Capture Moments-AWS Powered Photographer Booking System

Project Description:

At Capture Moments Photography, the growing client base has created challenges in managing bookings and connecting customers with the right photographers. The manual booking process was inefficient and prone to errors, causing scheduling conflicts and customer dissatisfaction.To solve this, Capture Moments developed a cloud-based Photographer Booking Platform. Using Flask for backend development, AWS EC2 for hosting, and DynamoDB for managing data, the system allows clients to register, log in, and book photographers online. This cloud-based solution enhances the booking experience, providing seamless service for all clients while optimizing photographer scheduling and data management.

Scenario 1: Efficient Booking System for Clients

In the Capture Moments Booking Platform, AWS EC2 ensures a reliable infrastructure to manage multiple clients accessing the platform simultaneously. For example, a client can log in, navigate to the booking page, and easily submit a request for their preferred photographer. Flask handles backend operations, efficiently retrieving and processing user data in real-time. The cloud-based architecture allows the platform to handle a high volume of booking requests during peak wedding seasons, ensuring smooth operation without delays.

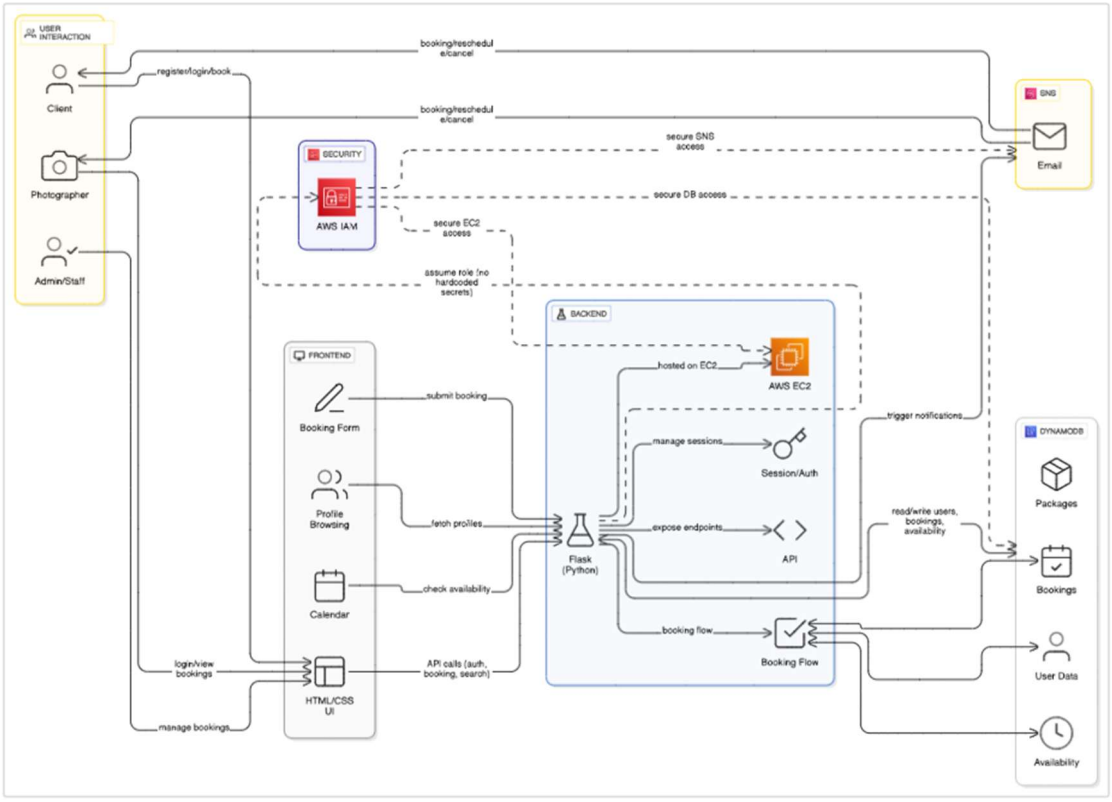
Scenario 2: Centralized Booking Management

When clients book photography sessions, the Capture Moments Booking System processes these requests through a streamlined workflow. For instance, a client books a wedding photographer, and Flask processes the request, storing all the booking details securely in DynamoDB. This centralized database approach ensures all booking information is maintained in a single source of truth, allowing administrative staff to track upcoming appointments, manage photographer schedules, and maintain complete records of client preferences and requirements.

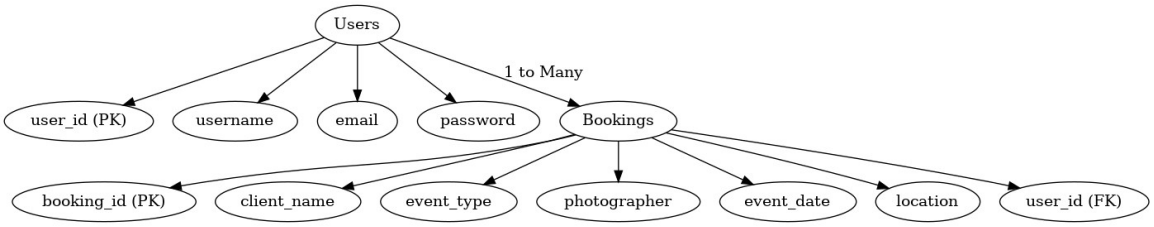
Scenario3:Easy Access to Library Resources

The Capture Moments Booking Platform provides clients with easy access to available photographers and packages. For example, a client logs in and views the list of available photographers specializing in their event type. They can quickly check availability for specific dates or book immediately. Flask manages real-time data fetching from DynamoDB, while EC2 hosting ensures the platform performs seamlessly even when multiple clients access it simultaneously during popular booking periods, offering a smooth and uninterrupted user experience.

AWSARCHITECTURE



Entity Relationship(ER)Diagram:



Pre-requisites:

1. AWS Account Setup:AWSAccountSetup
2. AWS IAM (Identity and Access Management) :IAMOverview
3. AWS EC2 (Elastic Compute Cloud):EC2Tutorial
4. AWS DynamoDB :DynamoDBIntroduction
5. Amazon SNS:SNSDocumentation
6. Git Documentation : GitDocumentation

Project Work Flow:

Milestone 1. Backend Development and Application Setup

Develop the Backend Using Flask.

Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

Set up an AWS account if not already done.

Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup

Create a DynamoDB Table.

Configure Attributes for photographers and Book Requests.

Milestone 4. IAM Role Setup

Create IAM Role

Attach Policies

Milestone 5. EC2 Instance Setup

Launch an EC2 instance to host the Flask application.

Configure security groups for HTTP, and SSH access.

Milestone 6. Deployment on EC2

Upload Flask Files

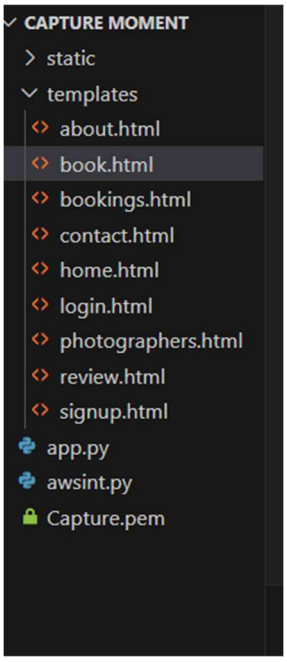
Run the Flask App

Milestone 7. Testing and Deployment

Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

Milestone 1. Backend Development and Application Setup

Folder structure



Description of the code :

Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for, flash, session
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import logging
import boto3
import uuid
import os
from botocore.exceptions import ClientError
```

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

Routes for Web Pages

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

```
@app.route('/logout')
def logout():
    session.pop('username', None)
    session.pop('fullname', None)
    flash('You have been logged out', 'info')
    return redirect(url_for('index'))

@app.route('/home')
def home():
    if 'username' not in session:
        return redirect(url_for('login', next=request.path))
    return render_template('home.html', username=session['username'])

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/services')
def services():
    return render_template('services.html')

@app.route('/photographers')
def photographers():
    return render_template('photographers.html')
```

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

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

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if 'username' in session:
        return redirect(url_for('home'))

    if request.method == 'POST':
        fullname = request.form['fullname']
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']

        try:
            # Check if username already exists
            response = users_table.get_item(Key={'username': username})
            if 'Item' in response:
                flash('Username already exists!', 'error')
                return redirect(url_for('signup'))

            # Create new user in DynamoDB
            users_table.put_item(
                Item={
                    'username': username,
                    'password': generate_password_hash(password),
                    'fullname': fullname,
                    'email': email,
                    'created_at': datetime.now().isoformat()
```

```
                'email': email,
                'created_at': datetime.now().isoformat()
            }
        )

        flash('Registration successful! Please login.', 'success')
        return redirect(url_for('login'))

    except ClientError as e:
        logger.error(f"Database error during signup: {e}")
        flash('An error occurred during registration. Please try again.', 'error')

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

Routes to the events with the specialist photographers in that particular sector

```
@app.route('/logout')
def logout():
    session.pop('username', None)
    session.pop('fullname', None)
    flash('You have been logged out', 'info')
    return redirect(url_for('index'))

@app.route('/home')
def home():
    if 'username' not in session:
        return redirect(url_for('login', next=request.path))
    return render_template('home.html', username=session['username'])

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/services')
def services():
    return render_template('services.html')

@app.route('/photographers')
def photographers():
    return render_template('photographers.html')
```

Route to the booking template

```python
@app.route('/booking', methods=['GET', 'POST'])
def booking():
    if 'username' not in session:
        flash('Please login to book a photographer', 'error')
        return redirect(url_for('login', next=request.path))

    event_type = request.args.get('event', '')

    if request.method == 'POST':
        try:
            # Retrieve user data
            response = users_table.get_item(Key={'username': session['username']})
            user = response.get('Item', {})

            # Validate required fields
            required_fields = ['event_type', 'photographer', 'start_date', 'end_date', 'name', 'email', 'phone', 'pac
            missing_fields = [field for field in required_fields if not request.form.get(field)]

            if missing_fields:
                flash(f'Please fill all required fields: {", ".join(missing_fields)}', 'error')
                return redirect(url_for('booking', event=event_type))

            # Generate a unique booking ID
            booking_id = str(uuid.uuid4())

            # Process booking form
            booking_data = {
                'booking_id': booking_id,
                'username': session['username'],
```

```python
            booking_data = {
                'booking_id': booking_id,
                'username': session['username'],
                'user': session['fullname'],
                'user_email': user.get('email', ''),
                'name': request.form['name'],
                'email': request.form['email'],
                'phone': request.form['phone'],
                'event_type': request.form['event_type'],
                'photographer': request.form['photographer'],
                'start_date': request.form['start_date'],
                'end_date': request.form['end_date'],
                'package': request.form['package'],
                'payment_method': request.form['payment'],
                'notes': request.form.get('notes', ''),
                'booking_date': datetime.now().isoformat(),
                'status': 'Confirmed'
            }

            # Save to DynamoDB
            bookings_table.put_item(Item=booking_data)
            logger.info(f"Booking created with ID: {booking_id}")

            # Send notification
            # send_booking_notification(booking_data)

            # Store booking ID in session for the success page
            session['last_booking_id'] = booking_id
```

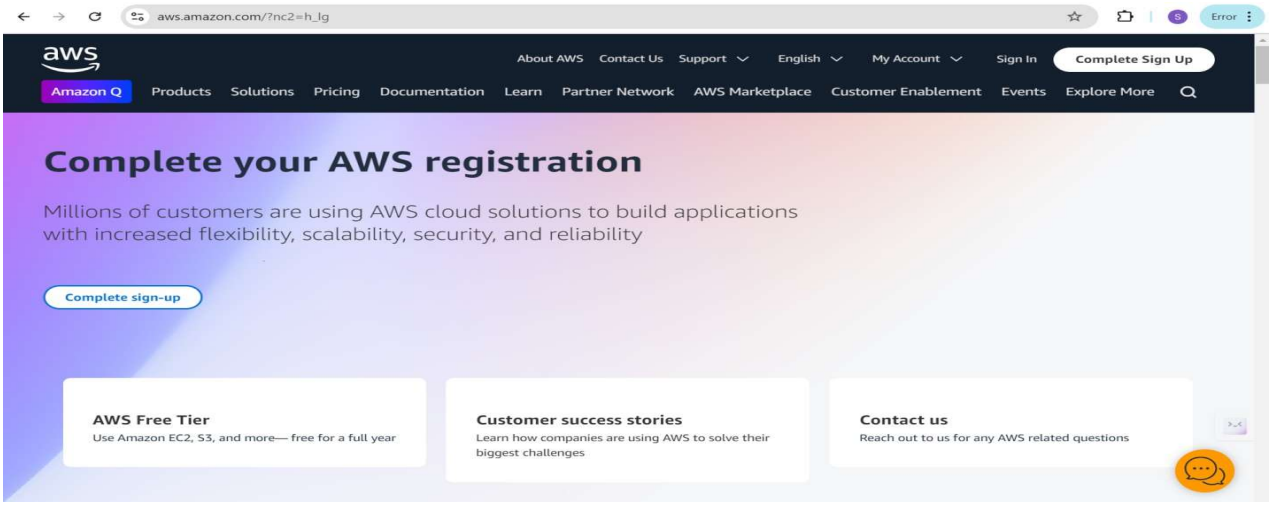
```
            # Store booking ID in session for the success page
            session['last_booking_id'] = booking_id

            return redirect(url_for('success'))

        except Exception as e:
            logger.error(f"Error in booking form: {str(e)}")
            flash(f'An error occurred: {str(e)}', 'error')
            return redirect(url_for('booking', event=event_type))

    return render_template('booking.html', event_type=event_type)

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

    # Just render the success template without passing booking details
    return render_template('success.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')
```

```
if__name__=='_main_':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Milestone 2. AWS Account Setup and Login

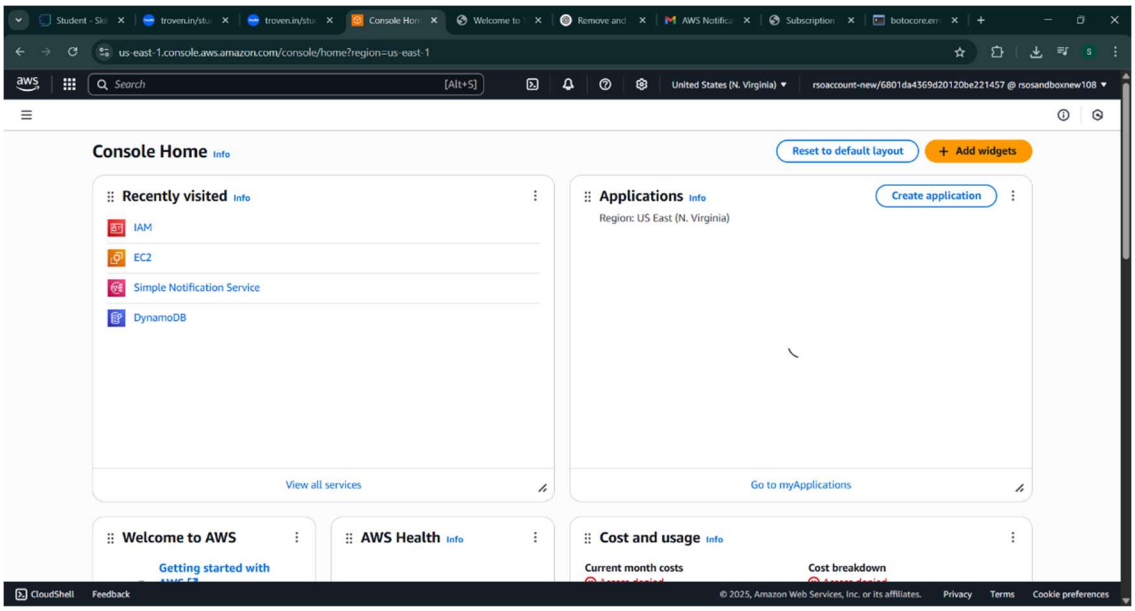
2.1 Set up an AWS account if not already done.

SignupforanAWSaccountandconfigurebillingsettings.



2.2 Log in to the AWS Management Console

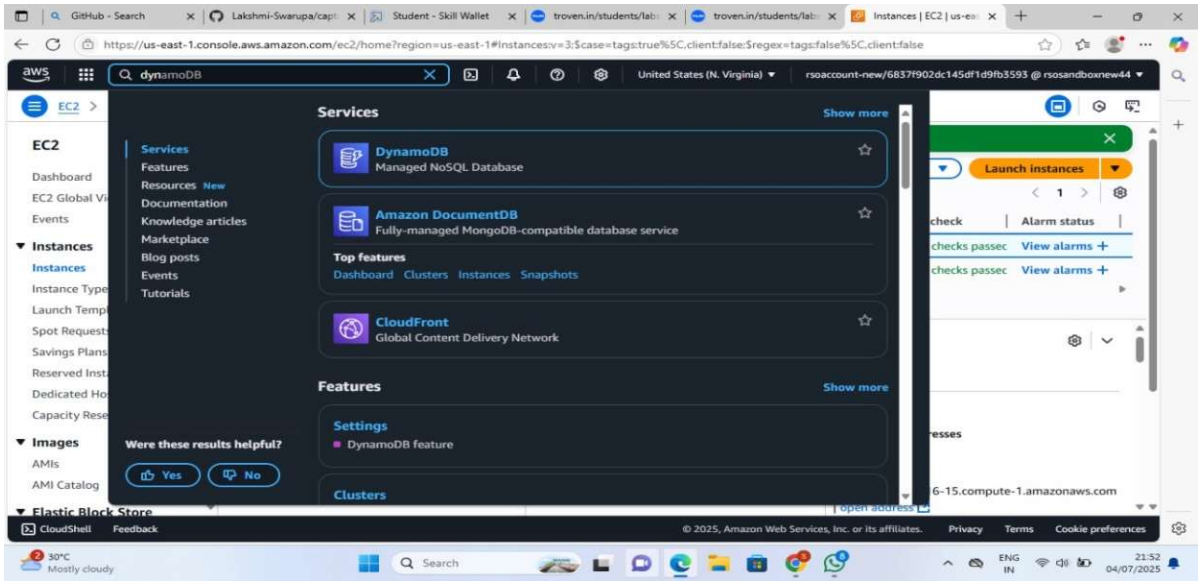
After setting up your account ,login to the AWSManagementConsole

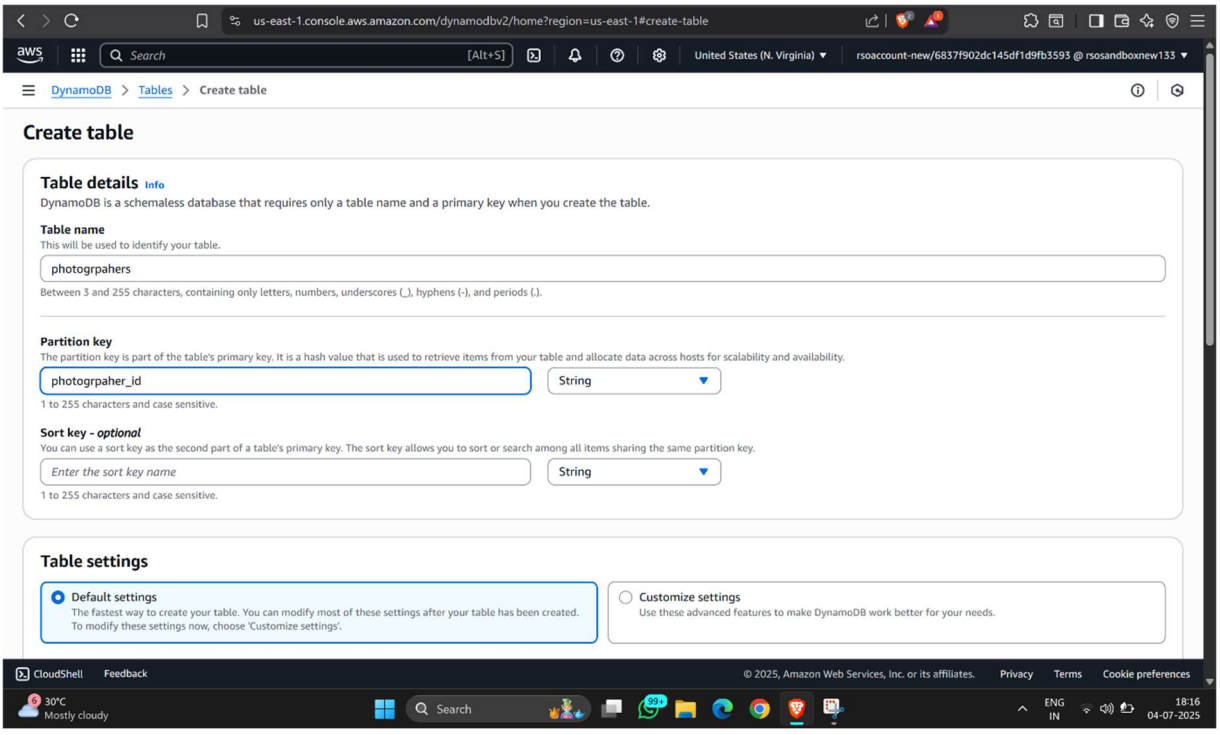


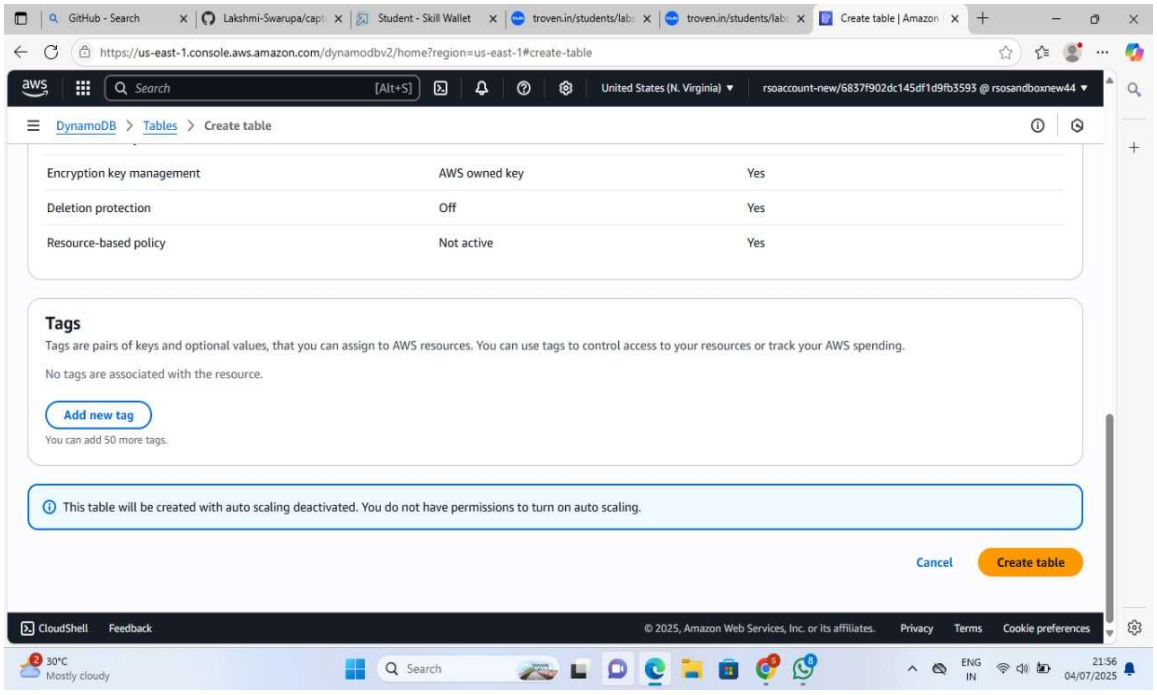
Milestone 3. DynamoDB Database Creation and Setup

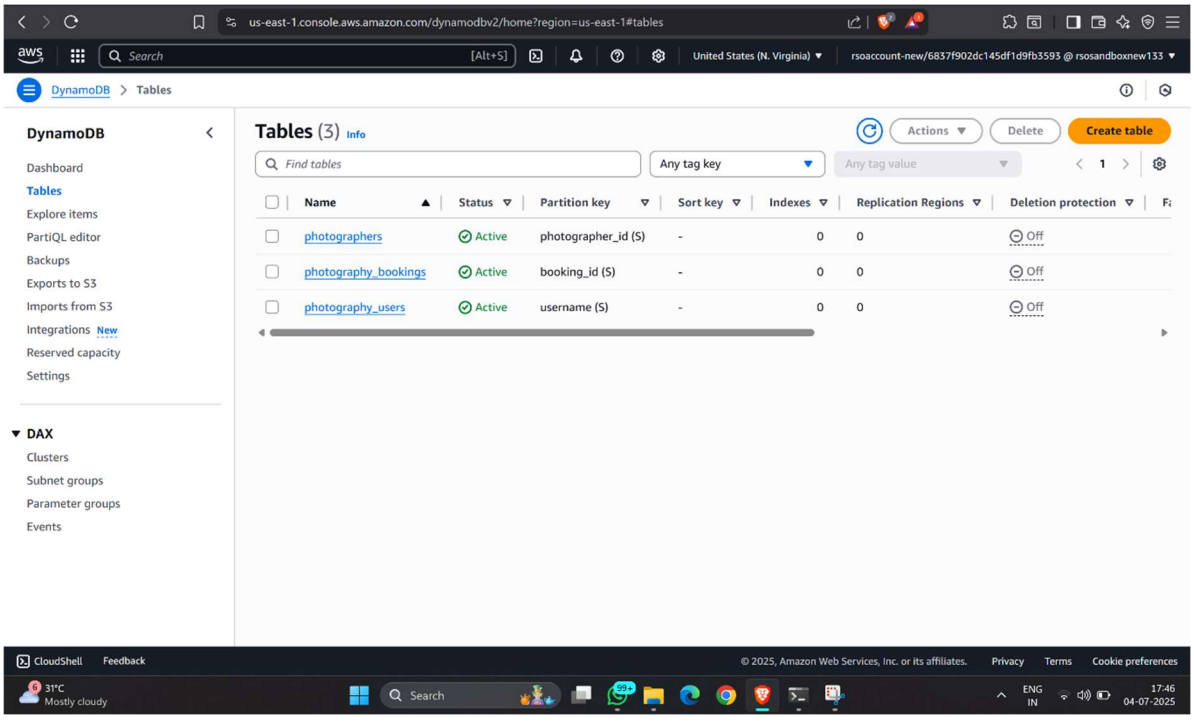
3.1 Create a DynamoDB Table.

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

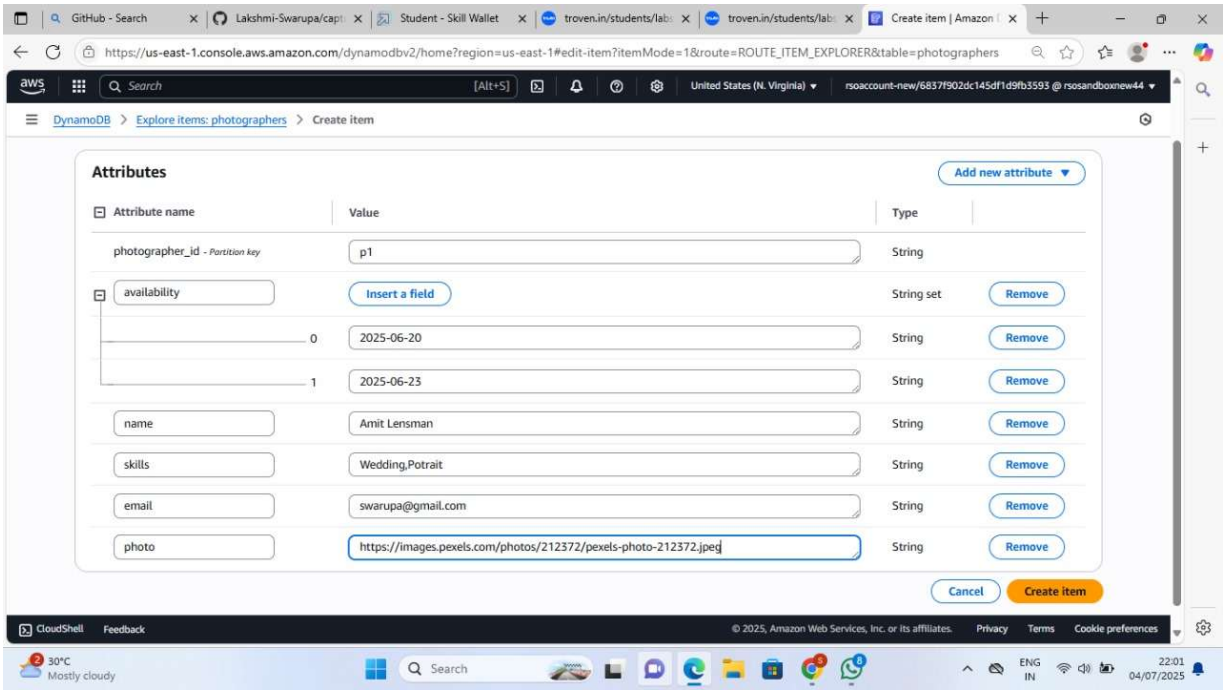





https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table
DynamoDB > Tables > Create table
Encryption key management
AWS owned key
Yes
Deletion protection
Off
Yes
Resource-based policy
Not active
Yes
Tags
Tags are pairs of keys and optional values, that you can assign to 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.
This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.
Cancel
Create table
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

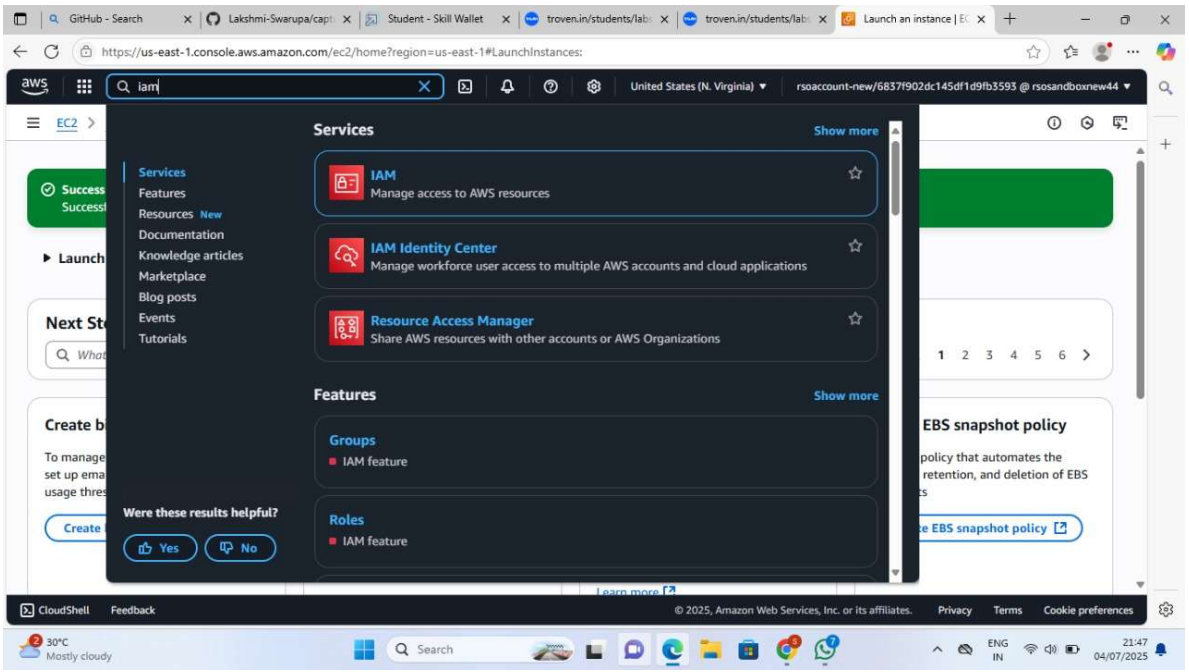

us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables
DynamoDB > Tables
DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
Tables (3) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
photographers
Active
photographer_id (S)
0
0
Off
photography_bookings
Active
booking_id (S)
0
0
Off
photography_users
Active
username (S)
0
0
Off
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

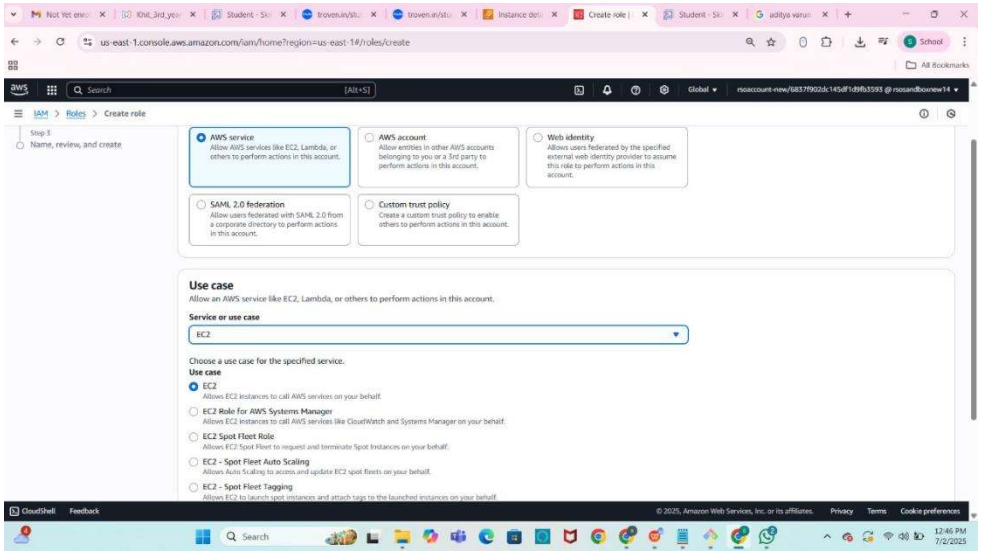
3.2 Configure Attributes for photographers and Book Requests



Milestone 4. IAM Role Setup

4.1 Create IAM Role

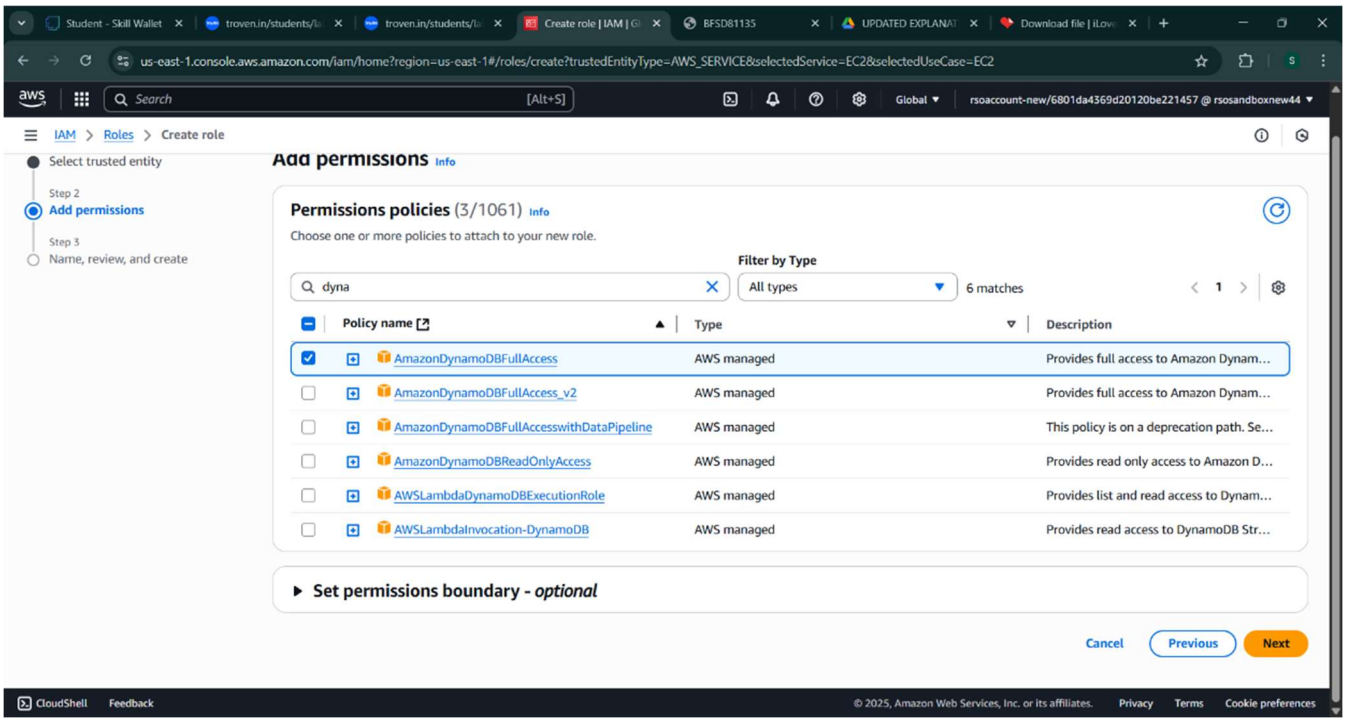


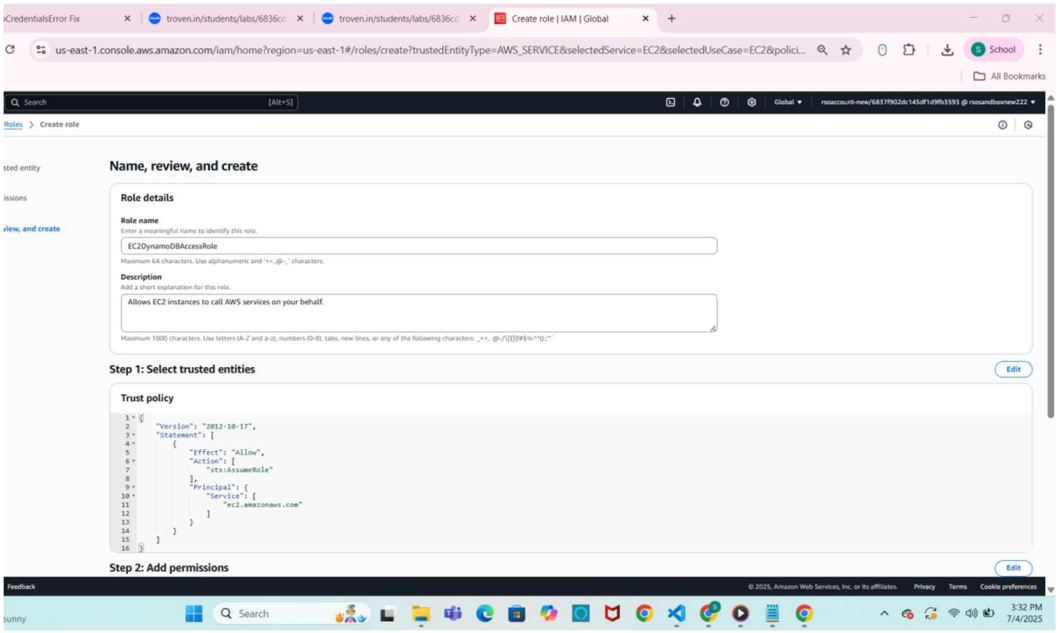


4.2 Attach Policies

Attach the following policies to the role:

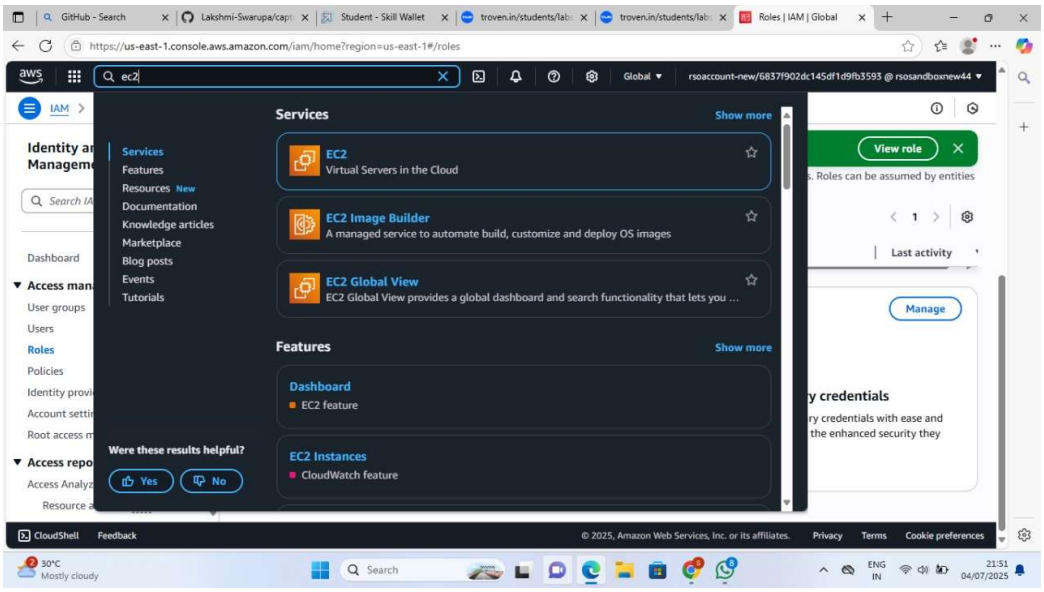
- AmazonDynamoDBFullAccess:AllowsEC2toperformread/writeoperationson DynamoDB.
- AmazonSNSFullAccess:GrantsEC2theabilitytosendnotificationsviaSNS.

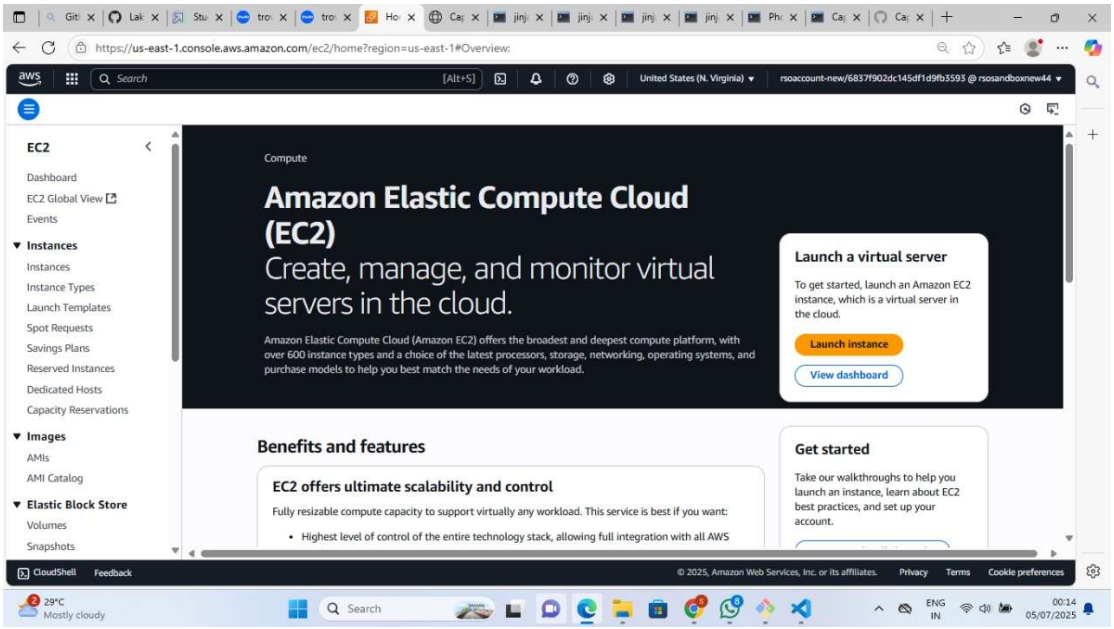




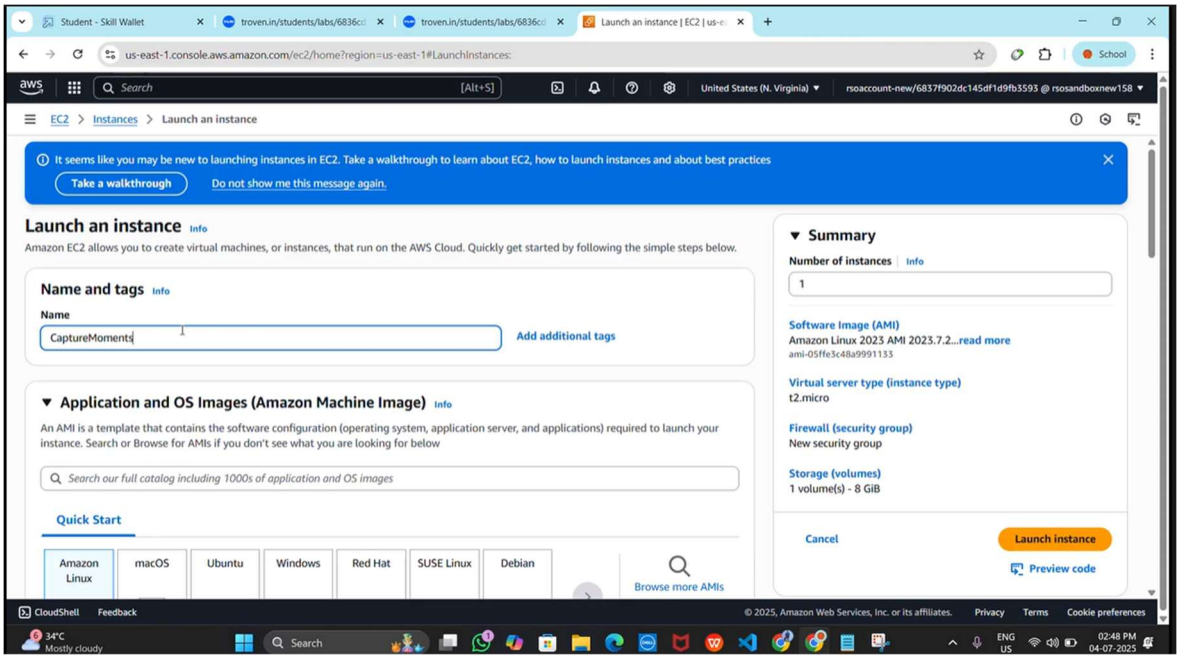
Milestone 5. EC2 Instance Setup

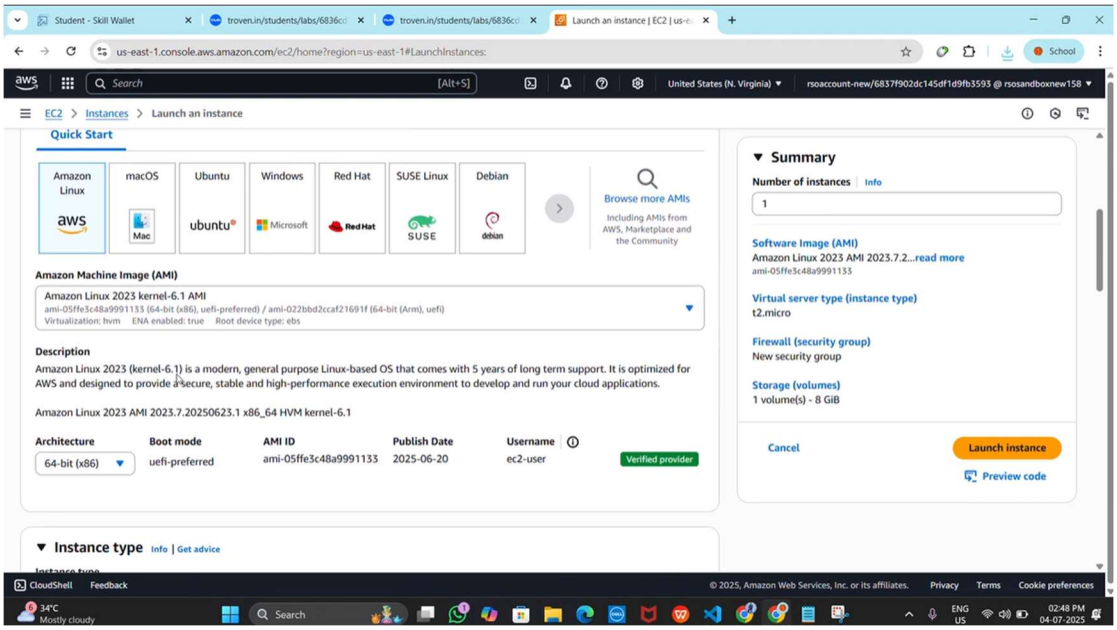
5.1 Launch an EC2 instance to host the Flask application.



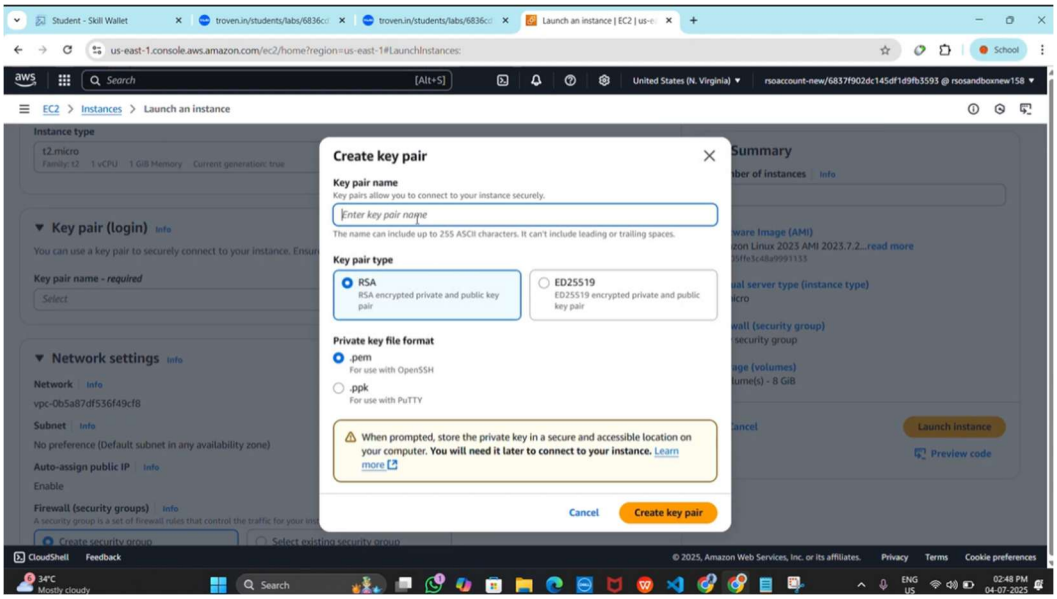


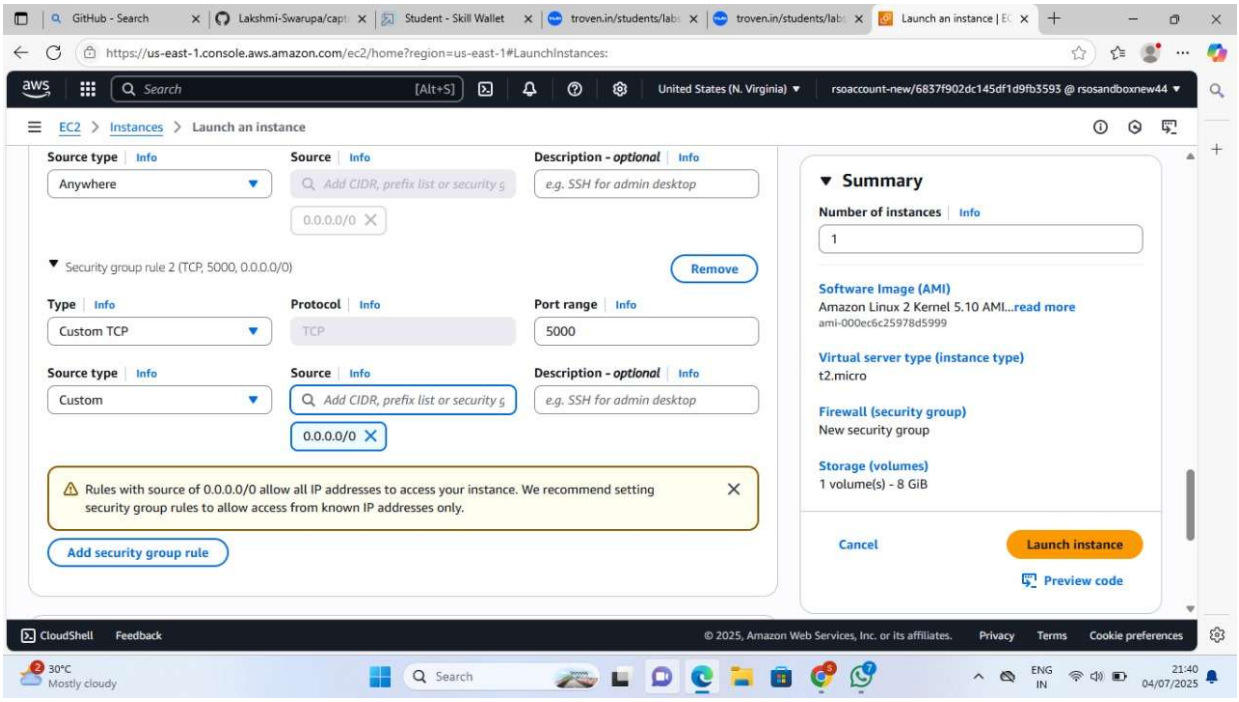
- ClickonLaunchinstancetolaunchEC2instance



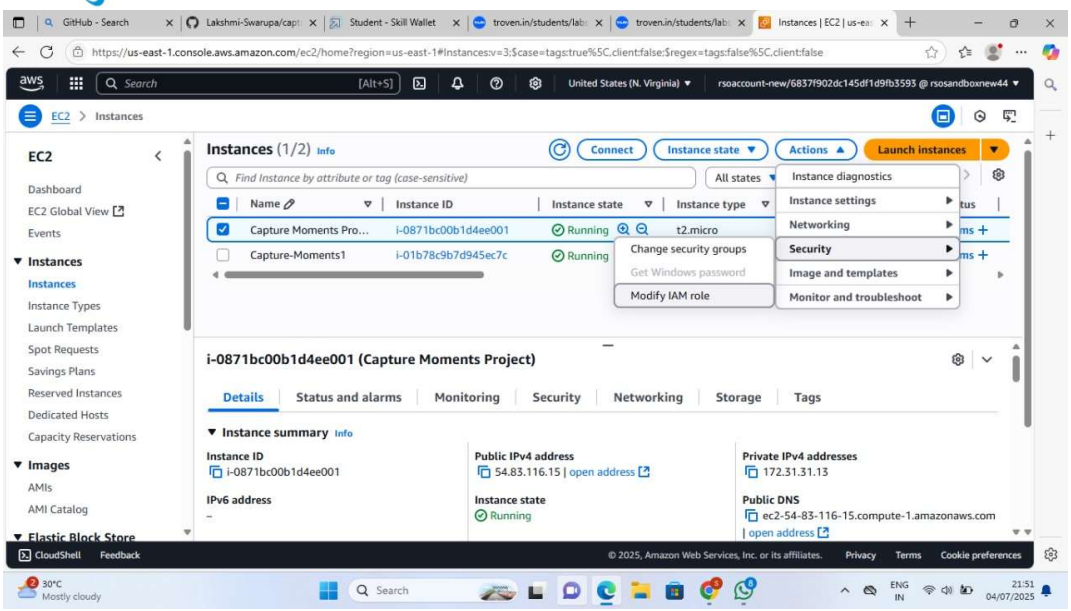


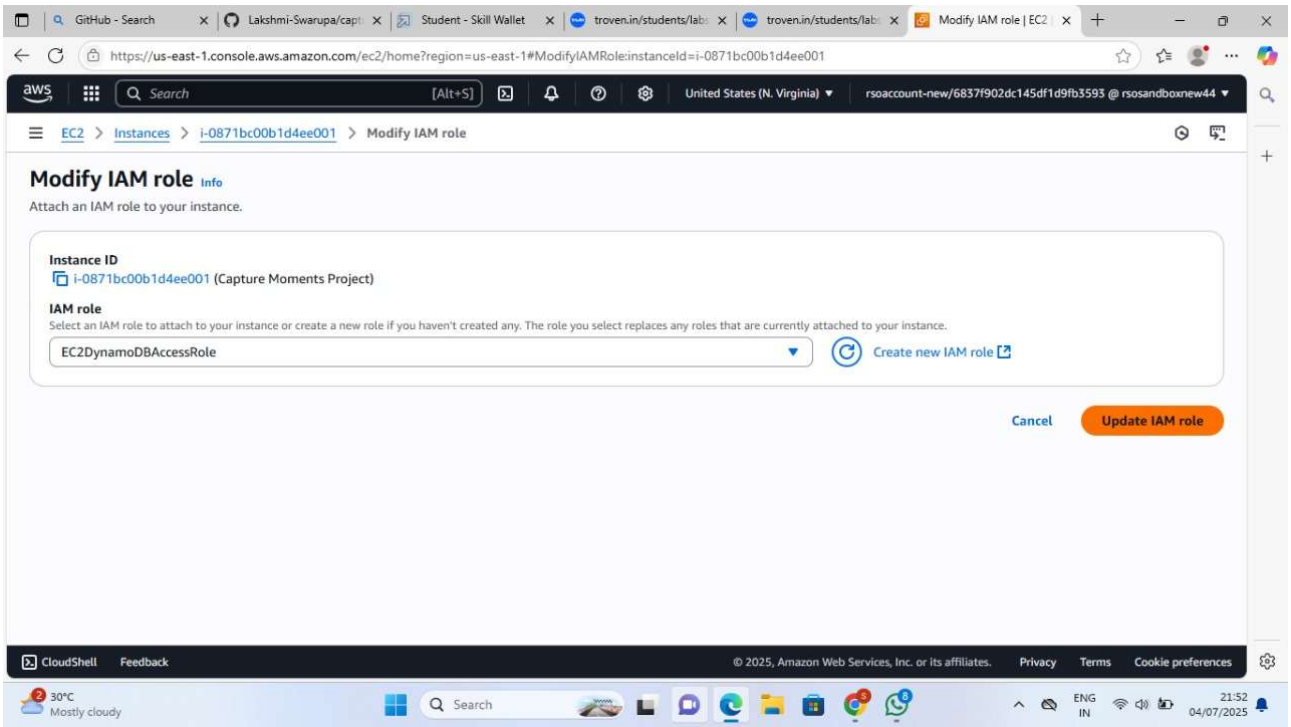
- Create a new key pair and save it with .pem extension



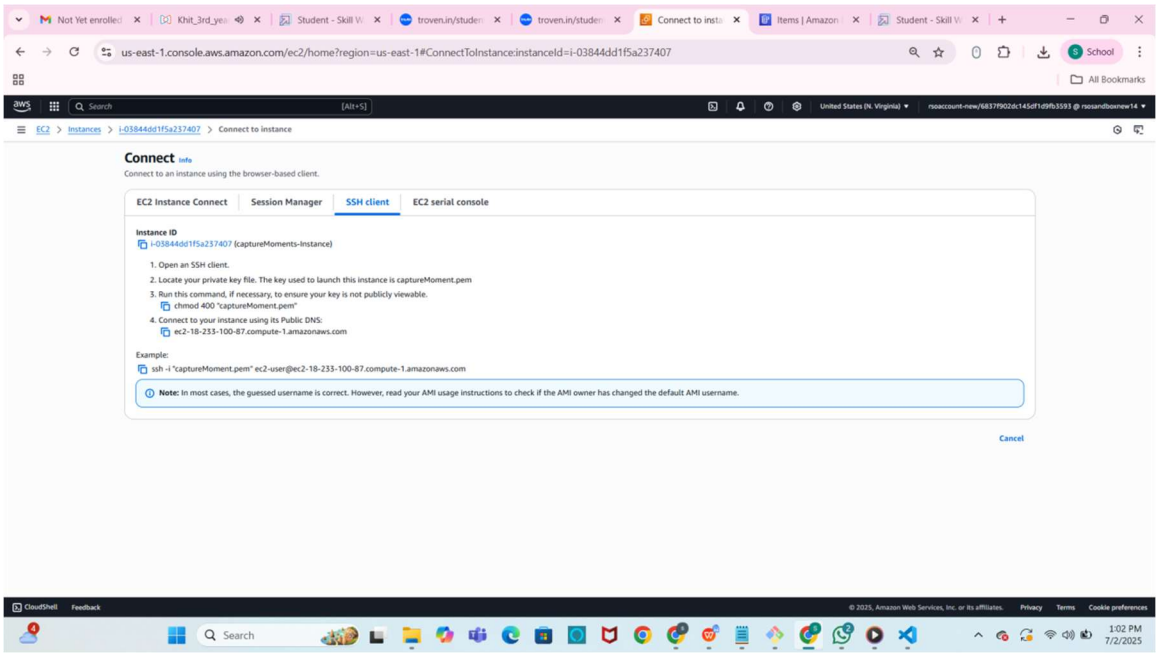


- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2Dashboard**,clickthe**Connect**button.Fromtheconnectionmethodspresented,choose **EC2InstanceConnect**.Finally,click**Connect**again,andanewbrowser-basedterminalwill open, allowing you to access your EC2 instance directly from your browser.


Instances (1/2) Info
Connect
Instance state
Actions
Launch instances
Capture Moments Pro...
i-0871bc00b1d4ee001
Running
t2.micro
Capture-Moments1
i-01b78c9b7d945ec7c
Running
Change security groups
Get Windows password
Modify IAM role
Instance diagnostics
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
i-0871bc00b1d4ee001 (Capture Moments Project)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary
Instance ID
i-0871bc00b1d4ee001
Public IPv4 address
54.83.116.15 | open address
Private IPv4 addresses
172.31.31.13
IPv6 address
Instance state
Running
Public DNS
ec2-54-83-116-15.compute-1.amazonaws.com | open address

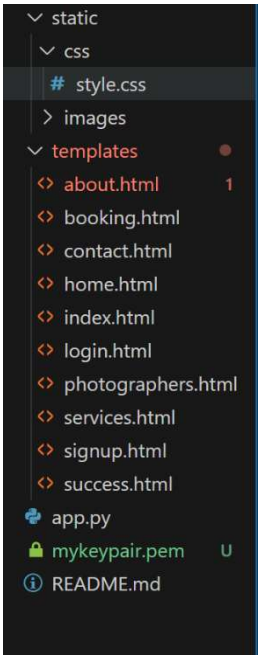

Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-0871bc00b1d4ee001 (Capture Moments Project)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
EC2DynamoDBAccessRole
Create new IAM role
Cancel
Update IAM role

- Now connect the EC2 with the files



Milestone 6. Deployment on EC2

- InstallSoftwareontheEC2Instance and copy the created .pem key in project folder



- Install the InstallPython3,Flask,andGit,On Amazon Linux 2:

- Run these following commands and install boto3,python3,flask and pip

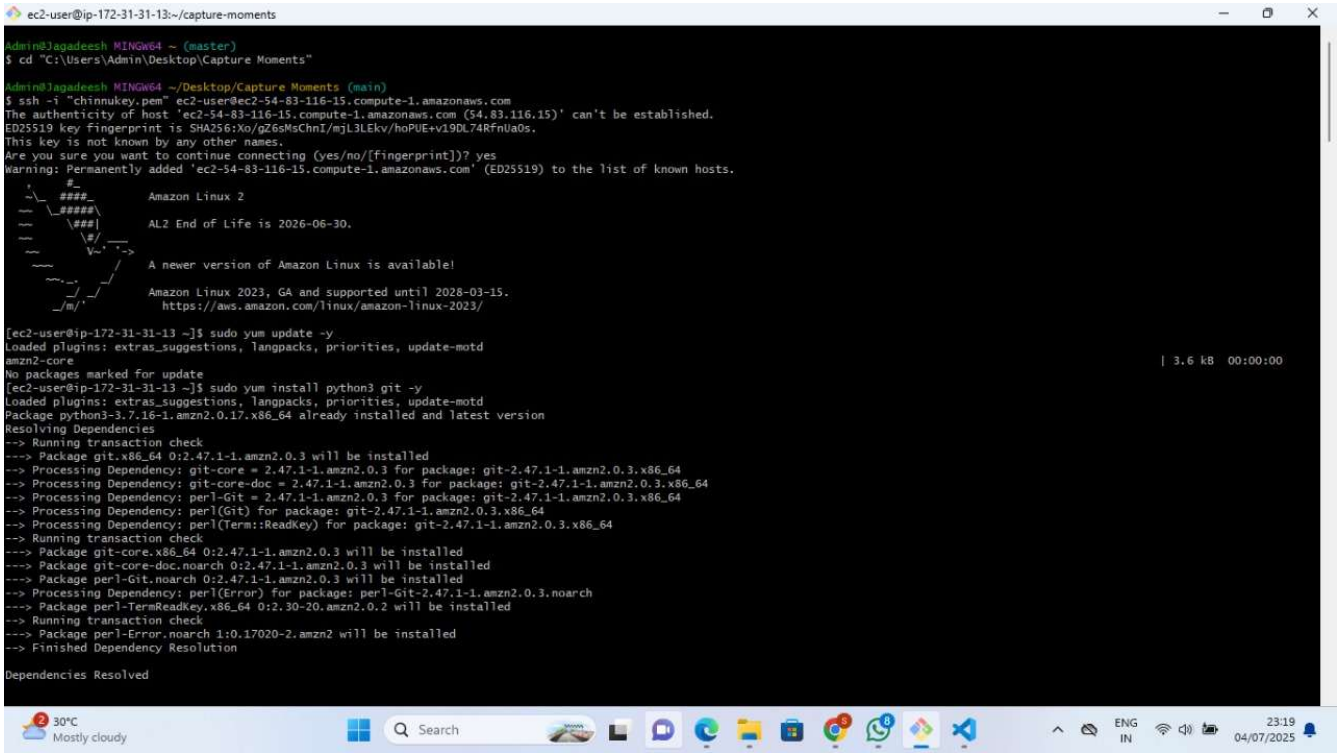
  sudo yum update -y

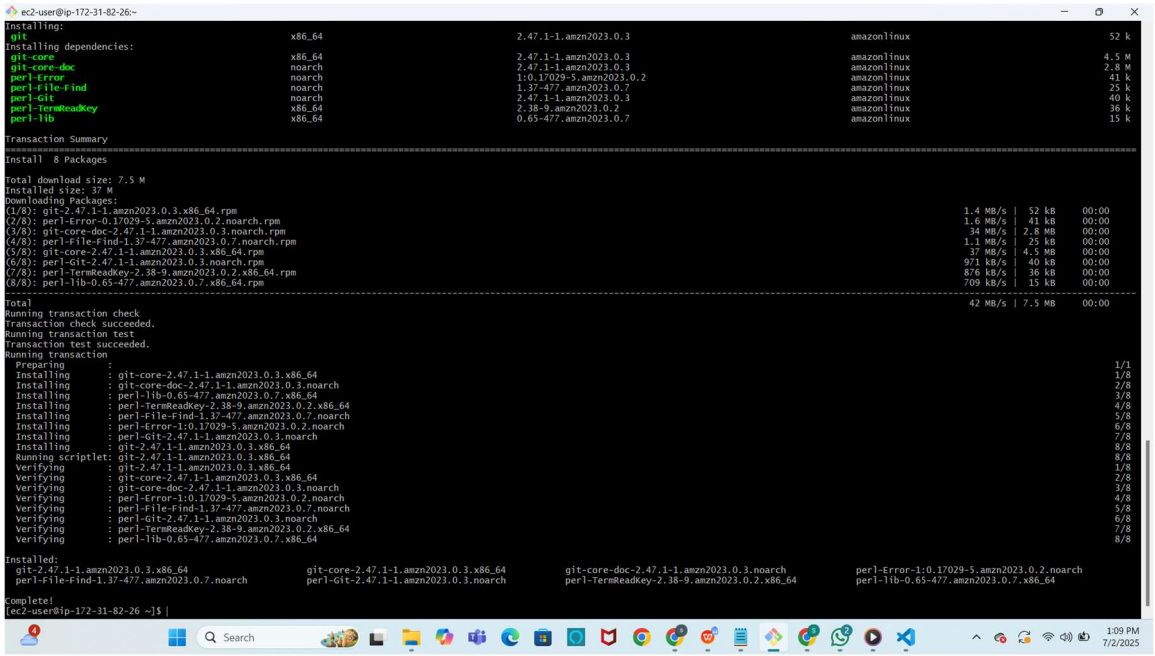
  sudo yum install python3 git -y

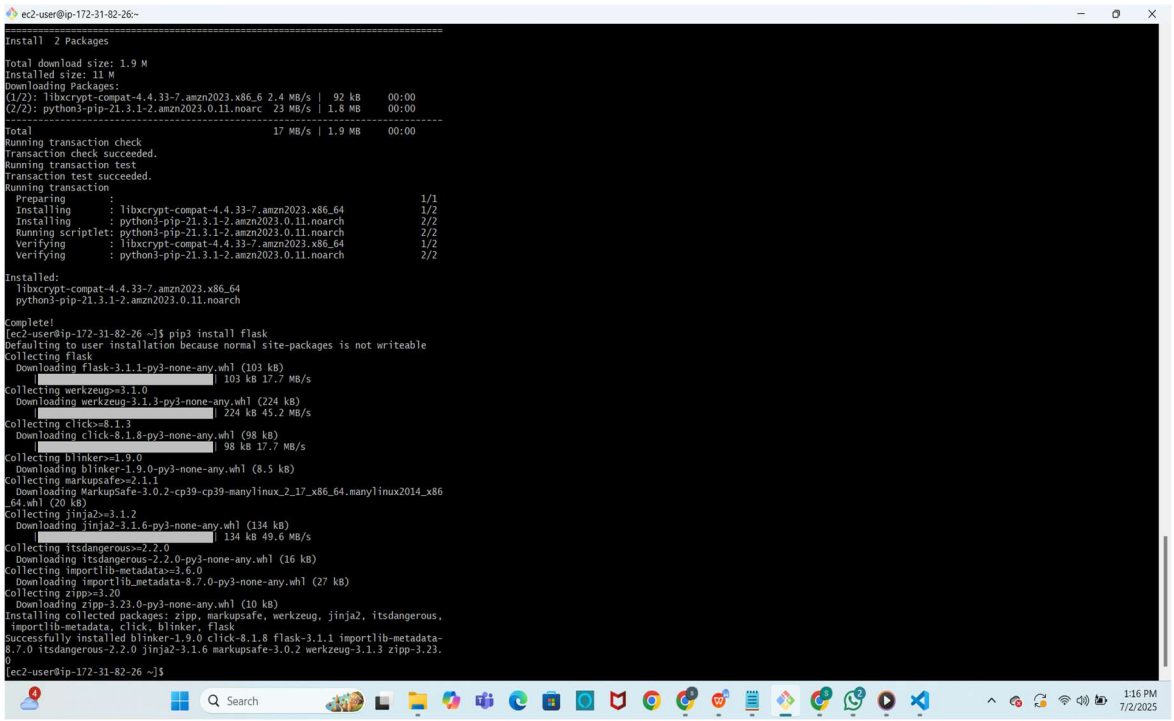
  sudo yum install python3-pip -y

  pip3 install flask

  pip3 install boto3




ec2-user@ip-172-31-82-26:~
Installing:
git x86_64 2.47.1-1.amzn2023.0.3 amazonlinux 52 k
Installing dependencies:
git-core x86_64 2.47.1-1.amzn2023.0.3 amazonlinux 4.5 M
git-core-doc noarch 2.47.1-1.amzn2023.0.3 amazonlinux 2.8 M
perl-Error noarch 1:0.17029-5.amzn2023.0.2 amazonlinux 41 k
perl-File-Find noarch 1.37-477.amzn2023.0.7 amazonlinux 25 k
perl-Git noarch 2.47.1-1.amzn2023.0.3 amazonlinux 40 k
perl-TermReadKey x86_64 2.38-9.amzn2023.0.2 amazonlinux 36 k
perl-lib x86_64 0.65-477.amzn2023.0.7 amazonlinux 15 k
Transaction Summary
Install 8 Packages
Total download size: 7.5 M
Installed size: 37 M
Downloading Packages:
(1/8): git-2.47.1-1.amzn2023.0.3.x86_64.rpm 1.4 MB/s | 52 kB 00:00
(2/8): perl-Error-0.17029-5.amzn2023.0.2.noarch.rpm 1.6 MB/s | 41 kB 00:00
(3/8): git-core-doc-2.47.1-1.amzn2023.0.3.noarch.rpm 34 MB/s | 2.8 MB 00:00
(4/8): perl-File-Find-1.37-477.amzn2023.0.7.noarch.rpm 1.1 MB/s | 25 kB 00:00
(5/8): git-core-2.47.1-1.amzn2023.0.3.x86_64.rpm 37 MB/s | 4.5 MB 00:00
(6/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm 971 kB/s | 40 kB 00:00
(7/8): perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64.rpm 876 kB/s | 36 kB 00:00
(8/8): perl-lib-0.65-477.amzn2023.0.7.x86_64.rpm 709 kB/s | 15 kB 00:00
Total 42 MB/s | 7.5 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing :
Installing : git-core-2.47.1-1.amzn2023.0.3.x86_64
Installing : git-core-doc-2.47.1-1.amzn2023.0.3.noarch
Installing : perl-lib-0.65-477.amzn2023.0.7.x86_64
Installing : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64
Installing : perl-File-Find-1.37-477.amzn2023.0.7.noarch
Installing : perl-Error-1:0.17029-5.amzn2023.0.2.noarch
Installing : perl-Git-2.47.1-1.amzn2023.0.3.noarch
Installing : git-2.47.1-1.amzn2023.0.3.x86_64
Running scriptlet: git-2.47.1-1.amzn2023.0.3.x86_64
Verifying : git-2.47.1-1.amzn2023.0.3.x86_64
Verifying : git-core-2.47.1-1.amzn2023.0.3.x86_64
Verifying : git-core-doc-2.47.1-1.amzn2023.0.3.noarch
Verifying : perl-Error-1:0.17029-5.amzn2023.0.2.noarch
Verifying : perl-File-Find-1.37-477.amzn2023.0.7.noarch
Verifying : perl-Git-2.47.1-1.amzn2023.0.3.noarch
Verifying : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64
Verifying : perl-lib-0.65-477.amzn2023.0.7.x86_64
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-172-31-82-26 ~]$
Search
1:09 PM
7/2/2025


ec2-user@ip-172-31-82-26:~
Installing 2 Packages
Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libxcrypt-compat-4.4.33-7.amzn2023.x86_6 2.4 MB/s | 92 kB 00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.11.noarc 23 MB/s | 1.8 MB 00:00
Total 17 MB/s | 1.9 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : libxcrypt-compat-4.4.33-7.amzn2023.x86_64 1/2
Installing : python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2
Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2
Verifying : libxcrypt-compat-4.4.33-7.amzn2023.x86_64 1/2
Verifying : python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2
Installed:
libxcrypt-compat-4.4.33-7.amzn2023.x86_64
python3-pip-21.3.1-2.amzn2023.0.11.noarch
Complete!
[ec2-user@ip-172-31-82-26 ~]$ pip3 install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
Downloading flask-3.1.1-py3-none-any.whl (103 kB)
103 kB 17.7 MB/s
Collecting werkzeug>=3.1.0
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
224 kB 45.2 MB/s
Collecting click>=8.1.3
Downloading click-8.1.8-py3-none-any.whl (98 kB)
98 kB 17.7 MB/s
Collecting blinker>=1.9.0
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting markupsafe>=2.1.1
Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting jinja2>=3.1.2
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
134 kB 49.6 MB/s
Collecting itsdangerous>=2.2.0
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting importlib-metadata>=3.6.0
Downloading importlib_metadata-8.7.0-py3-none-any.whl (27 kB)
Collecting zipp>=3.20
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-172-31-82-26 ~]$
Search
1:16 PM
7/2/2025

- Clone Your Flask Project from GitHub

Run:‘gitclone: https://github.com/Harikagolusu/CaptureMoments-AWS.git ‘

Note:changeyour-github-usernameandyour-repository-namewithyourcredentials

To navigate to the project directory , run the following command:

cd directory name

Once inside the project directory ,configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

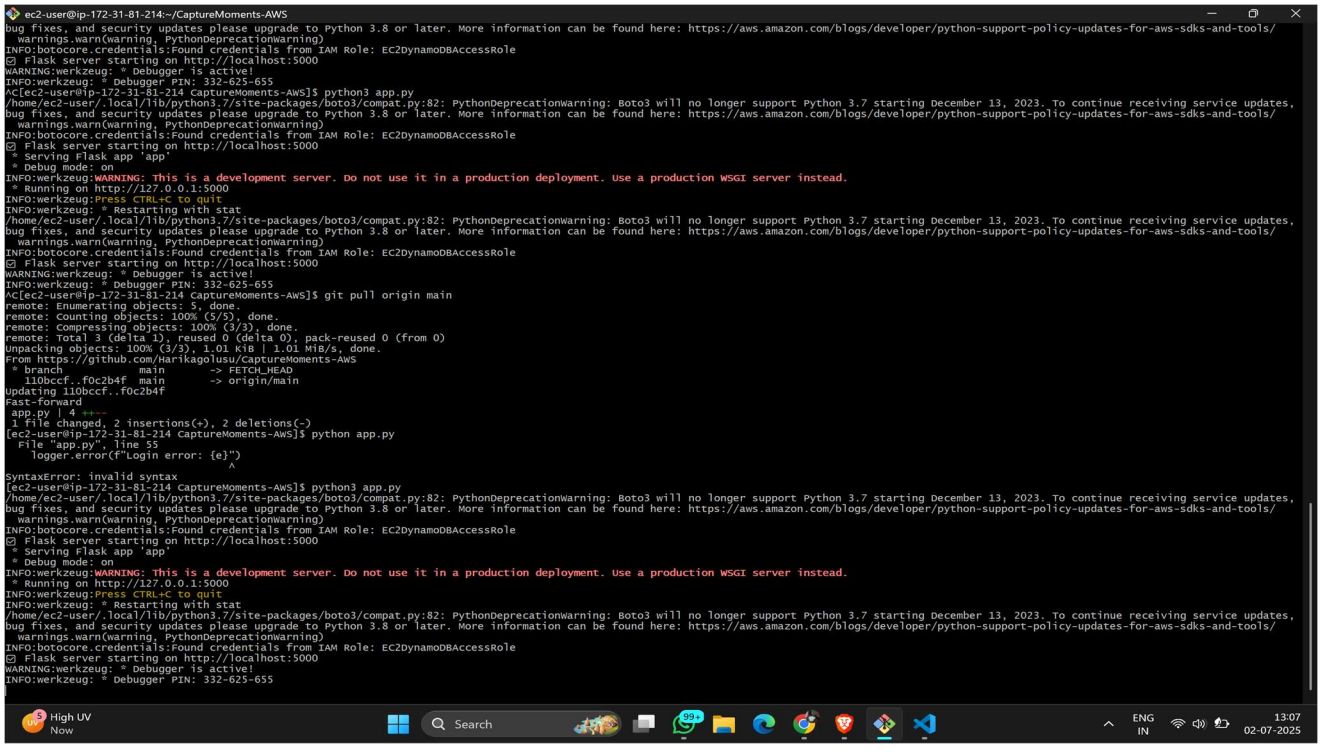
python3 app.py

Verify the Flask app is

running:

http://54.83.116.15:5000

- Run the Flask app on the EC2 instance

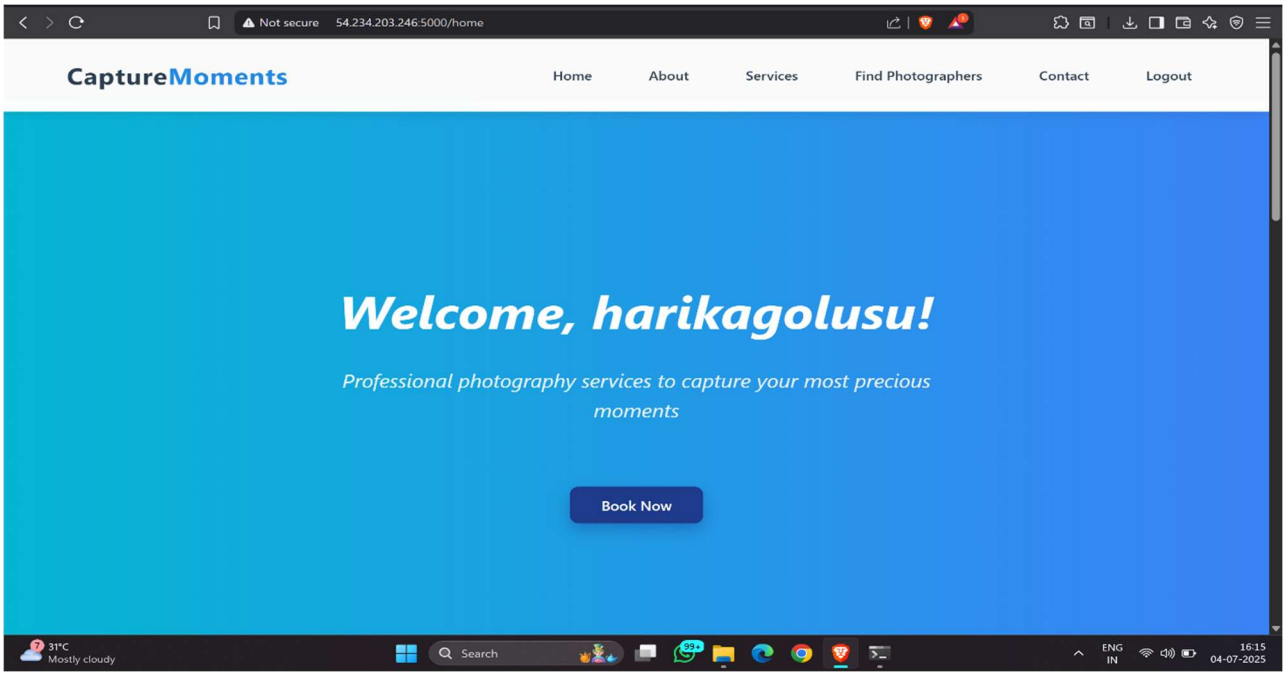


Milestone 7. Testing and Deployment

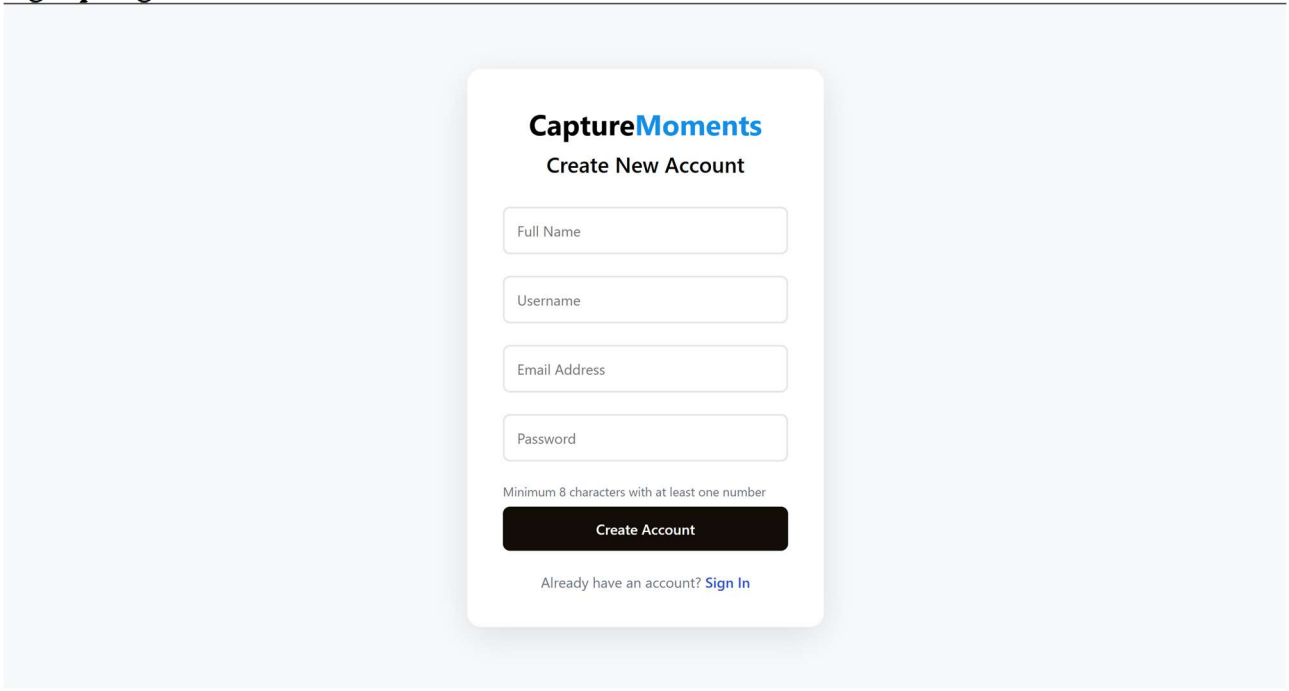
Conduct functional testing to verify user signup, login, home, booking, photographers,

Contact us , about us pages.

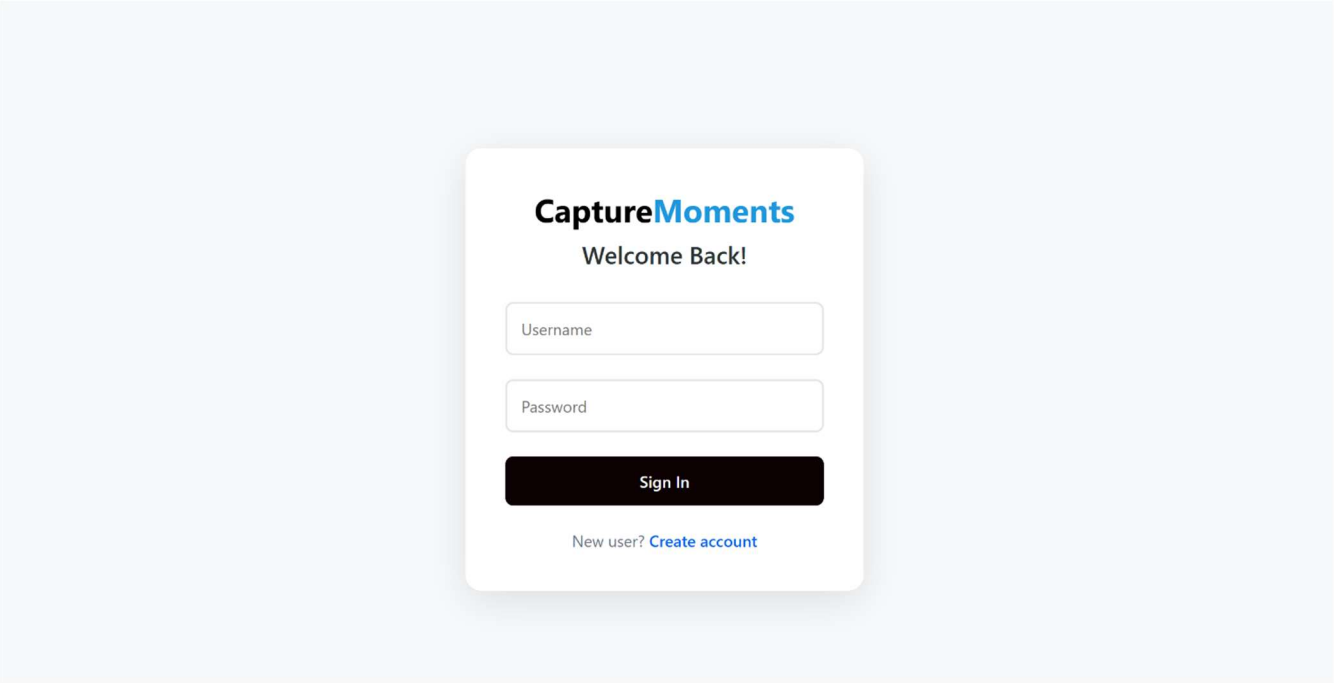
Home Page :



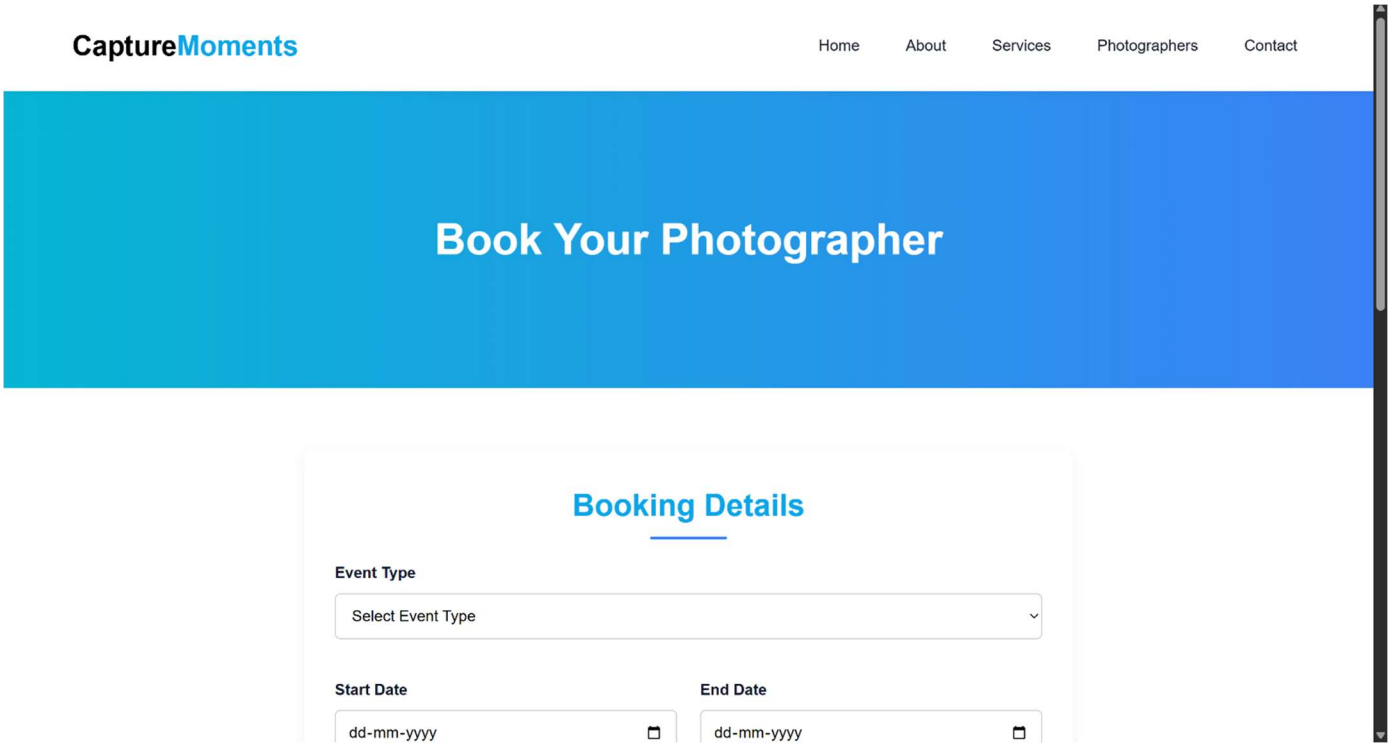
Signup Page :



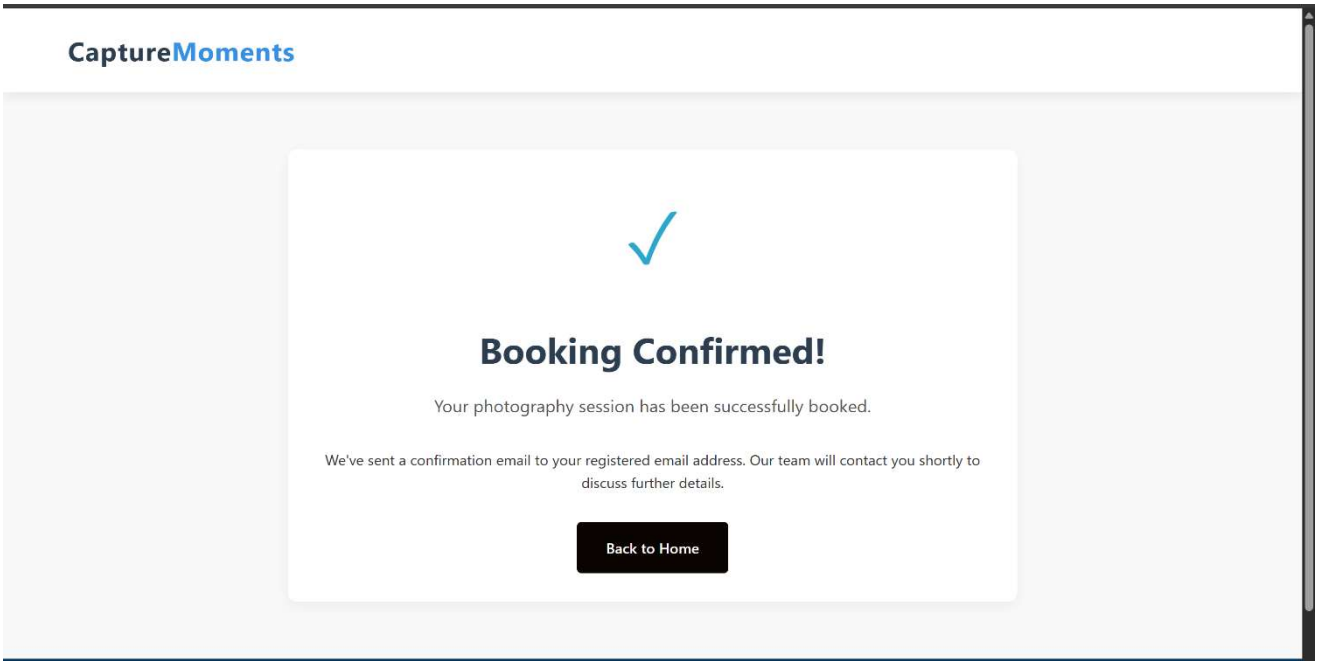
Login Page



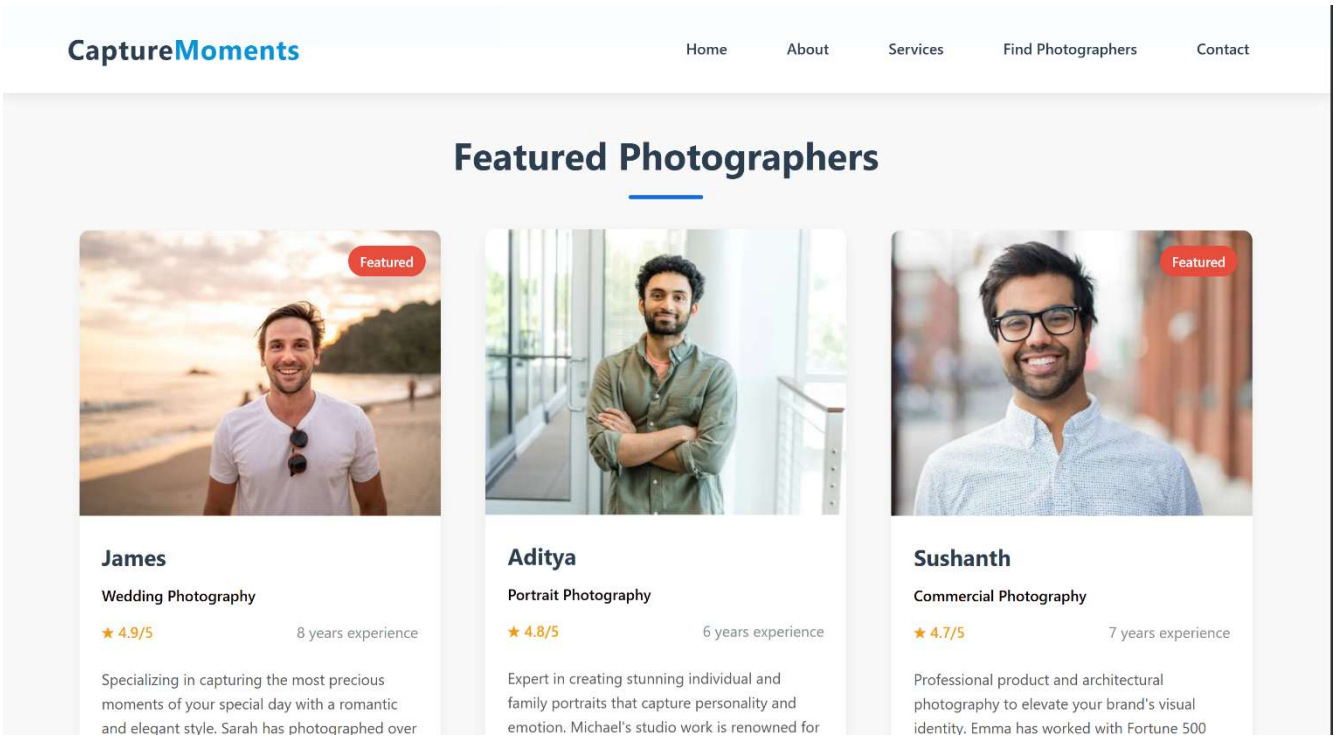
Booking Page



Confirmation Page :



Photographers page:



Conclusion:

The Capture Moments platform has been successfully developed and deployed using a reliable cloud-based architecture tailored for seamless event photography management. Leveraging AWS services such as EC2 for hosting and DynamoDB for efficient data storage, the system ensures high availability, scalability, and quick access to media content. Designed to address the common challenges of disorganized photo sharing and limited storage, the platform offers a centralized and user-friendly interface for uploading, organizing, and accessing event photos. Flask powers the backend operations, enabling smooth workflows for user authentication, album creation, and secure photo management. Throughout the testing phase, all core functionalities—from user registration to photo uploads and album navigation—were validated for performance and reliability. In conclusion, Capture Moments showcases the potential of cloud-native solutions to transform traditional event experiences by simplifying the way memories are captured, stored, and shared. This project stands as a modern, efficient response to the evolving needs of digital photo management in dynamic event settings.