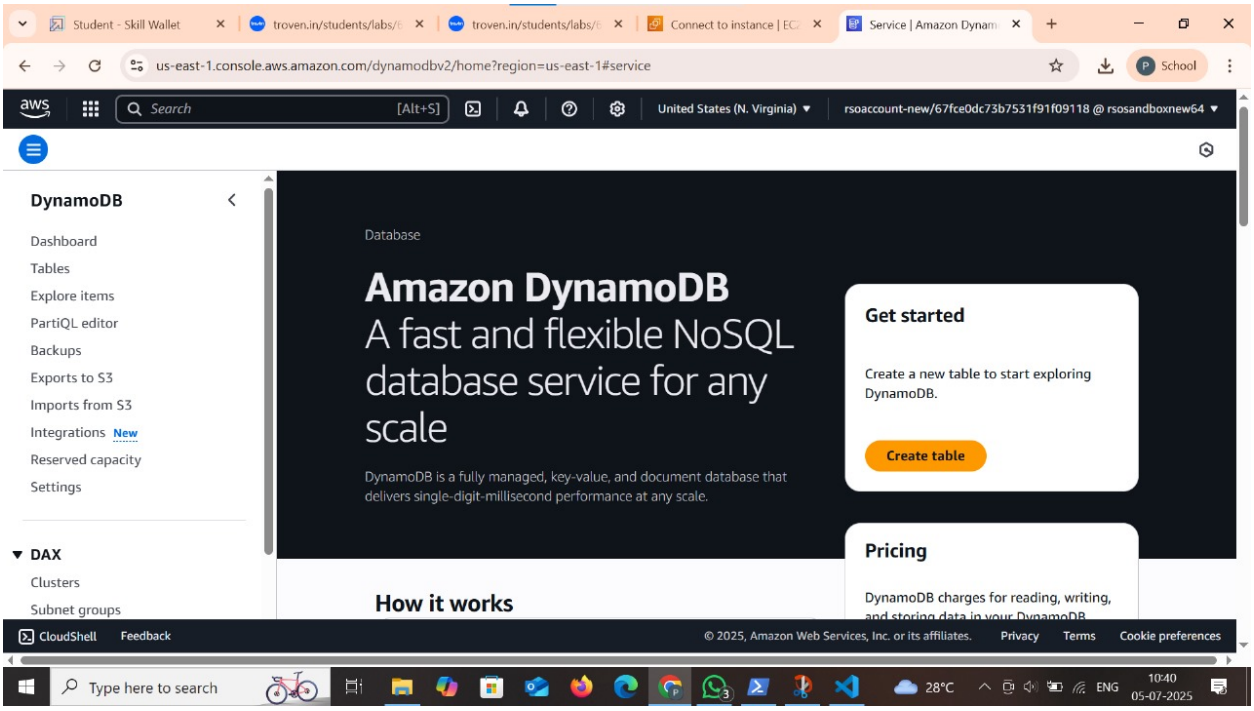


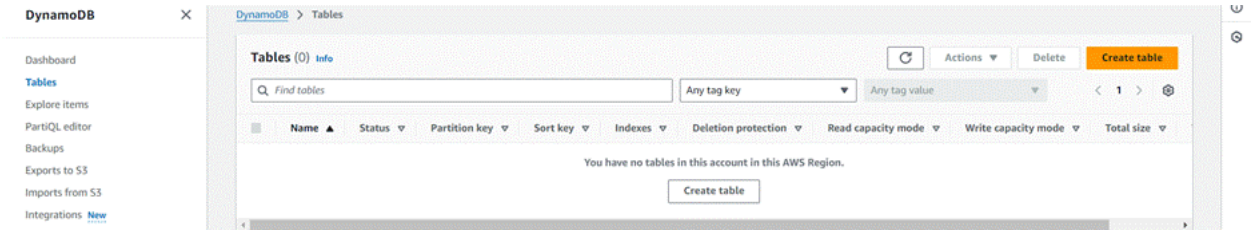
- **Activity 1.2: Log in to the AWS Management Console :**
    - ➤ After setting up your account, log in to the AWS Management Console.



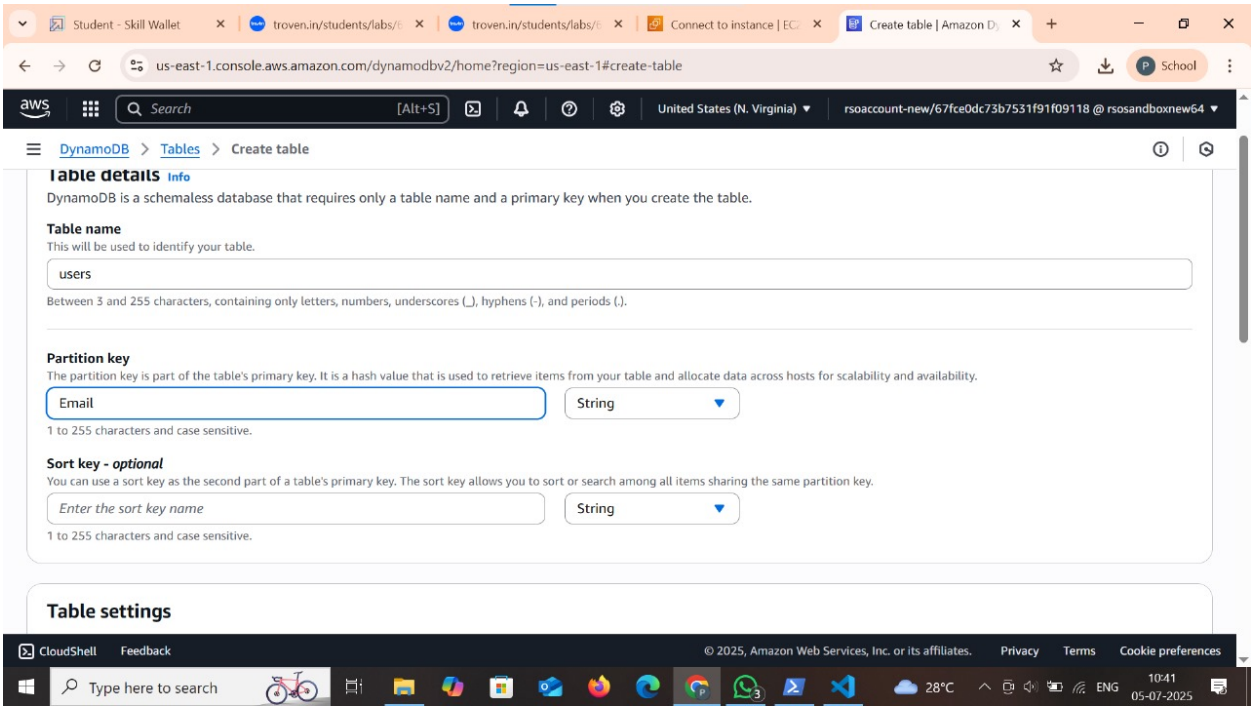

# Milestone 3: DynamoDBDatabase Creation and Setup

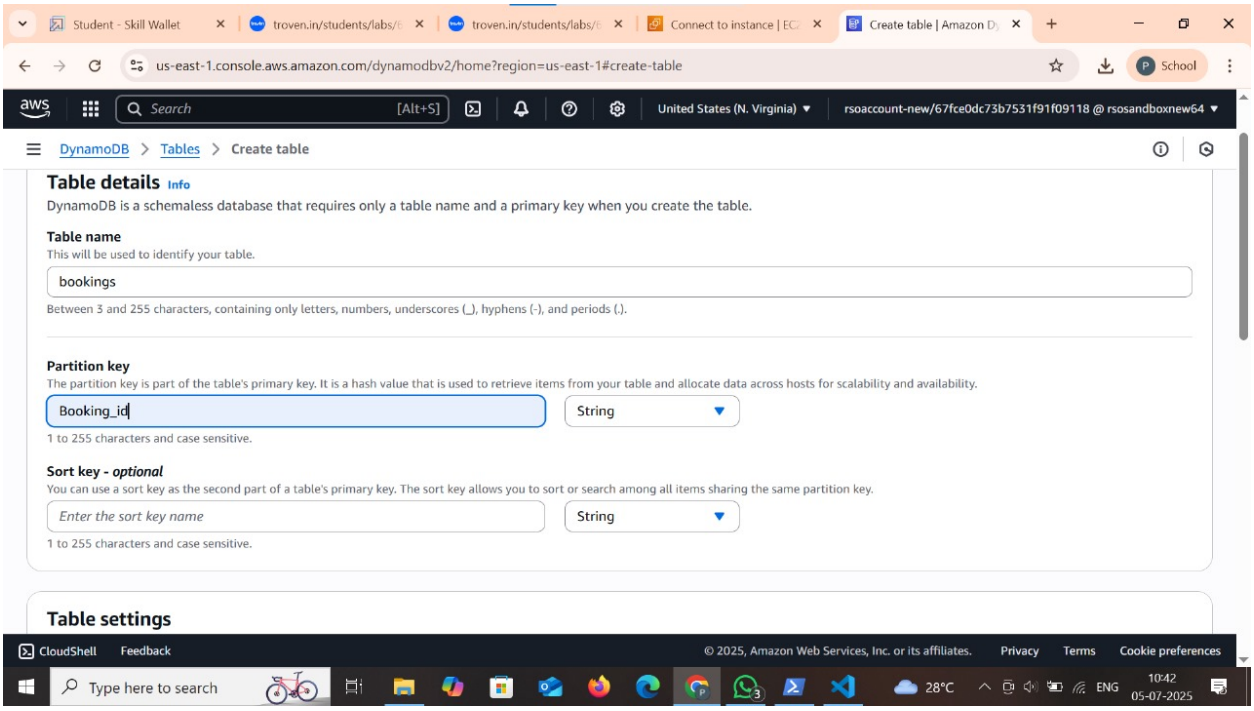
- **Activity 3.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.

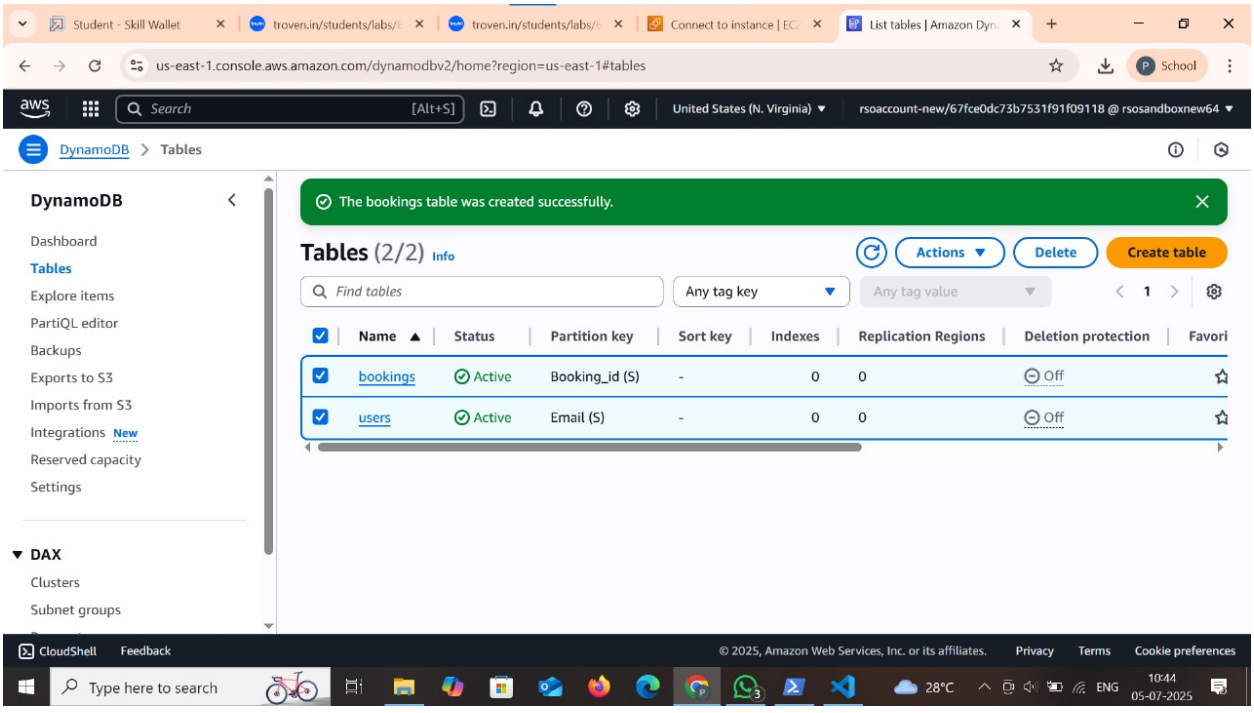





- **Activity 3.2:Create a DynamoDBtable for storingregistration details and book requests.**



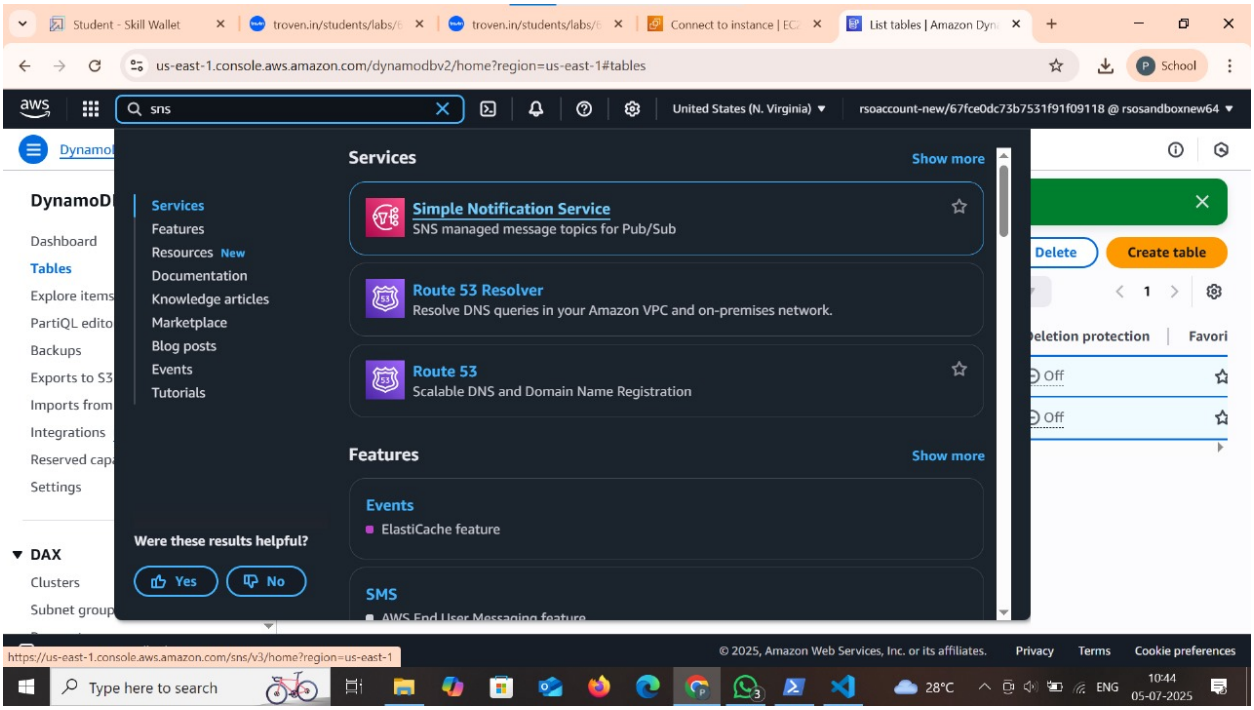

Create table | Amazon D
us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table
United States (N. Virginia)
rsoaccount-new/67fce0dc73b7531f91f09118 @ rsosandboxnew64
DynamoDB > Tables > Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
bookings
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.
Booking_id
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
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C
ENG
10:42
05-07-2025

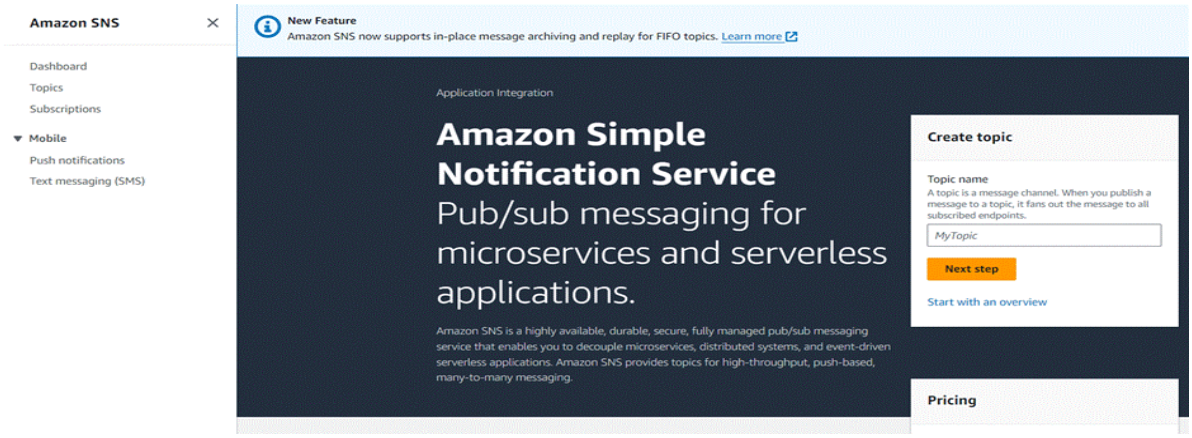

List tables | Amazon Dyn
us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables
United States (N. Virginia)
rsoaccount-new/67fce0dc73b7531f91f09118 @ rsosandboxnew64
DynamoDB > Tables
DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
The bookings table was created successfully.
Tables (2/2) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
Favori
bookings
Active
Booking_id (S)
-
0
0
Off
users
Active
Email (S)
-
0
0
Off
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C
ENG
10:44
05-07-2025

# Milestone 4 : SNS Notification Setup :

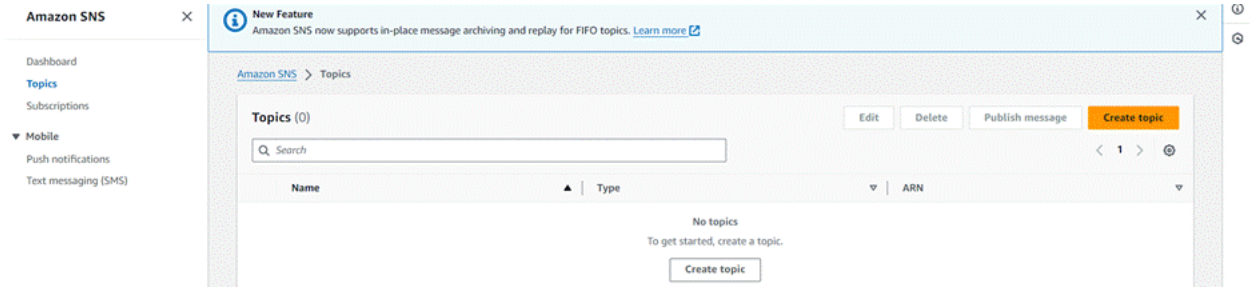
- **Activity 4.1: Create SNS topics for sending email notifications to users and library staff.**

- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

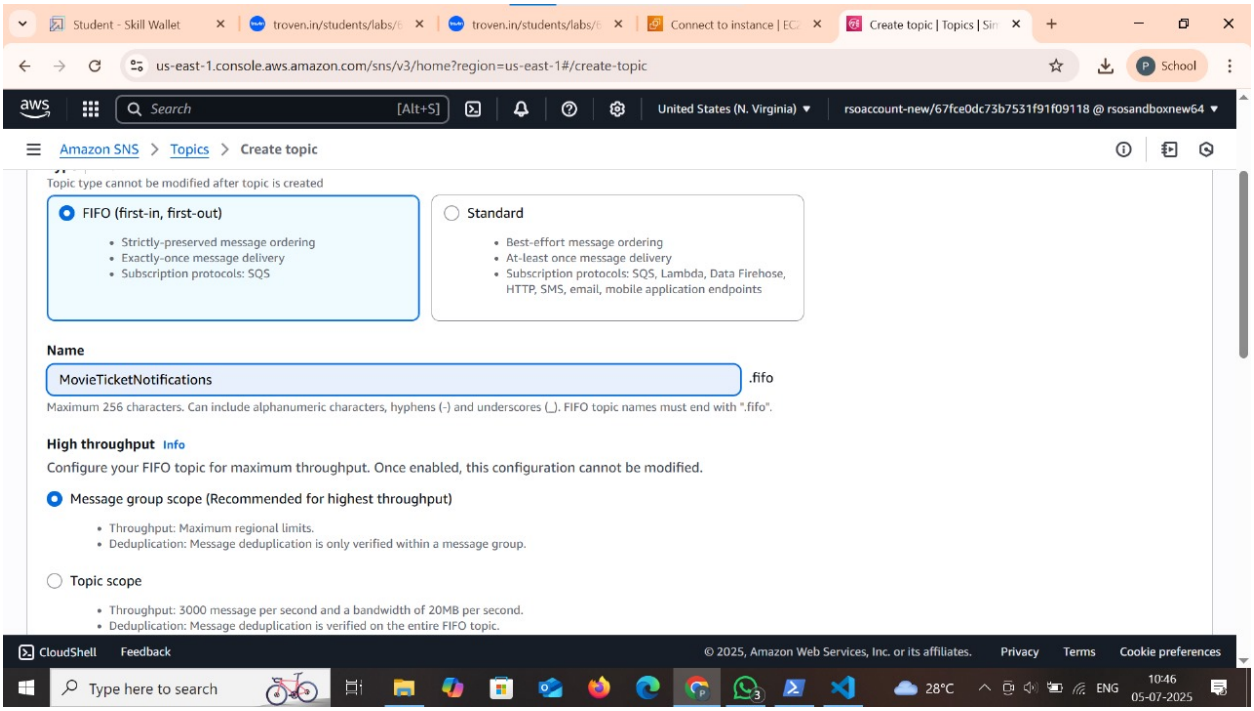


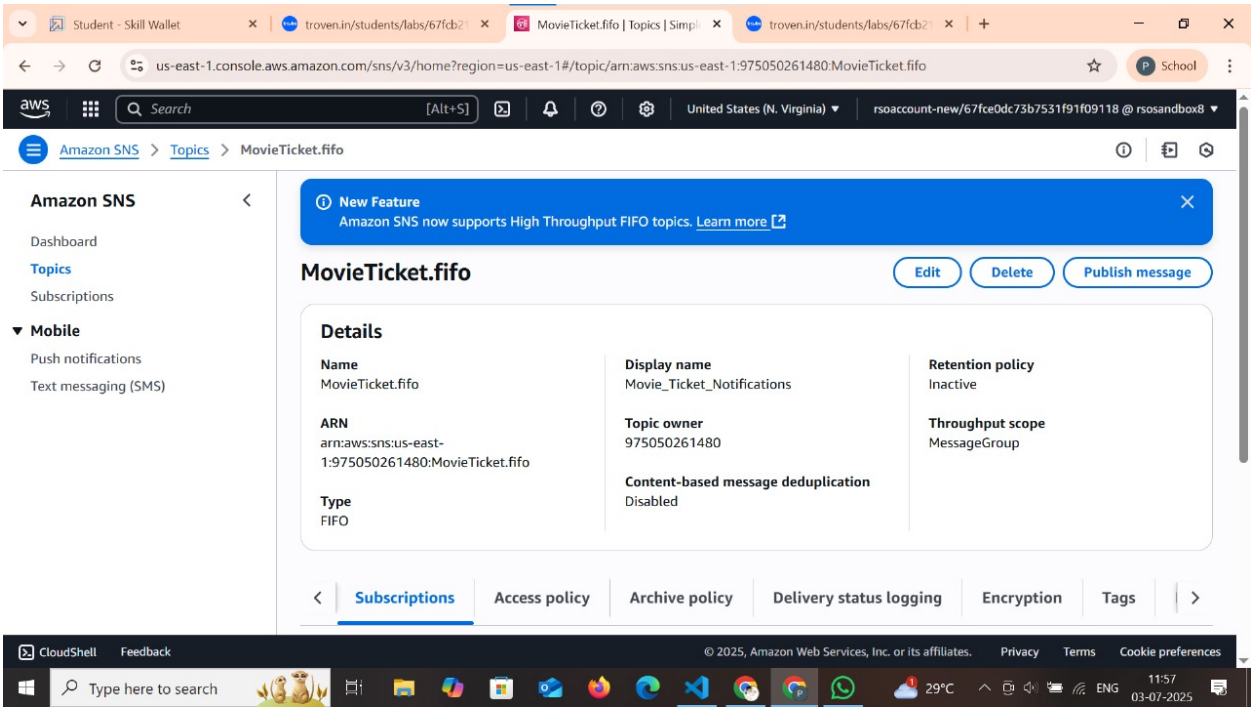

- Click on Create Topic and choose a name for the topic.




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

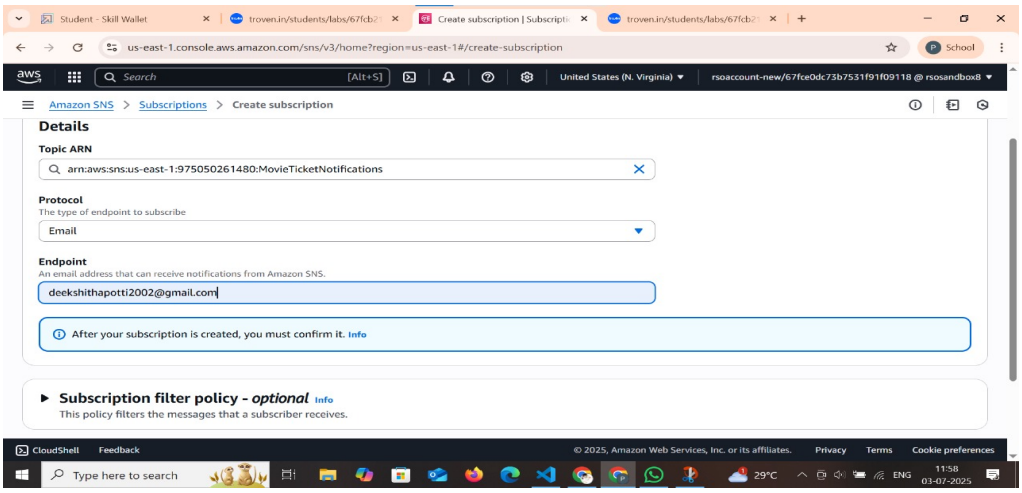



- Configure the SNS topic and note down the Topic ARN.

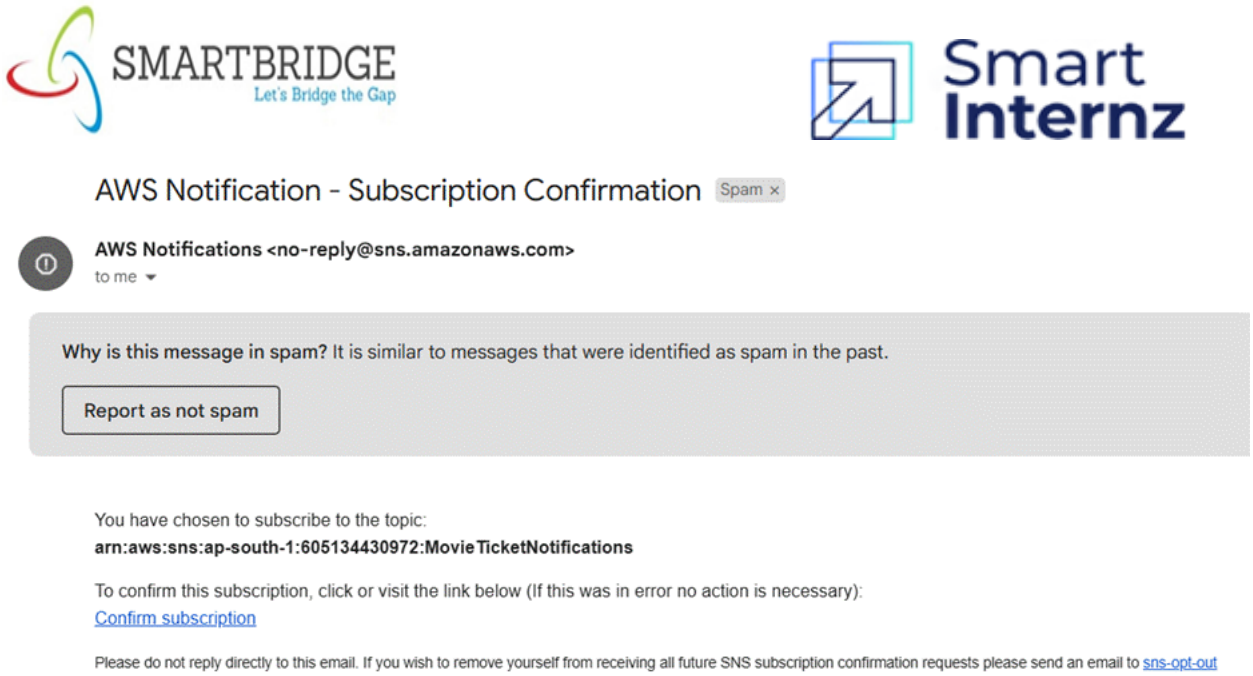



- **Subscribe Users And Admin**

Subscribe users (or admin staff) to this topic via Email. When a movie ticket is booked, notifications will be sent to the user's emails.



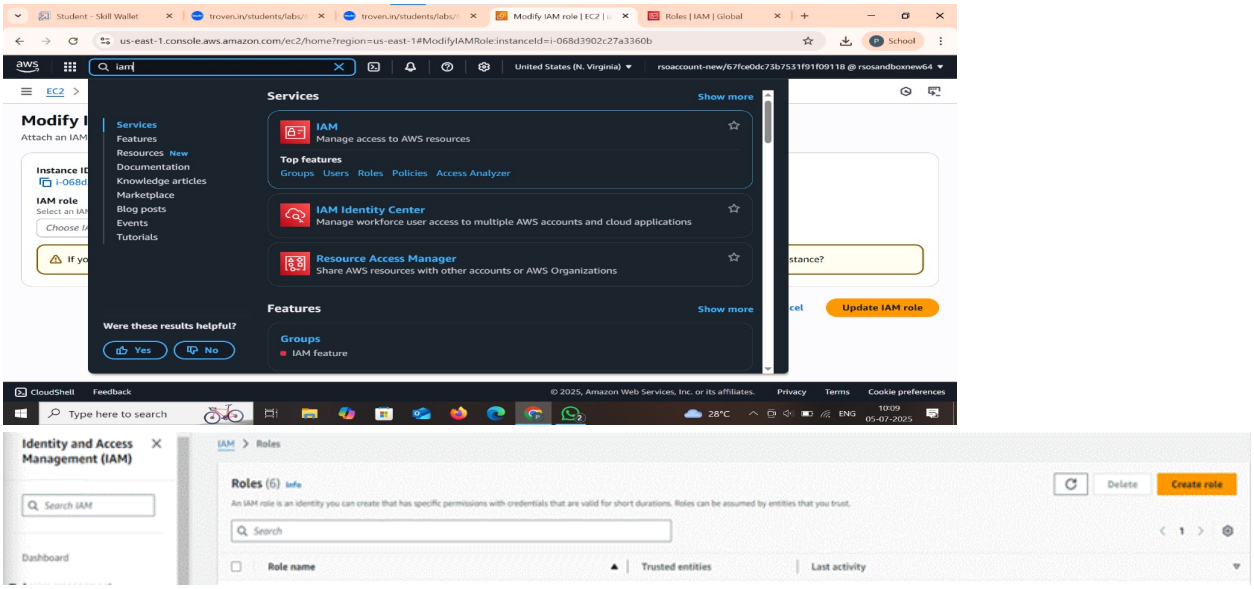

- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

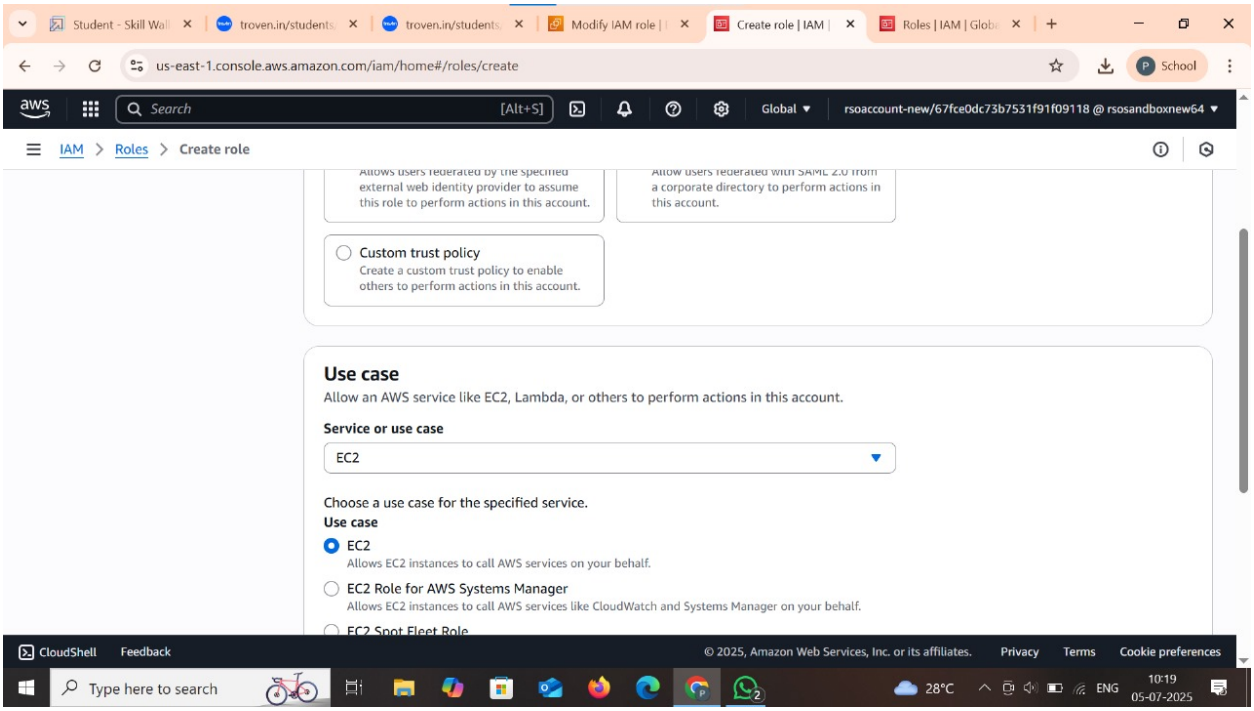



- Successfully done with the SNS mail subscription and setup, now store the ARN link.

# Milestone 5 : IAM Role Setup

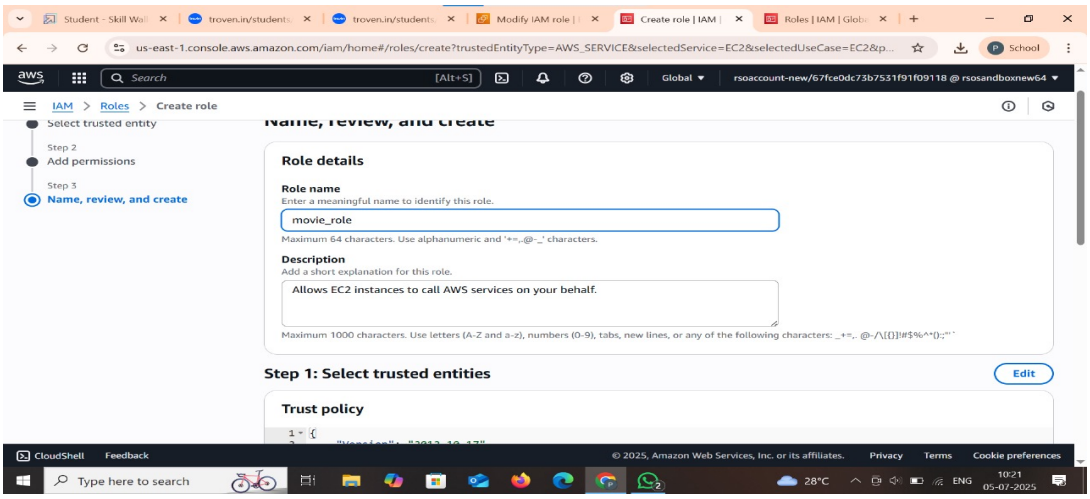
- **Activity 5.1:Create IAM Role.**
    - In the AWS Console,go to IAM and createa new IAM Role for EC2 to interact with DynamoDB and SNS.



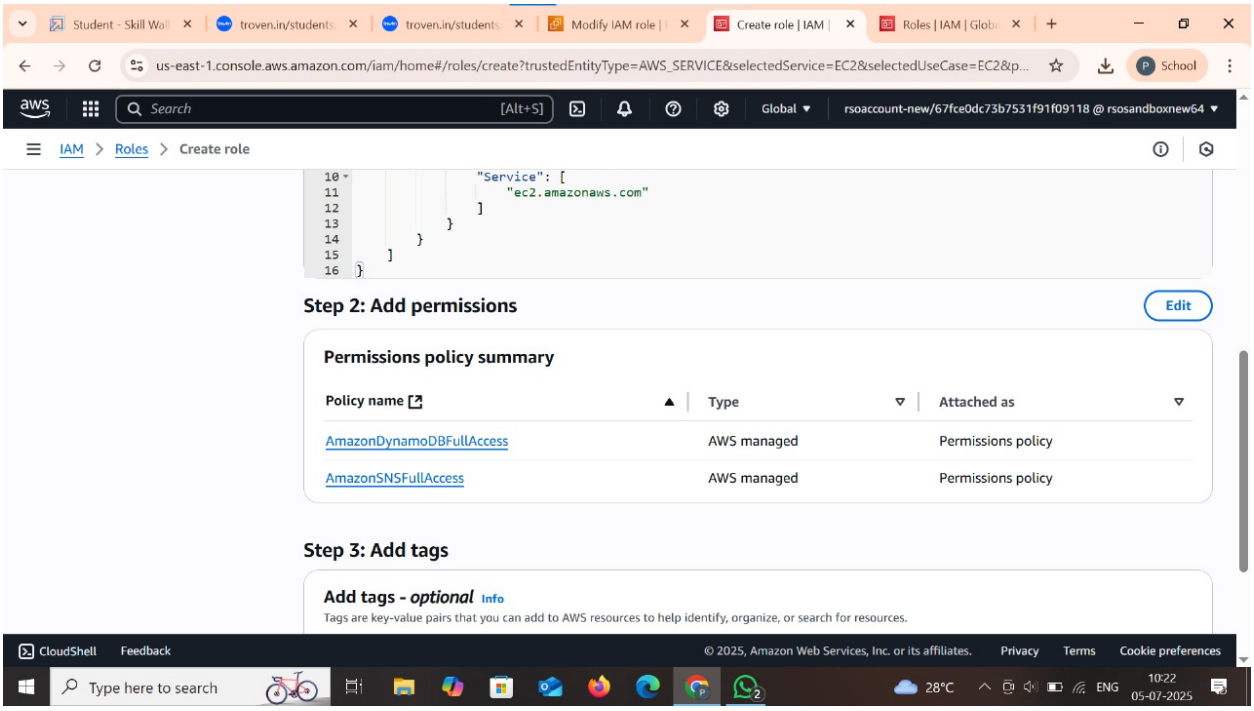



- **Activity 5.2: AttachPolicies.**

Attach the following policiesto the role:

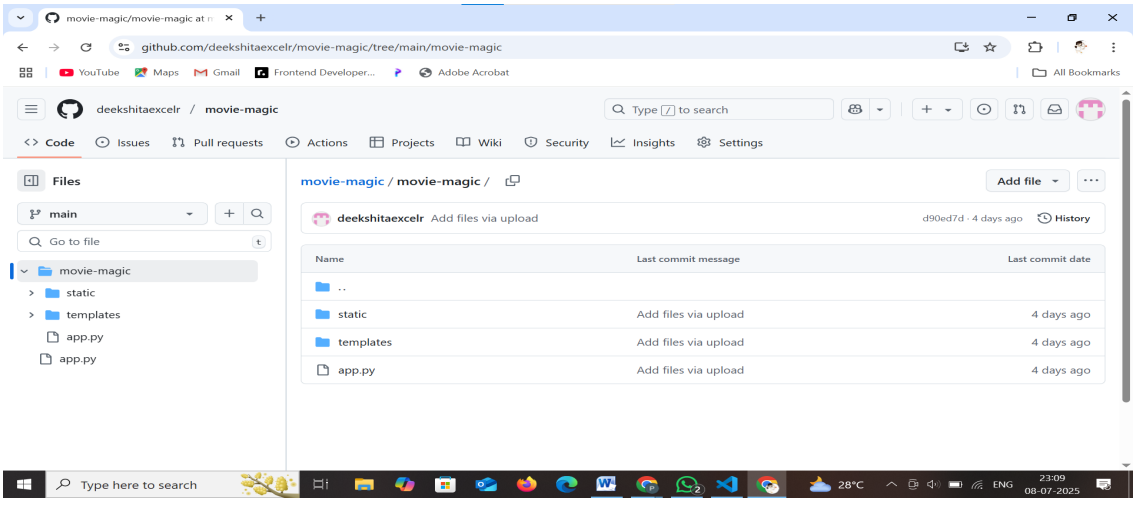


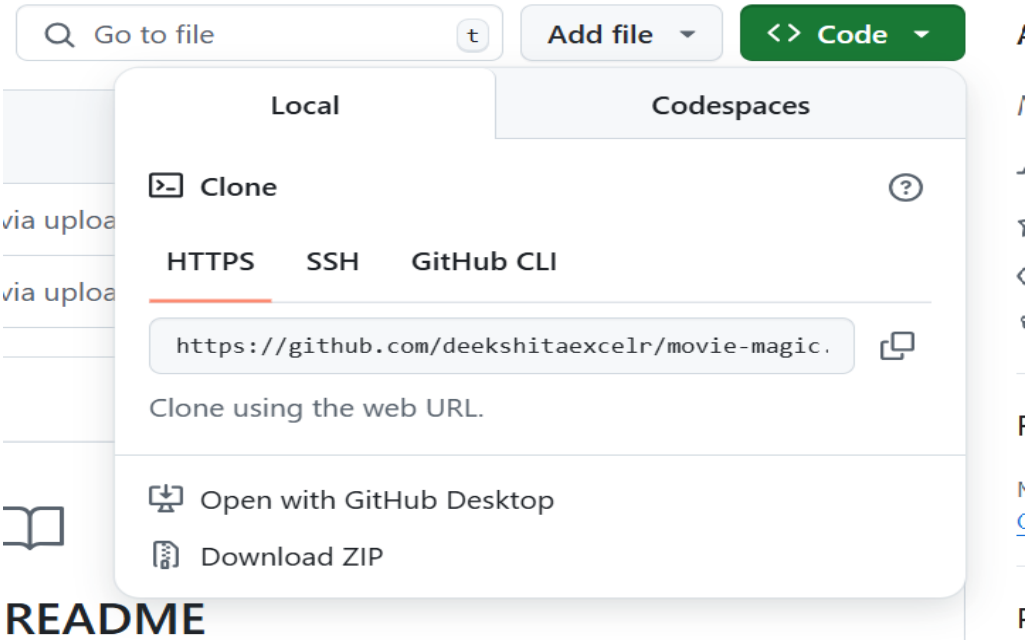

- ✓ AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- ✓ AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.




# Milestone 6: EC2 InstanceSetup :

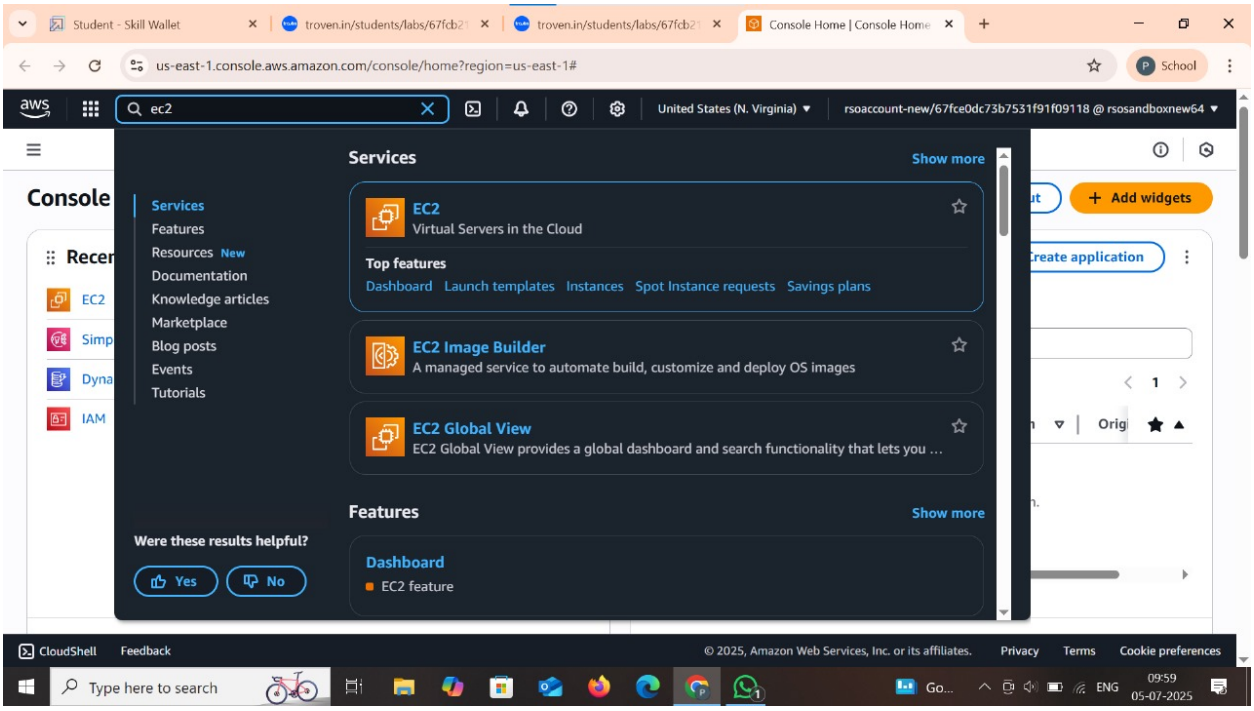
- **Note: Load your Flask app and Html files into GitHub repository.**



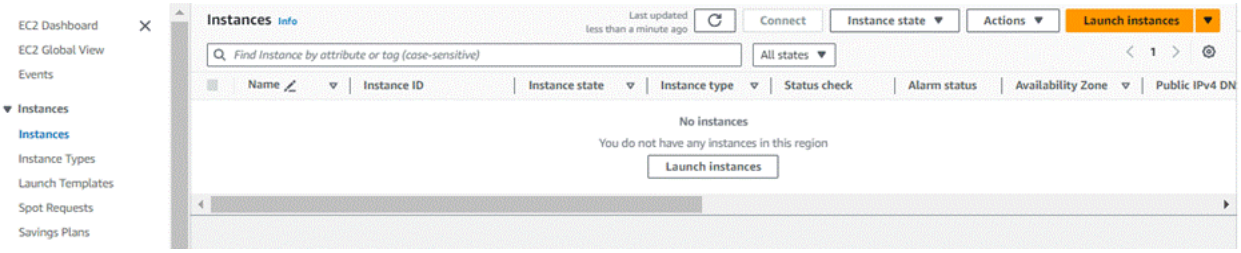


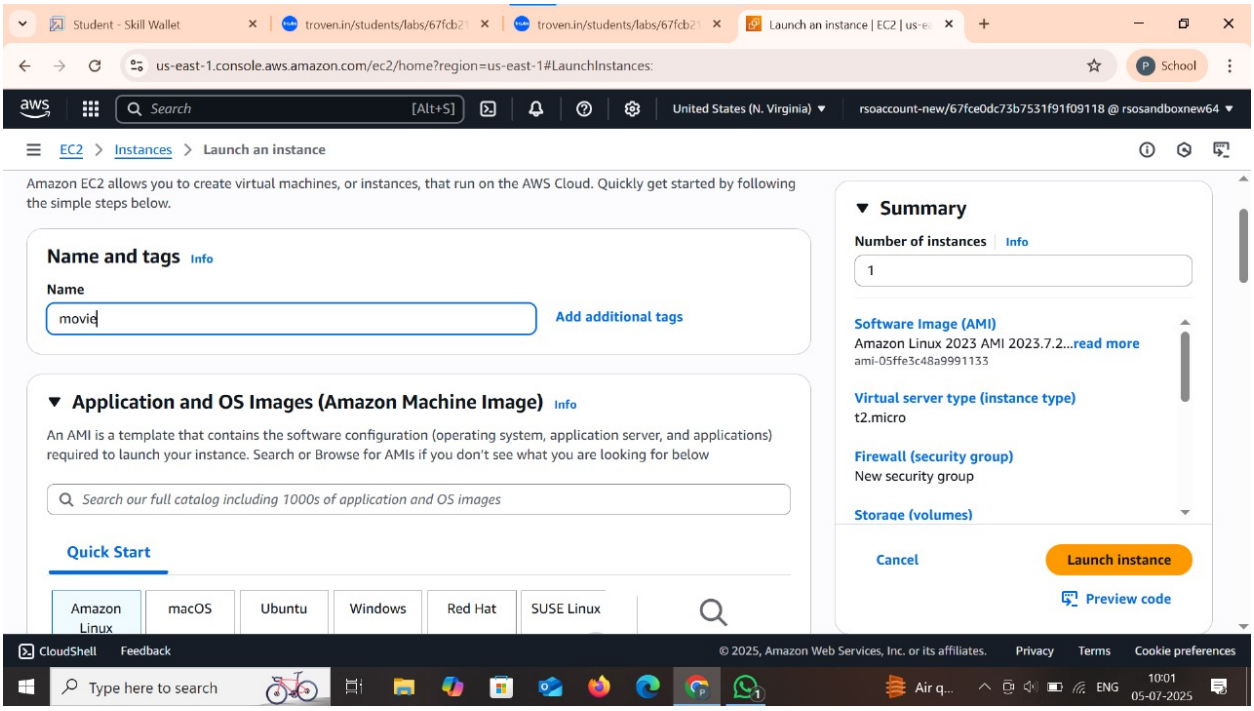

- **Launch an EC2 instance to host the Flask**

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

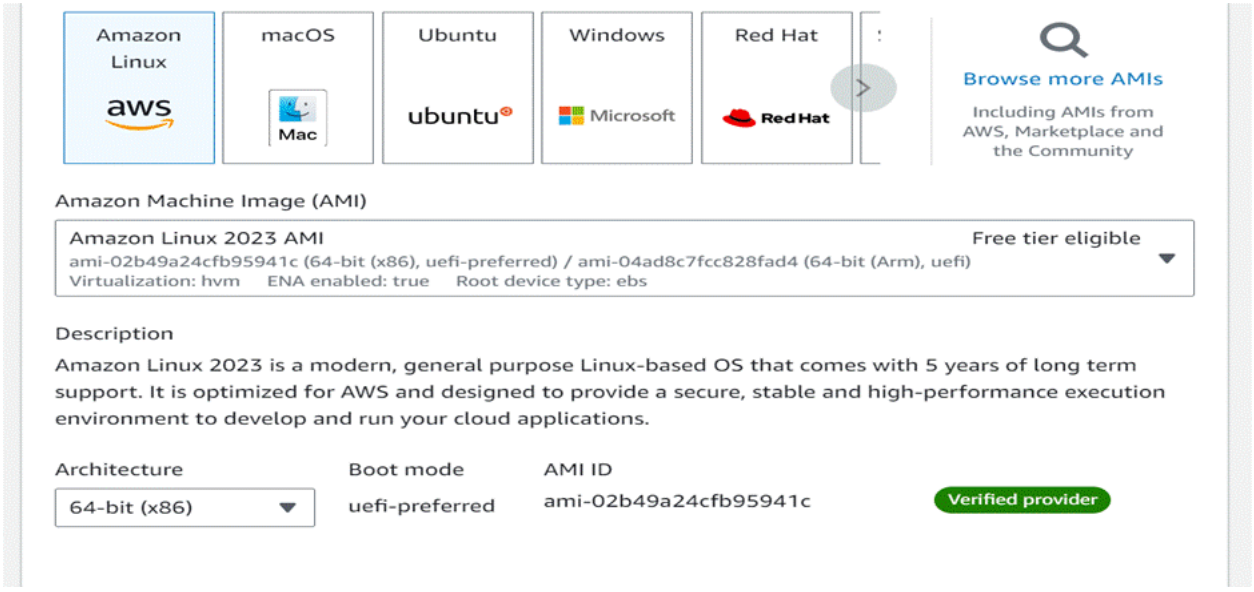



- Click on Launch instance to launch EC2 instance

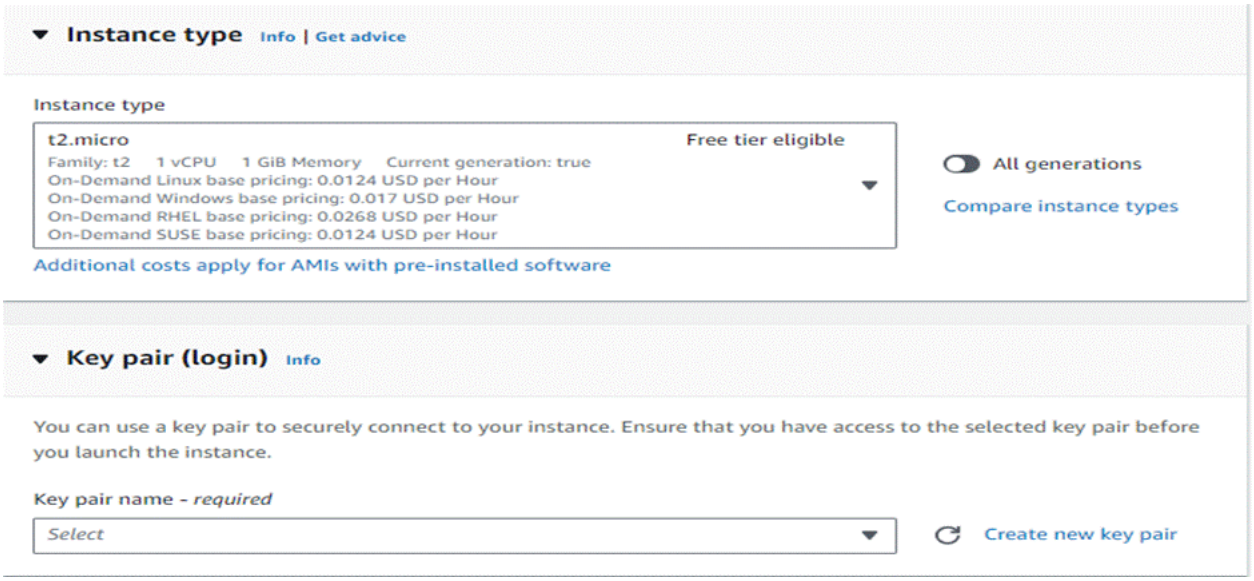


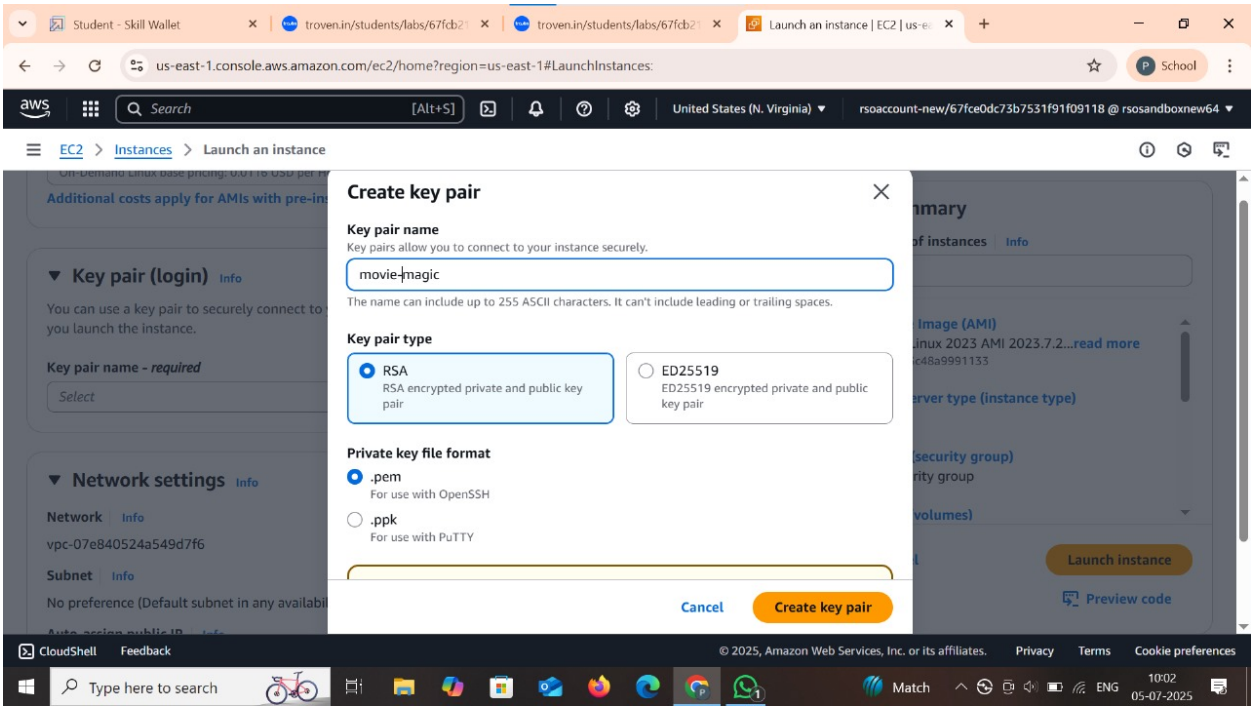


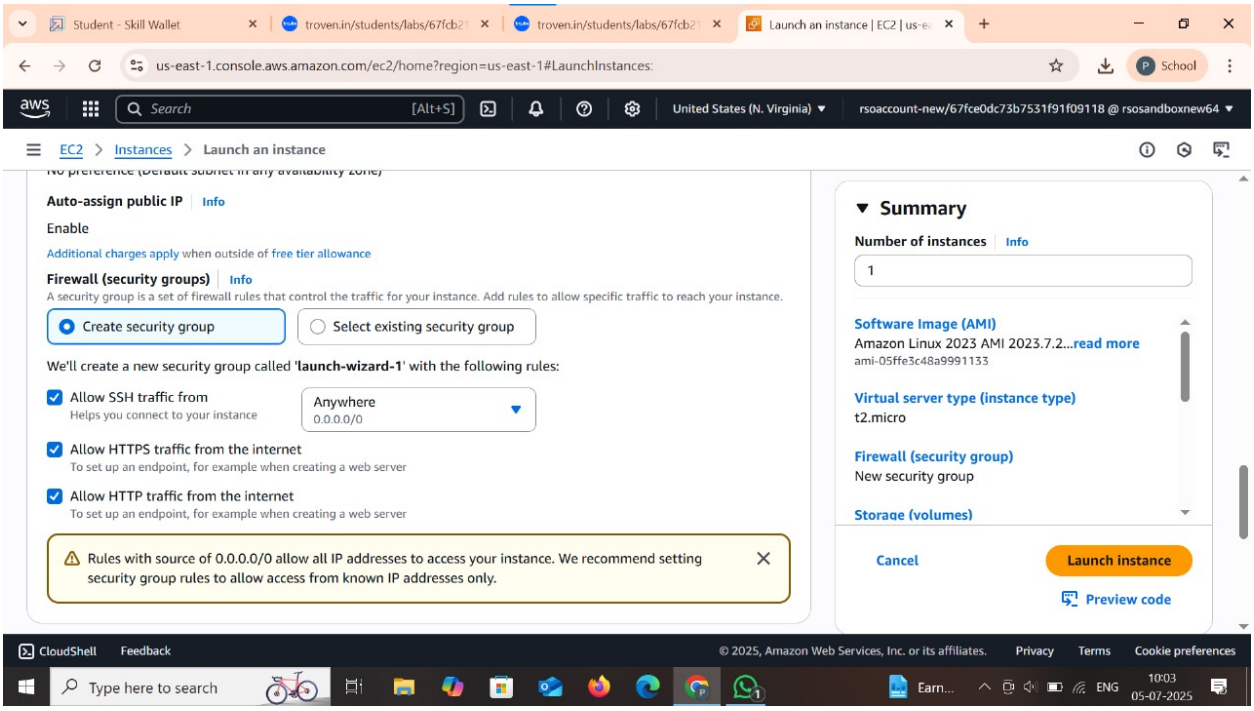

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



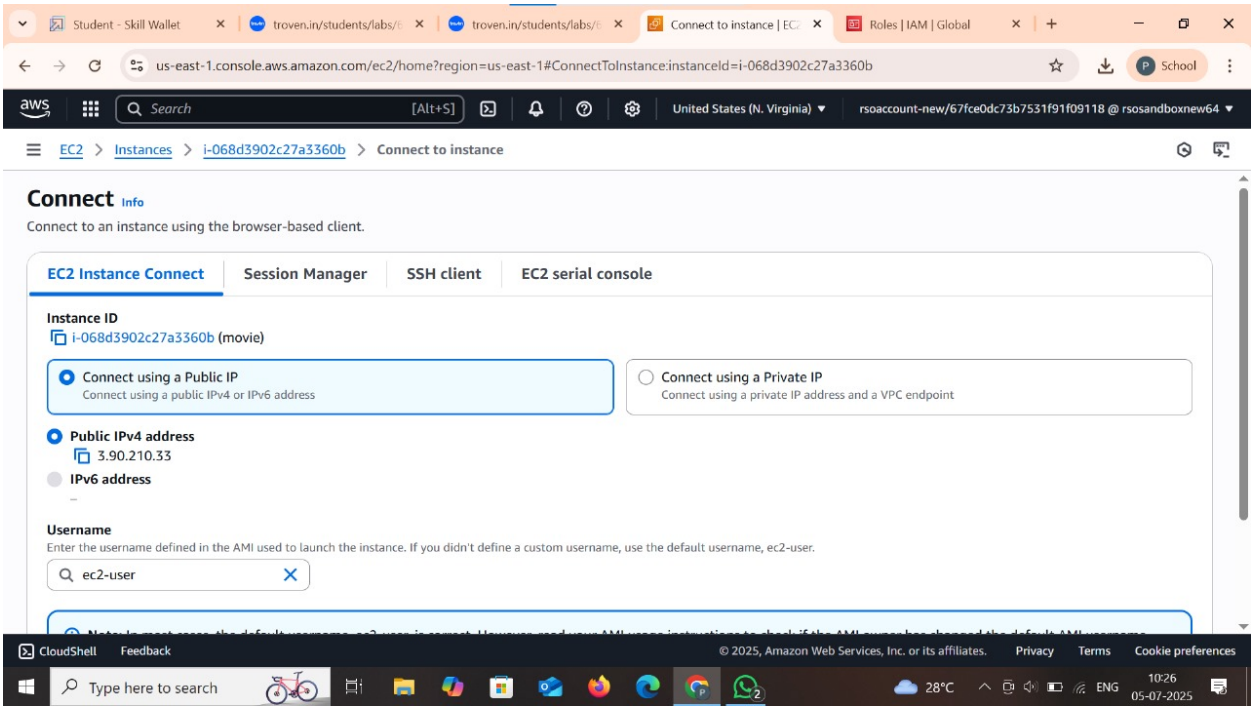
- Create and download the key pair for Server access.



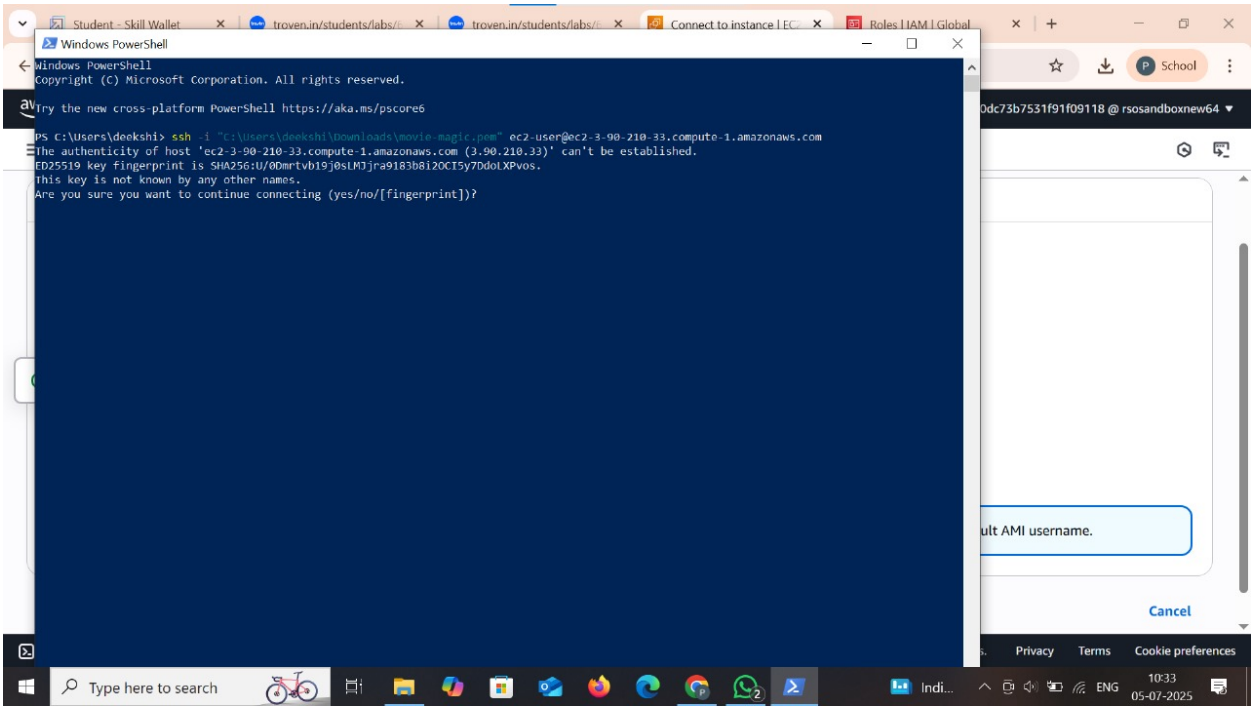



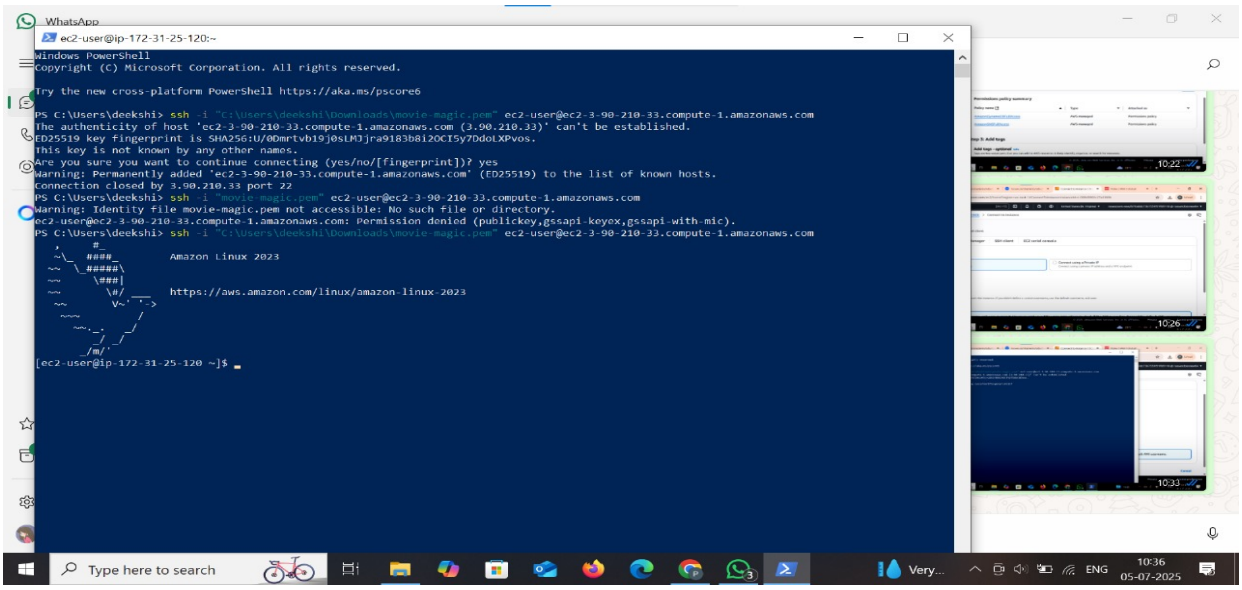



- **Configure security groups for HTTP and SSH for client:**



- Now open windows posershell and navigate to the key pair download pem file path and type the command  as "ssh -i "path of pem file" ec2-user@copy the 4 th point in that above image"



# Milestone 7: Deployment on EC2

## Activity 7.1: Install Softwareon the EC2 Instance

Install Python3, Flask, and Git

- **On Amazon Linux 2:**

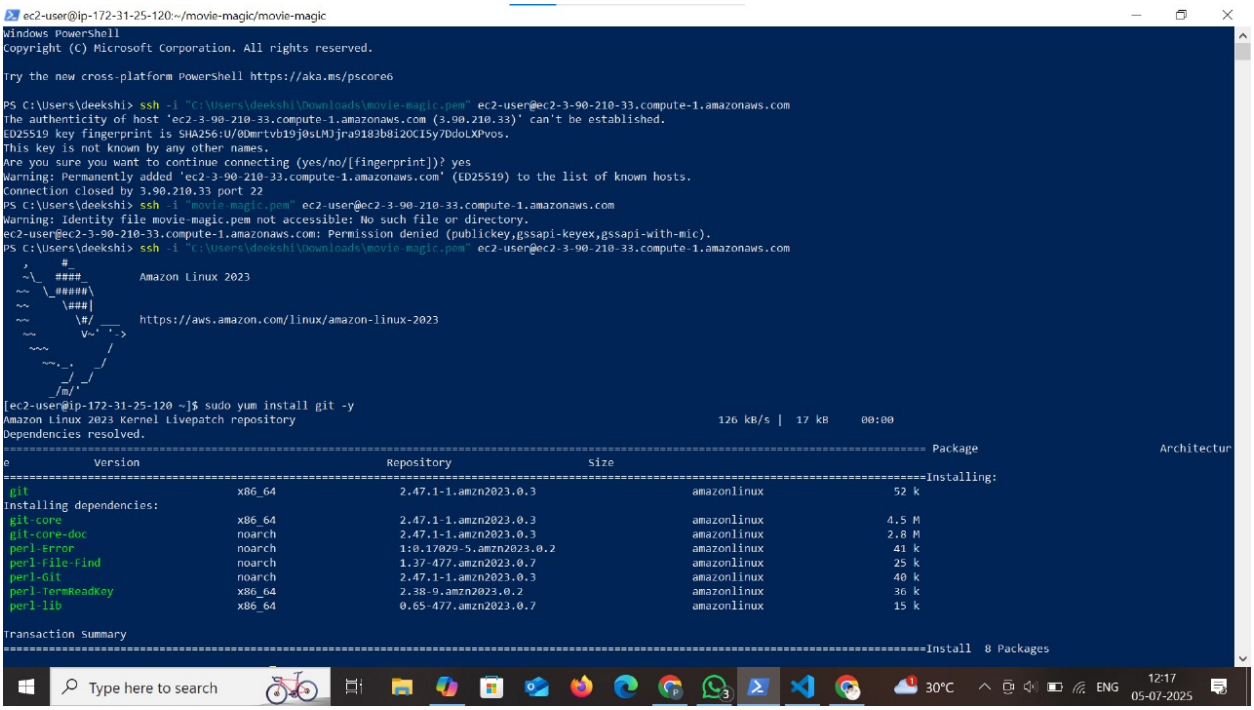
  sudo yum update -y

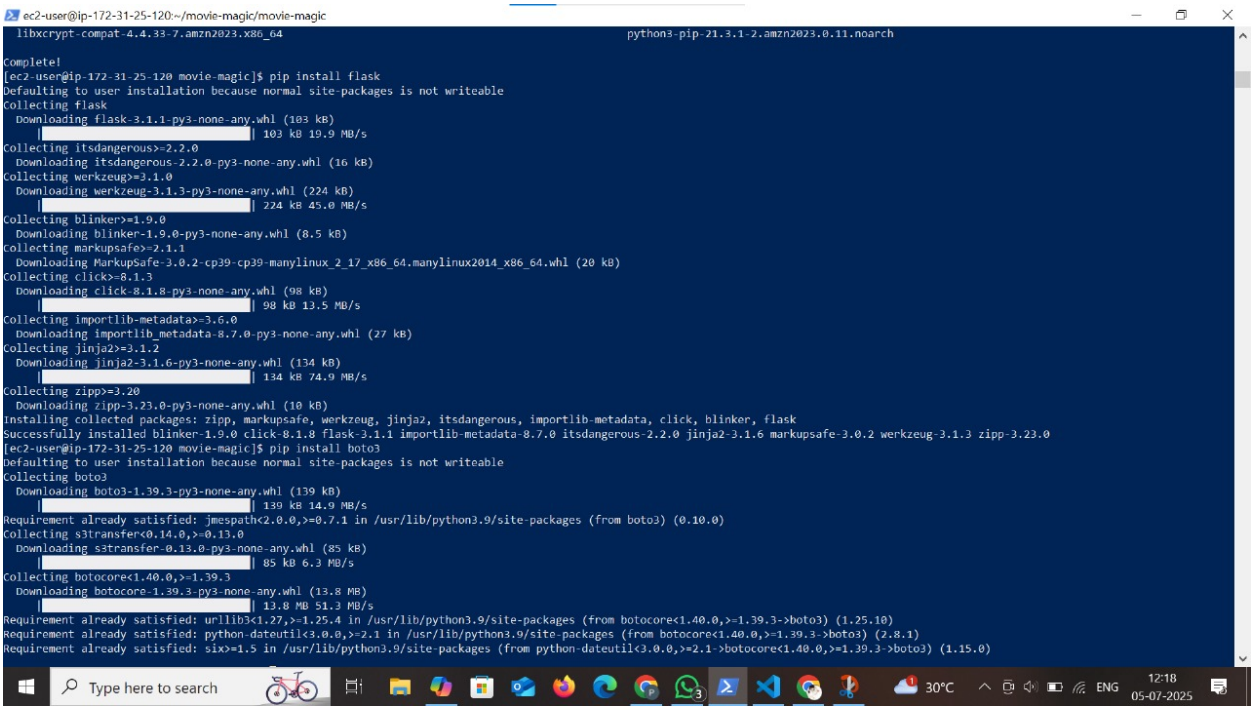
  sudo yum install python3 git

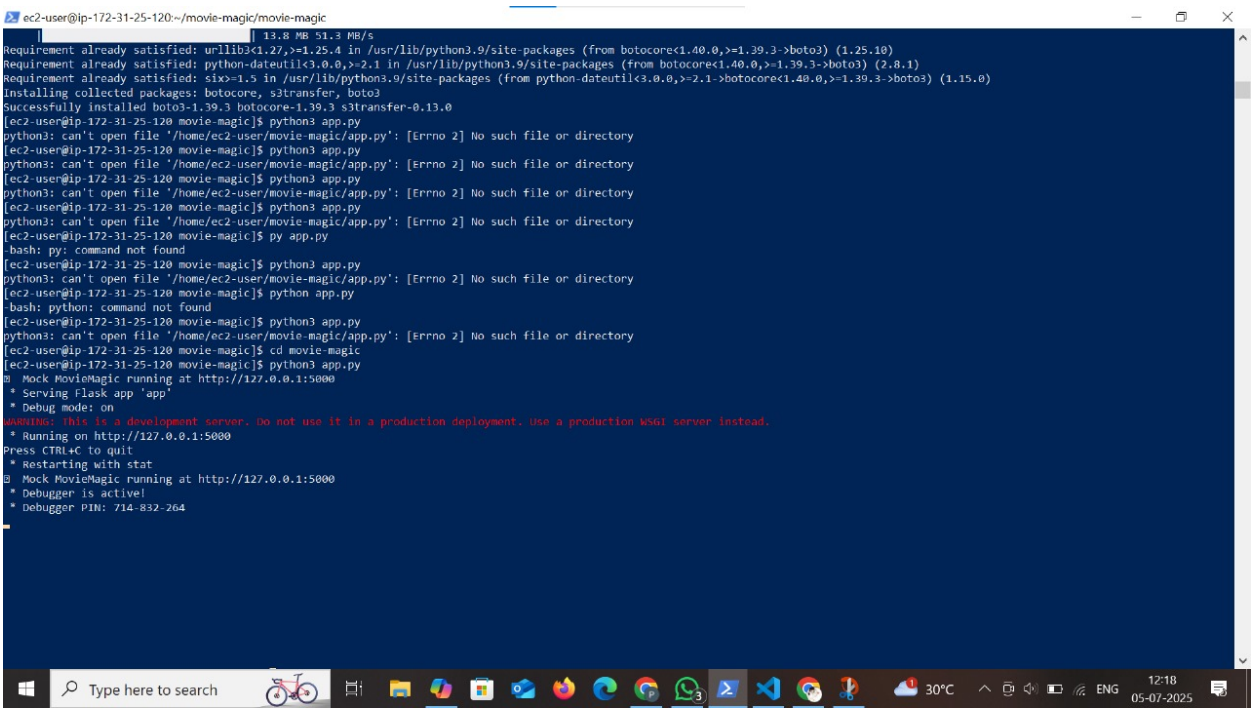
  sudo pip3 install flask boto3

- **Verify Installations:**

  flask --version
  git --version





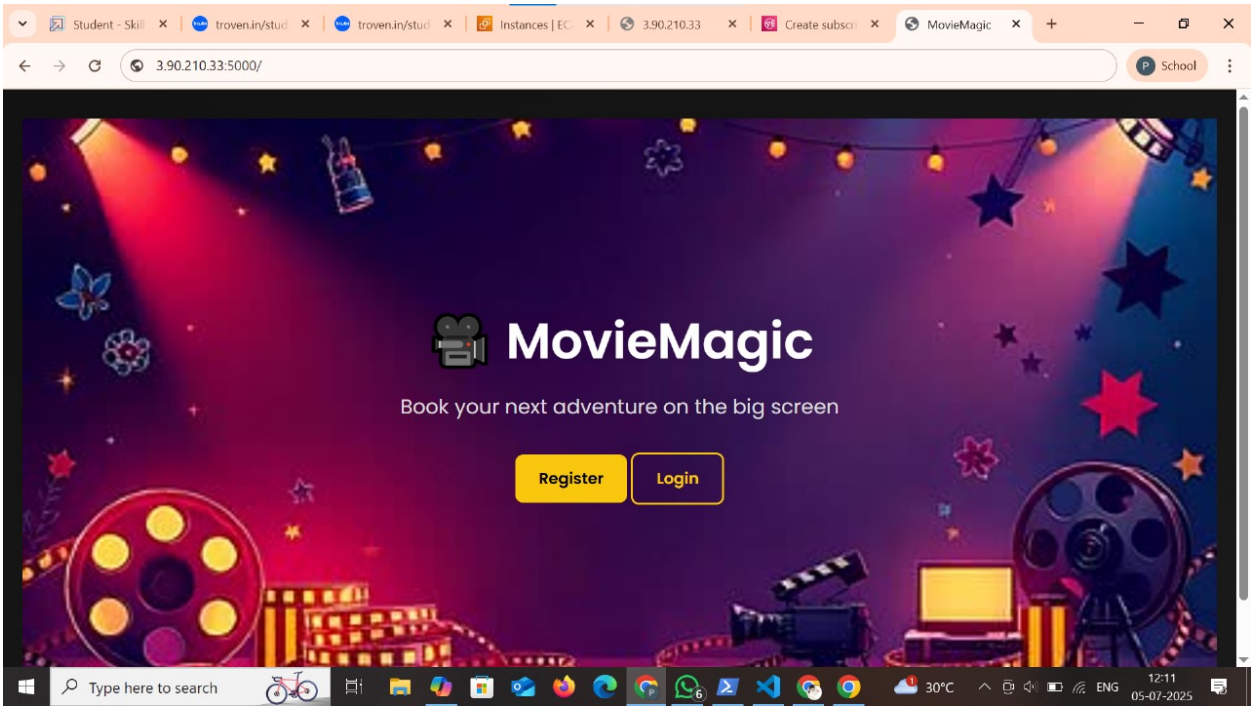




- Access the website through: your-ec2-public-ip
- Public IPs : http://3.90.210.33:5000/

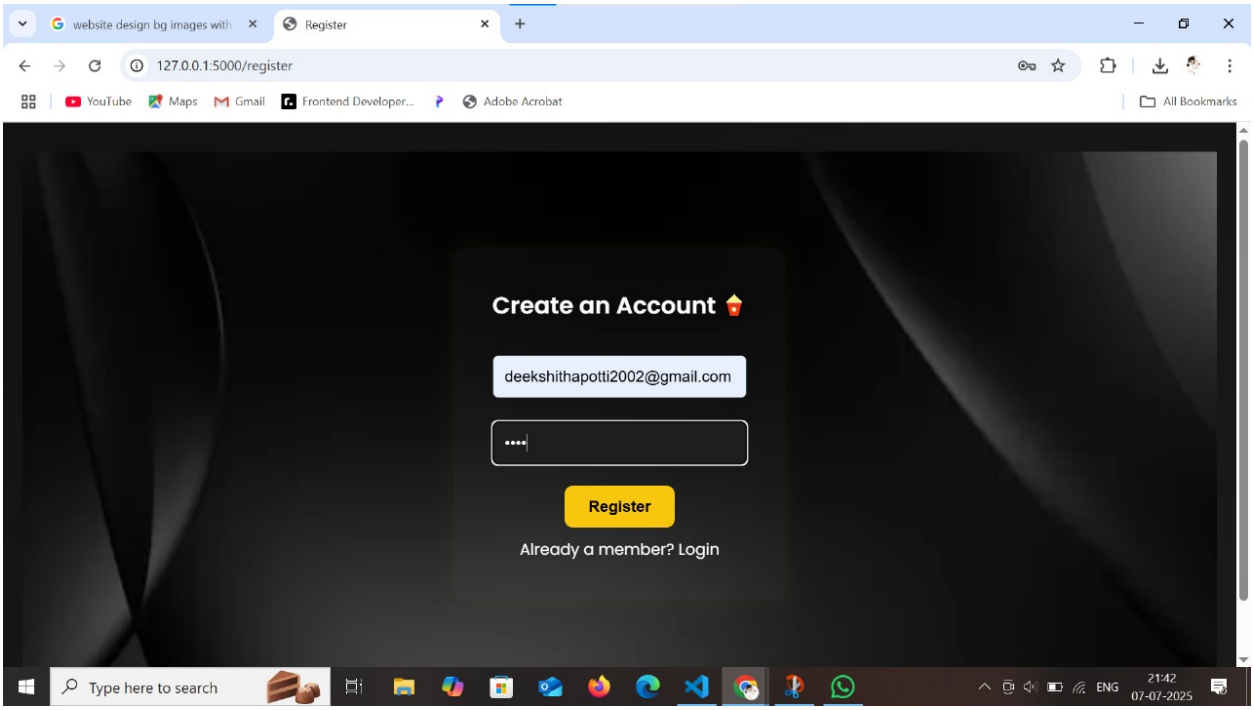
# Milestone 8: Testing and Deployment

## Functional testing to verify the Project :

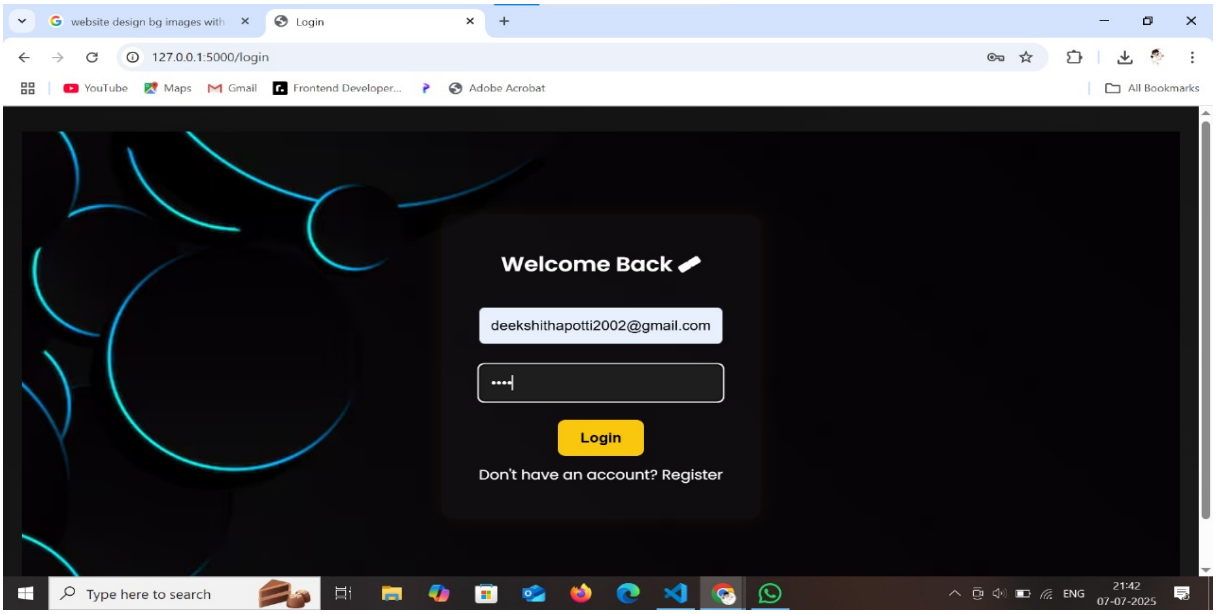
### Index Page :



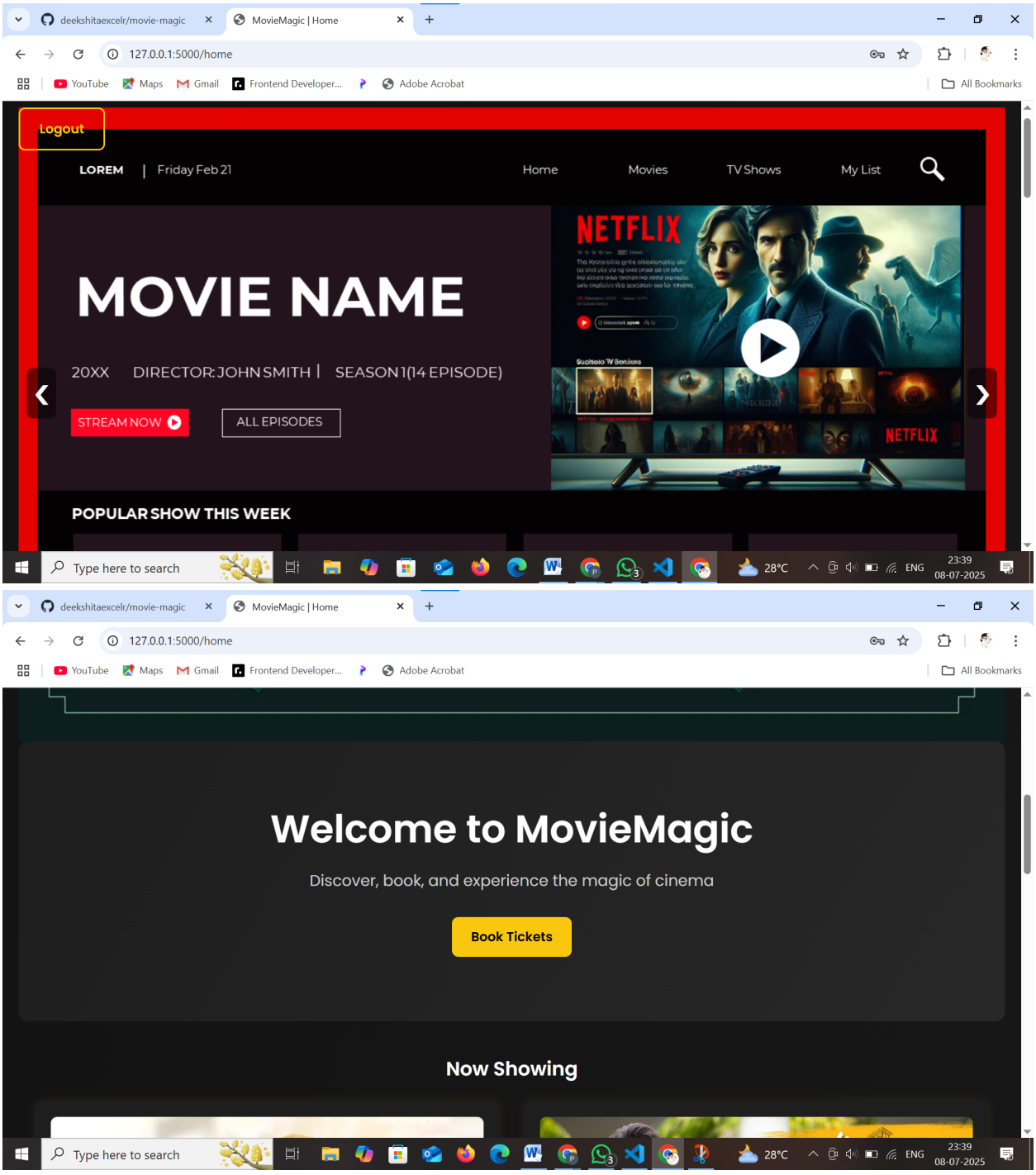
## Registeration Page :

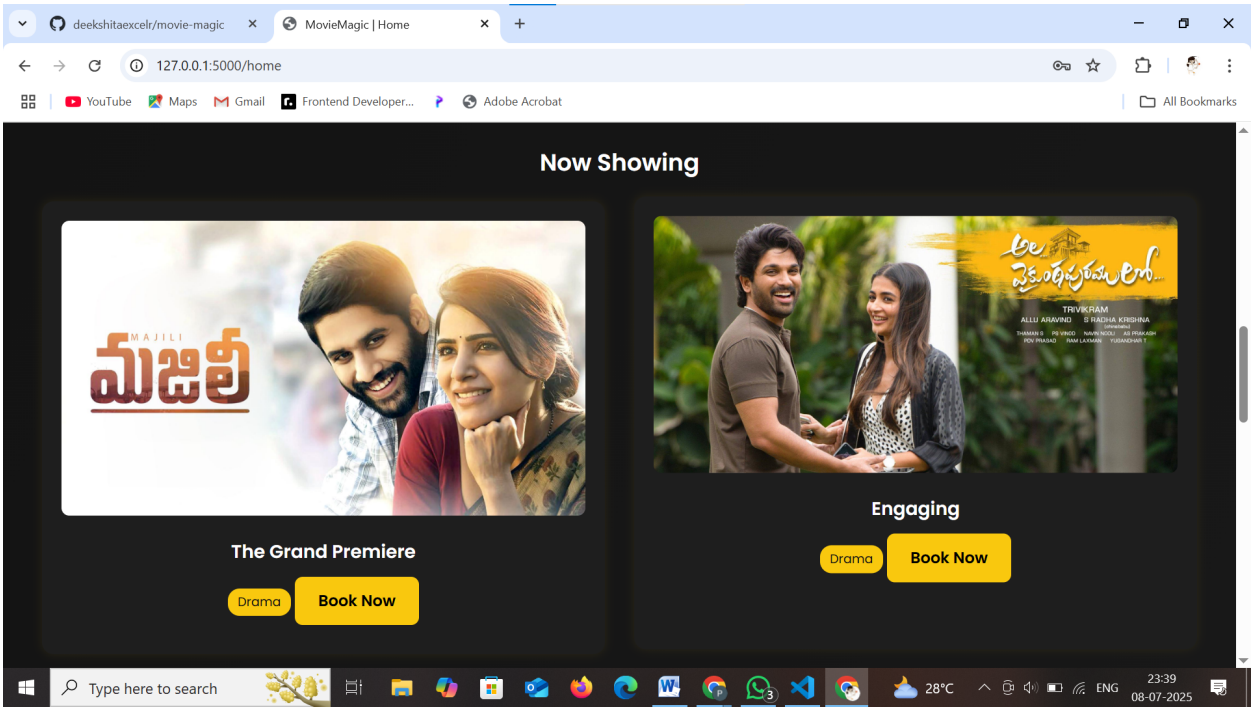


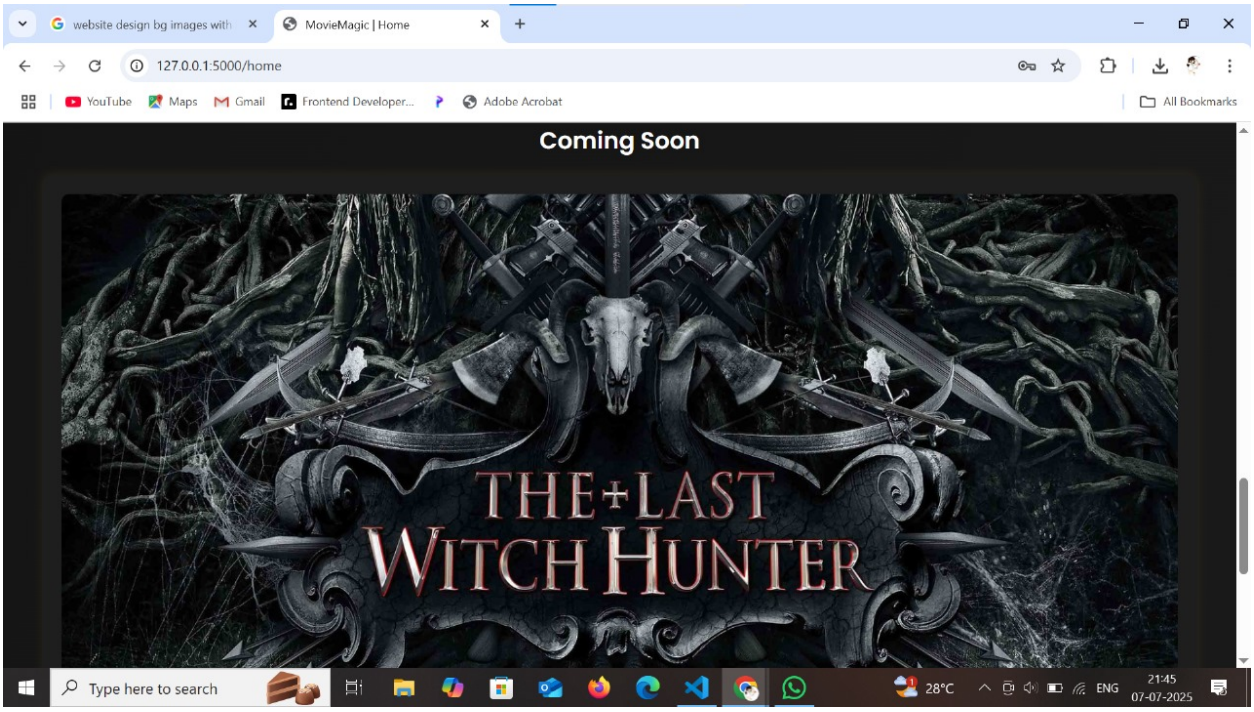

## Login Page:

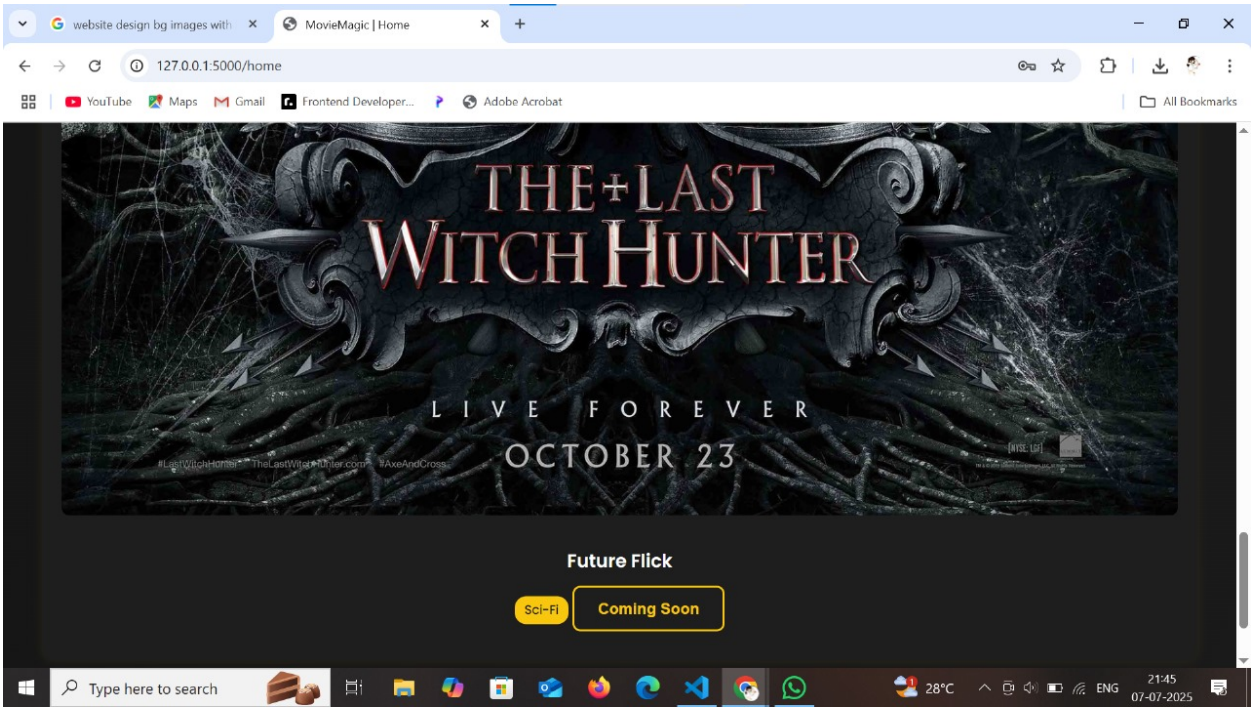


# Home Page :

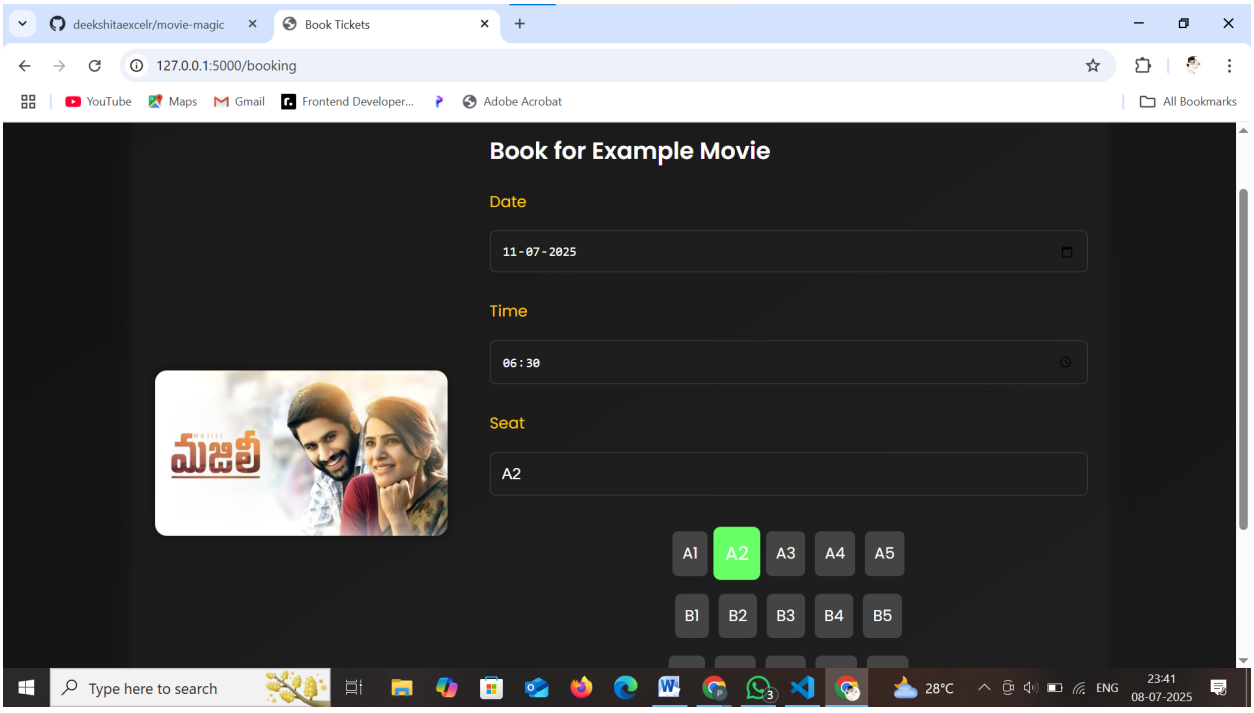

deekshitaexcelr/movie-magic
MovieMagic | Home
127.0.0.1:5000/home
YouTube
Maps
Gmail
Frontend Developer...
Adobe Acrobat
All Bookmarks
Logout
LOREM | Friday Feb 21
Home
Movies
TV Shows
My List
MOVIE NAME
20XX DIRECTOR: JOHN SMITH | SEASON 1 (14 EPISODE)
STREAM NOW
ALL EPISODES
NETFLIX
POPULAR SHOW THIS WEEK
Type here to search
28°C
ENG
23:39
08-07-2025
Welcome to MovieMagic
Discover, book, and experience the magic of cinema
Book Tickets
Now Showing

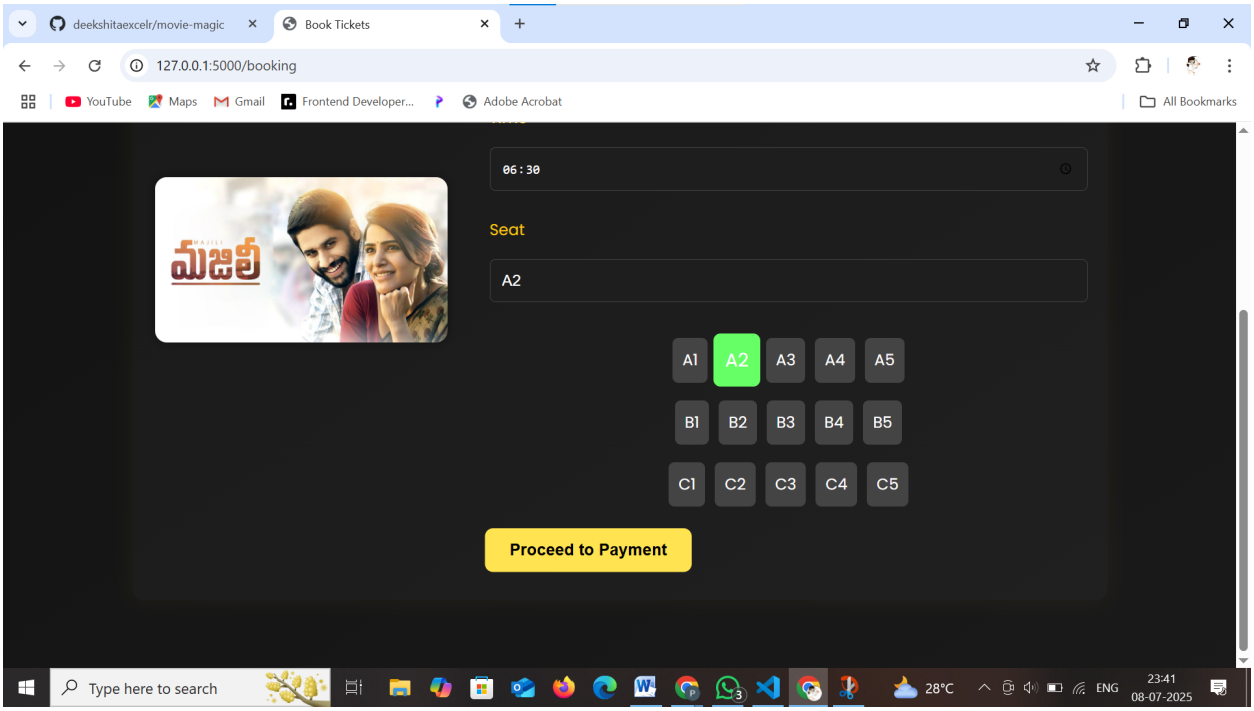

deekshitaexcelr/movie-magic
MovieMagic | Home
127.0.0.1:5000/home
YouTube
Maps
Gmail
Frontend Developer...
Adobe Acrobat
All Bookmarks
Now Showing
MAJILI
మజిలీ
The Grand Premiere
Drama
Book Now
TRIVIKRAM
Engaging
Drama
Book Now
Type here to search
28°C
ENG
23:39
08-07-2025


website design bg images with
MovieMagic | Home
127.0.0.1:5000/home
YouTube
Maps
Gmail
Frontend Developer...
Adobe Acrobat
All Bookmarks
Coming Soon
THE LAST
WITCH HUNTER
Type here to search
28°C
ENG
21:45
07-07-2025

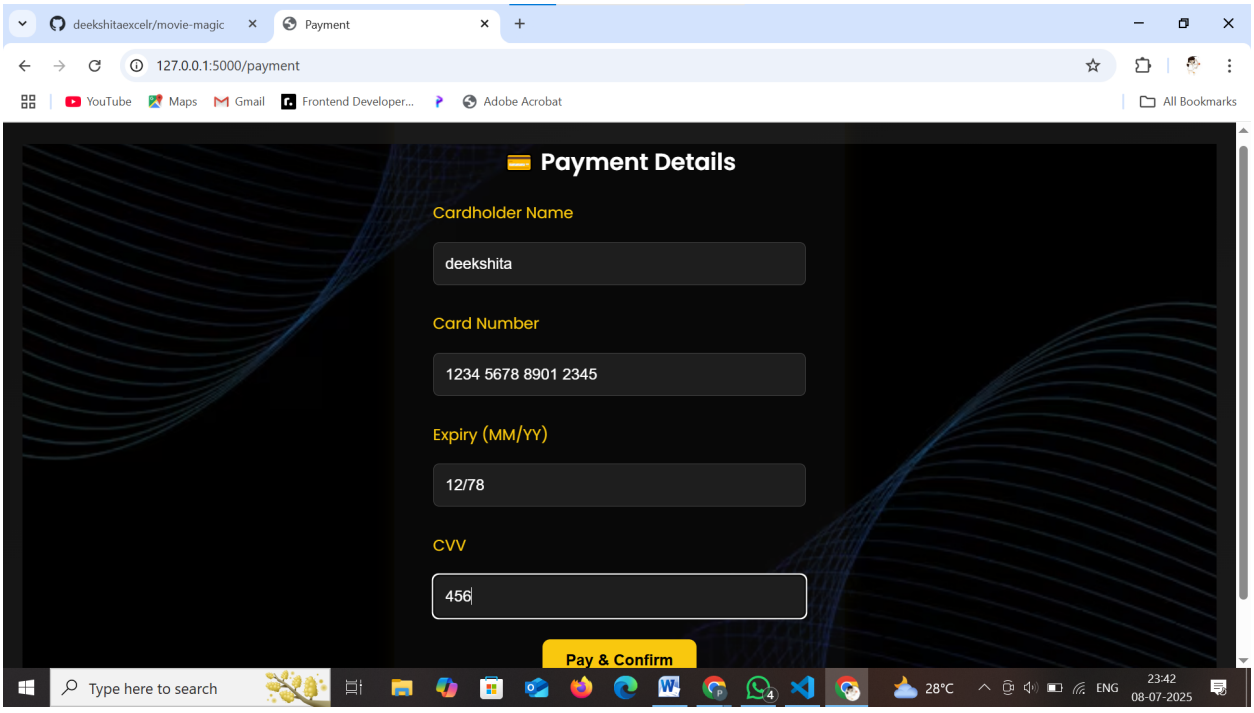



## Booking_form Page :

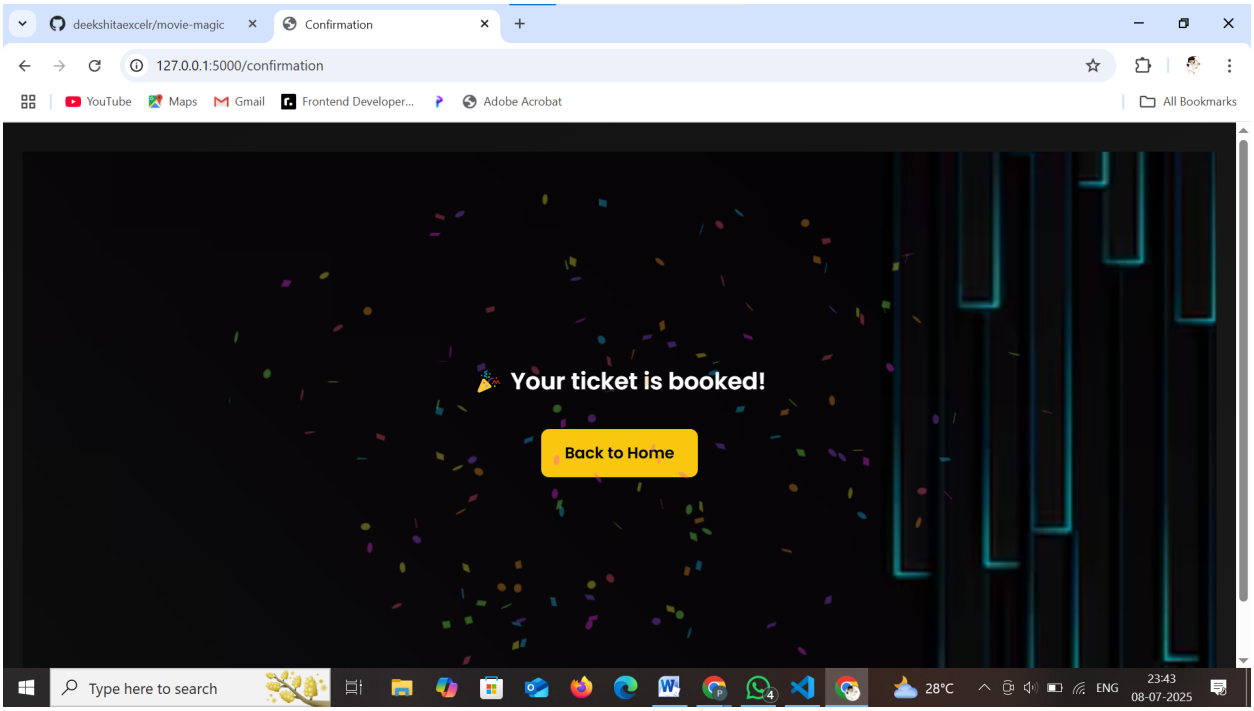





# Payment Page :

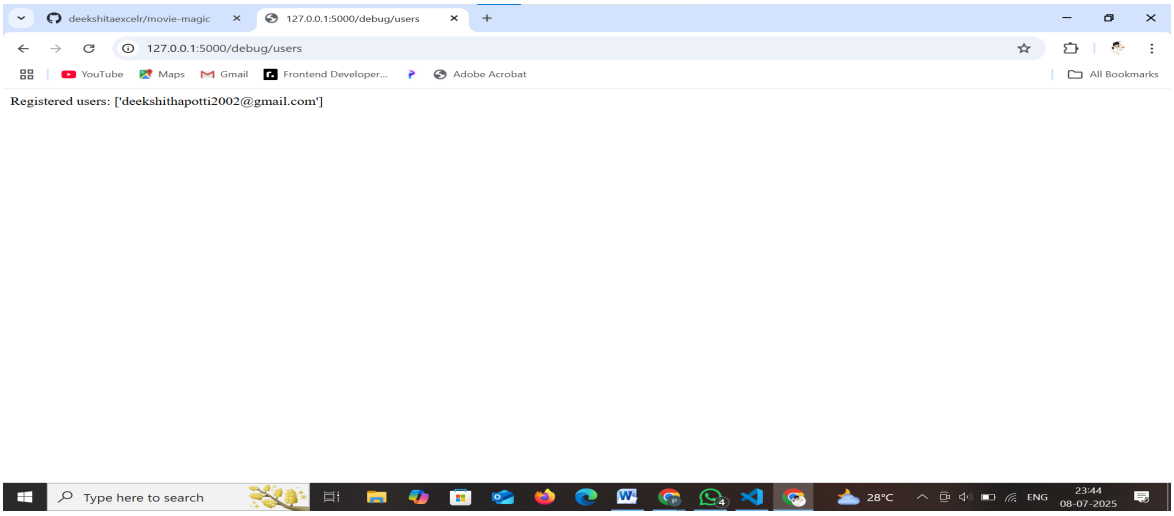


## confirmation Page :

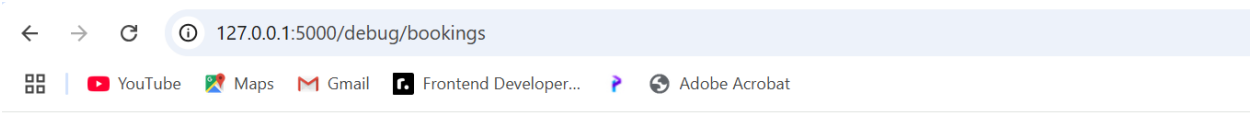



## It stores the data in the backend :

## a.users data :



## b. booking data :




### All Bookings

- **User:** deekshithapotti2002@gmail.com, **Movie:** Example Movie, **Seat:** A2, **Date:** 2025-07-11, **Time:** 06:30, **ID:** 4c386d92


SMARTBRIDGE
Let's Bridge the Gap


Smart
Internz