

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.

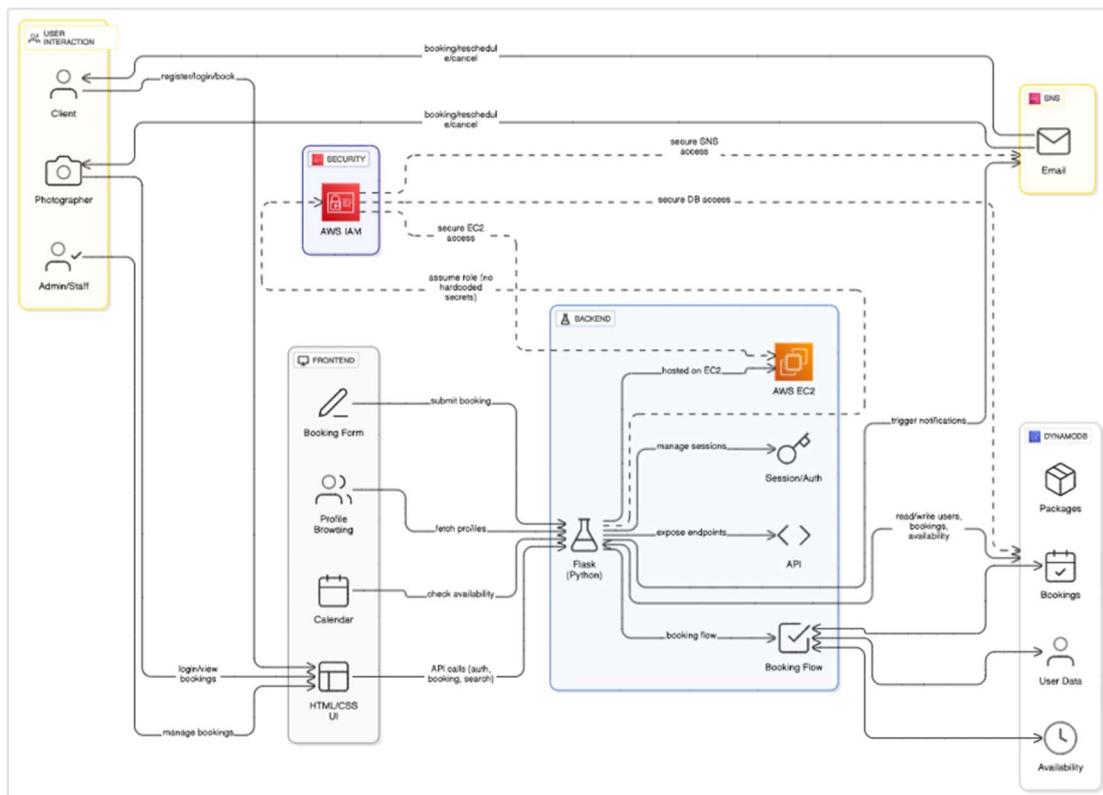
Scenario 2: Centralized Booking Management

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.

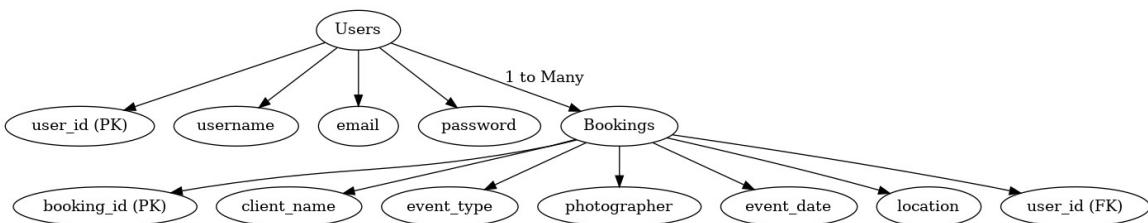
Scenario 3: Easy Access to Library Resources

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.

AWS ARCHITECTURE



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

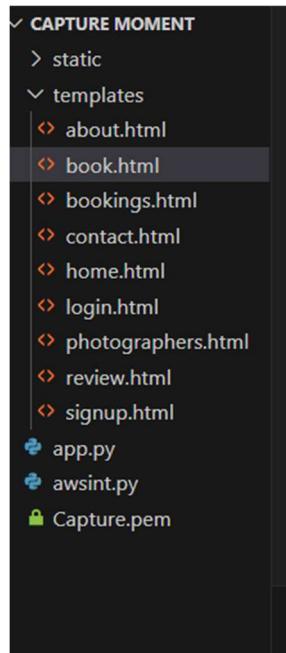
Run the Flask App

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

```
1  from flask import Flask, render_template, request, redirect, url_for, flash, session
2  from werkzeug.security import generate_password_hash, check_password_hash
3  from datetime import datetime
4  import logging
5  import boto3
6  import uuid
7  import os
8  from botocore.exceptions import ClientError
9
```

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()
                }
            )
        except Exception as e:
            flash(str(e), 'error')
            return redirect(url_for('signup'))
    else:
        return render_template('signup.html')

```

```

        '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

```
@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', 'package']
            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'],
                ...
            }

            # 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
        except Exception as e:
            logger.error(f"Error processing booking: {e}")
            flash('An error occurred while processing your booking. Please try again.', 'error')
            return redirect(url_for('booking', event=event_type))
    else:
        return render_template('booking.html', event_type=event_type)
```

```
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')

```

```

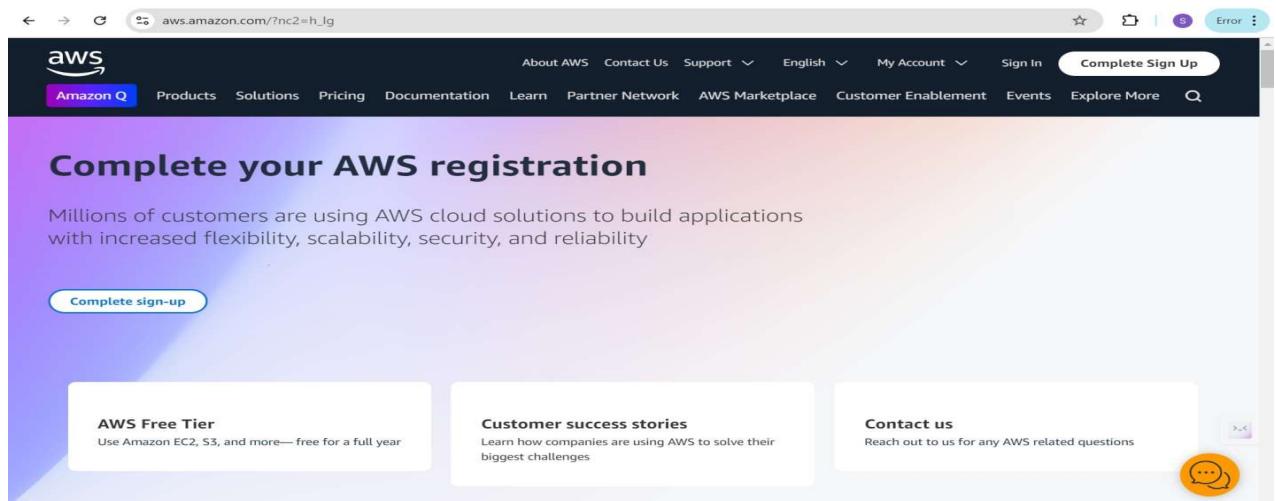
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

Milestone 2. AWS Account Setup and Login

2.1 Set up an AWS account if not already done.

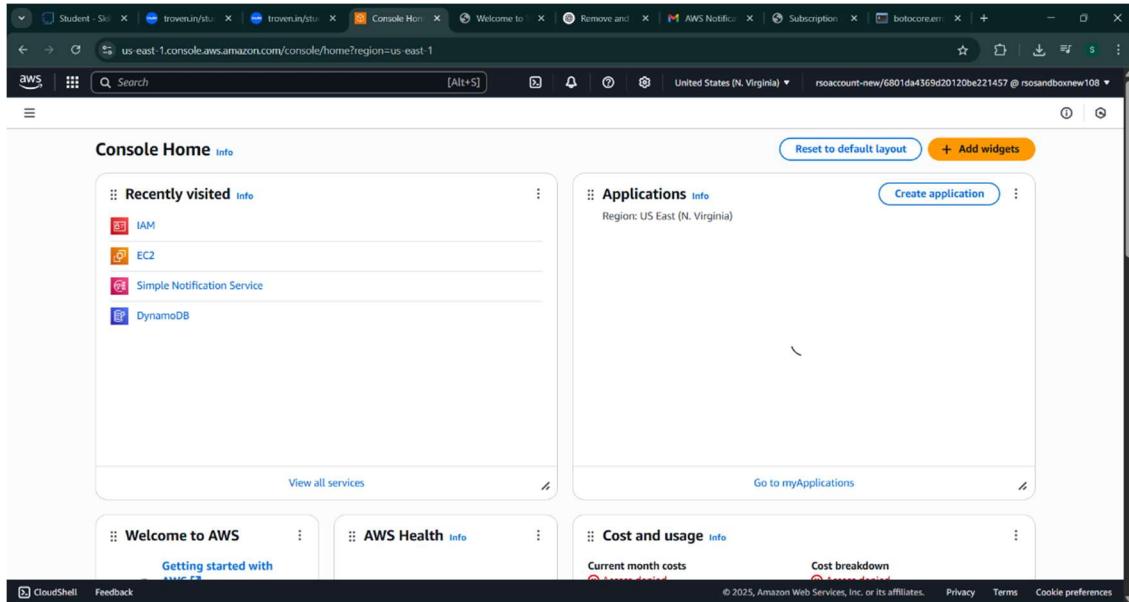
Signup for an AWS account and configure billing settings.



The screenshot shows the AWS Complete Sign Up page. At the top, there is a navigation bar with links for About AWS, Contact Us, Support, English, My Account, Sign In, and Complete Sign Up. Below the navigation bar, the main heading is "Complete your AWS registration". A sub-headline states: "Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability". There is a "Complete sign-up" button. On the left, there is a box for "AWS Free Tier" with the subtext "Use Amazon EC2, S3, and more—free for a full year". In the center, there is a box for "Customer success stories" with the subtext "Learn how companies are using AWS to solve their biggest challenges". On the right, there is a "Contact us" section with the subtext "Reach out to us for any AWS related questions" and an orange "Ask a question" button.

2.2 Log in to the AWS Management Console

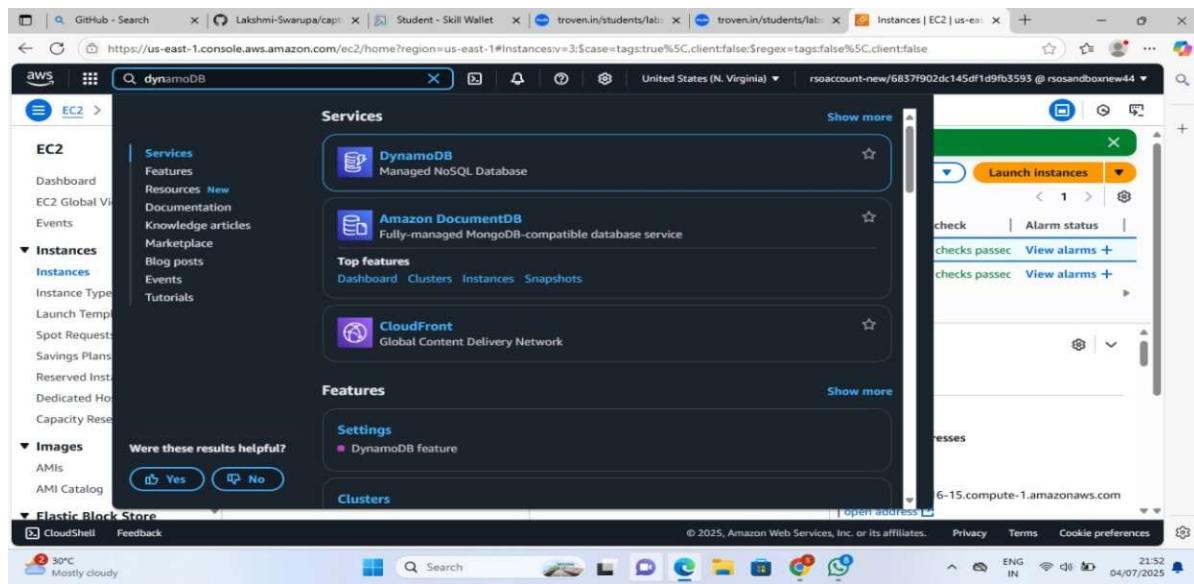
After setting up your account ,login to the [AWSManagementConsole](#)



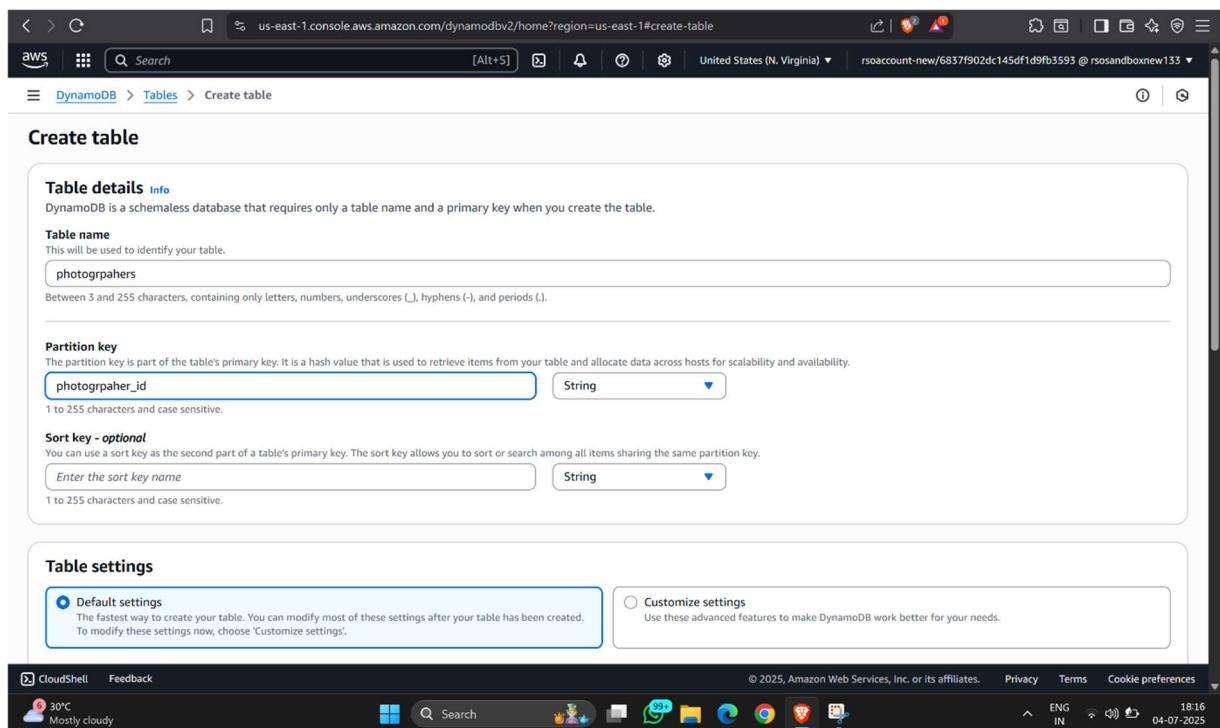
Milestone 3. DynamoDB Database Creation and Setup

3.1 Create a DynamoDB Table.

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



The screenshot shows the AWS EC2 Services page. The search bar at the top contains 'dynamoDB'. Under the 'Services' section, 'DynamoDB' is highlighted as a Managed NoSQL Database. Other services like Amazon DocumentDB and CloudFront are also listed. On the right side, there is a modal window titled 'Launch Instances' with options for selecting instance types and regions.

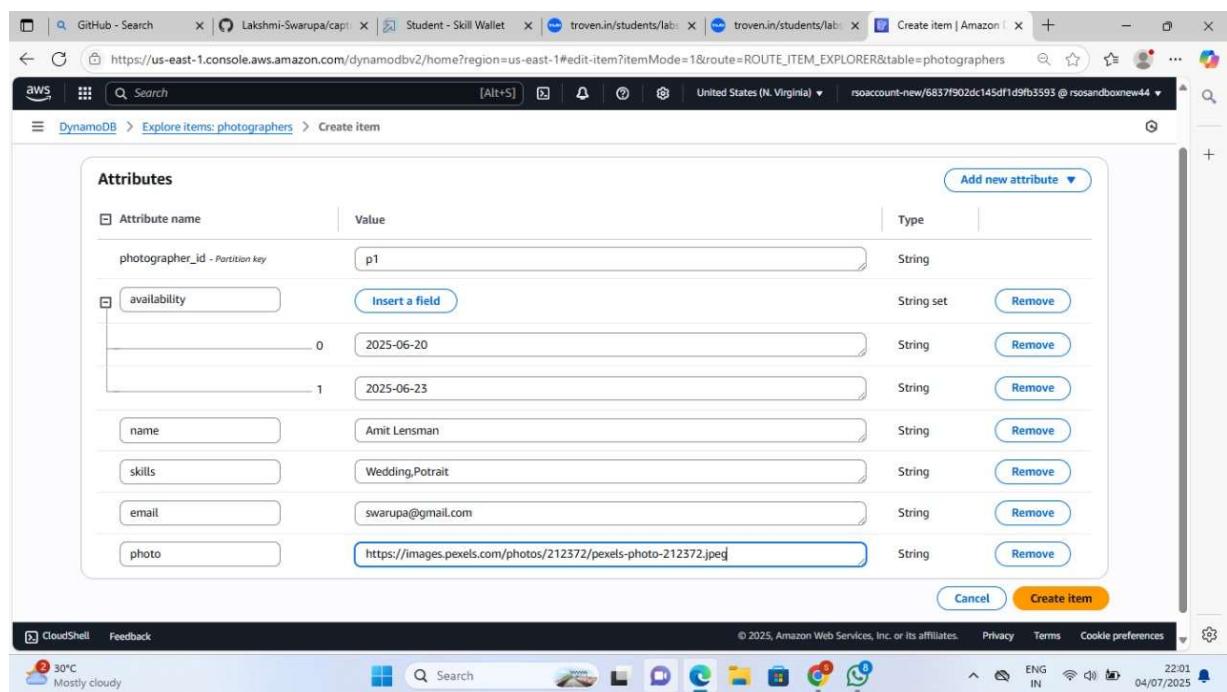


The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The first step, 'Table details', is active. It requires a table name ('photographers') and a partition key ('photographer_id'). Both fields are set to 'String'. The 'Sort key - optional' field is also present but empty. Below these fields, there are two radio button options: 'Default settings' (selected) and 'Customize settings'. The 'Default settings' option is described as the fastest way to create the table. The 'Customize settings' option is described as using advanced features to make DynamoDB work better for specific needs.

The screenshot shows the AWS DynamoDB console interface. The left sidebar includes links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings, and DAX. The main content area displays a list of three tables: "photographers", "photography_bookings", and "photography_users". Each table row includes columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, and Deletion protection. The "photographers" table has a partition key of "photographer_id" and no sort key. The "photography_bookings" table has a partition key of "booking_id" and no sort key. The "photography_users" table has a partition key of "username" and no sort key. The "Deletion protection" column for all three tables shows "Off".

| Name | Status | Partition key | Sort key | Indexes | Replication Regions | Deletion protection |
|----------------------|--------|----------------------|----------|---------|---------------------|---------------------|
| photographers | Active | photographer_id (\$) | - | 0 | 0 | Off |
| photography_bookings | Active | booking_id (\$) | - | 0 | 0 | Off |
| photography_users | Active | username (\$) | - | 0 | 0 | Off |

3.2 Configure Attributes for photographers and Book Requests



The screenshot shows the AWS DynamoDB 'Create item' dialog. The 'Attributes' section is displayed, listing the following fields:

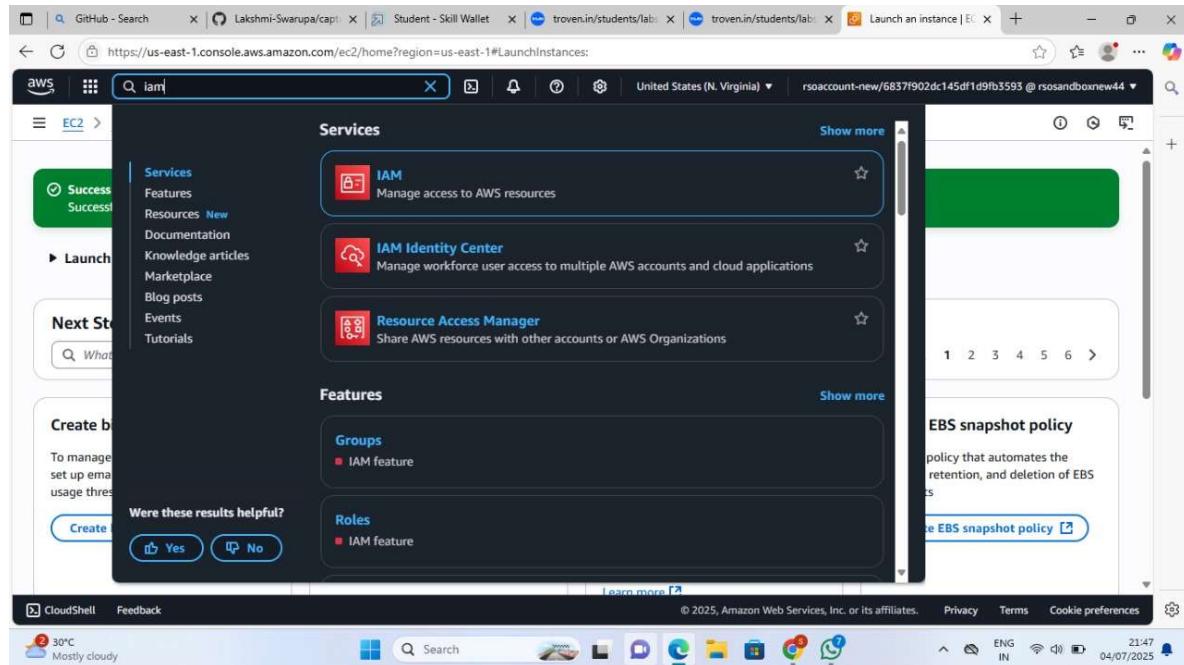
| Attribute name | Value | Type |
|---------------------------------|---|------------|
| photographer_id - Partition key | p1 | String |
| availability | Insert a field | String set |
| | 0 2025-06-20 | String |
| | 1 2025-06-23 | String |
| name | Amit Lensman | String |
| skills | Wedding,Potrait | String |
| email | swarupa@gmail.com | String |
| photo | https://images.pexels.com/photos/212372/pexels-photo-212372.jpeg | String |

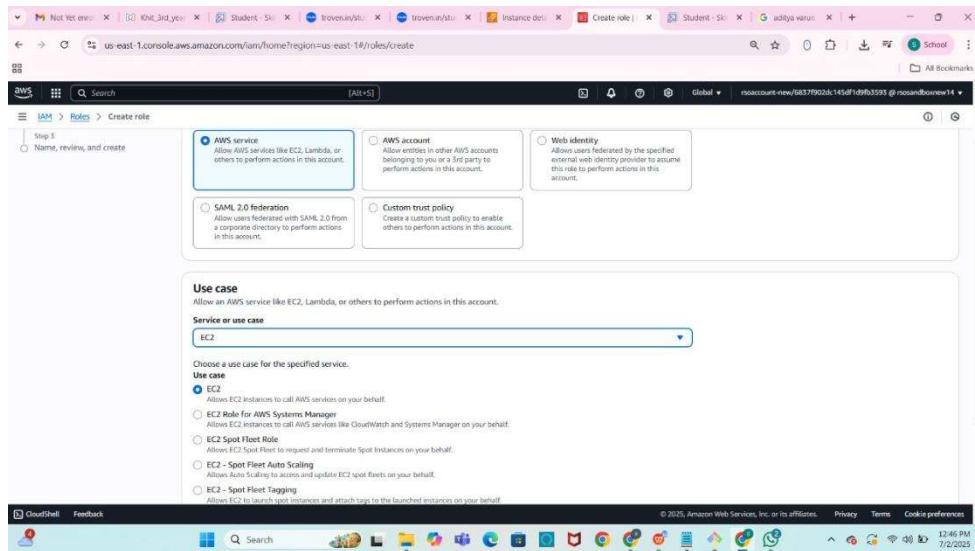
At the bottom right of the dialog are 'Cancel' and 'Create item' buttons.

The browser toolbar at the top includes tabs for GitHub, Lakshmi-Swarupa, Student - Skill Wallet, troven.in/students/lab:, troven.in/students/lab:, Create item | Amazon, and several others. The address bar shows the URL for the AWS DynamoDB console. The operating system taskbar at the bottom shows the date (04/07/2025), time (22:01), battery level (ENG IN), and system icons.

Milestone 4. IAM Role Setup

4.1 Create IAM Role

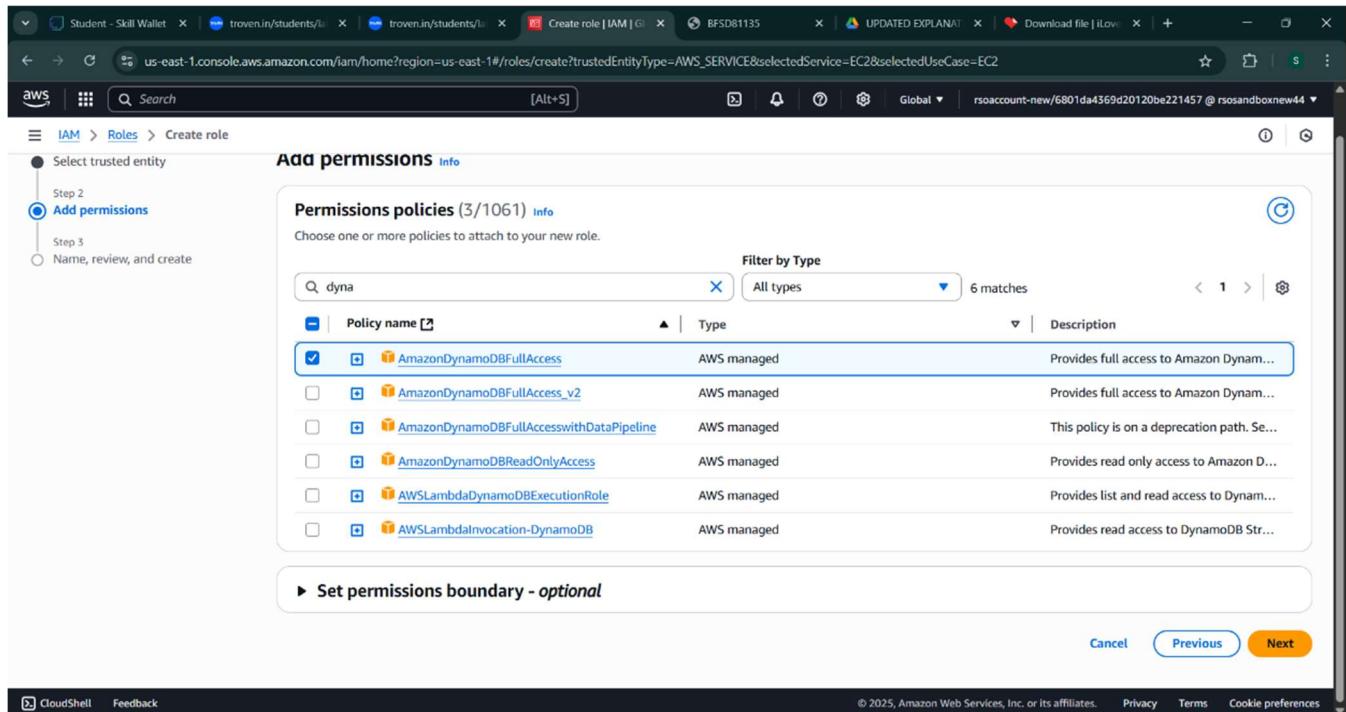


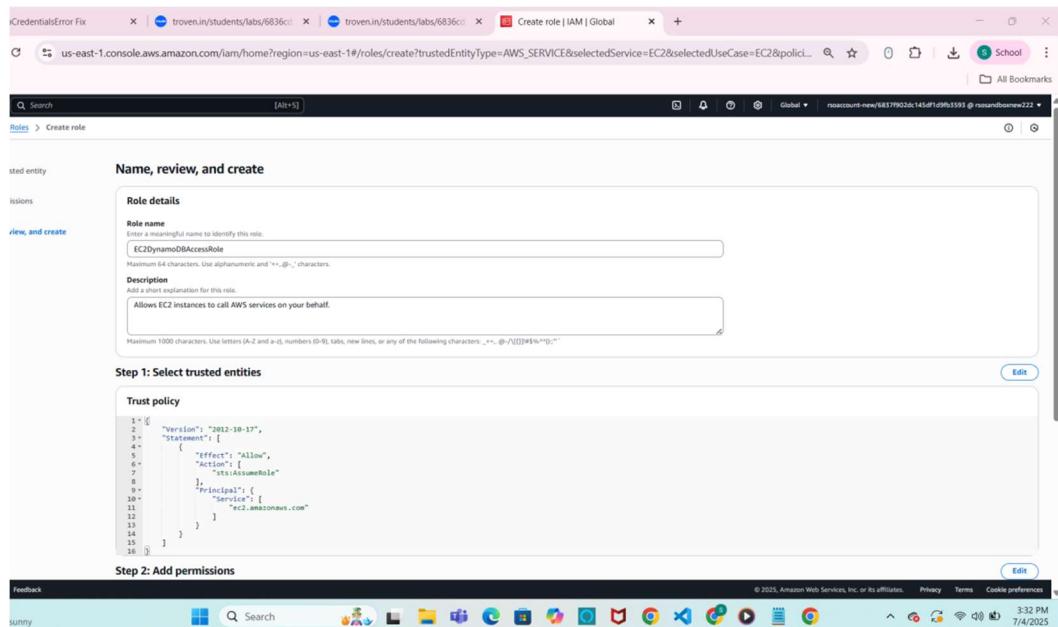


4.2 Attach Policies

Attach the following policies to the role:

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





The screenshot shows the 'Create role' wizard in the AWS IAM console. In Step 1: Select trusted entities, a trust policy is defined:

```

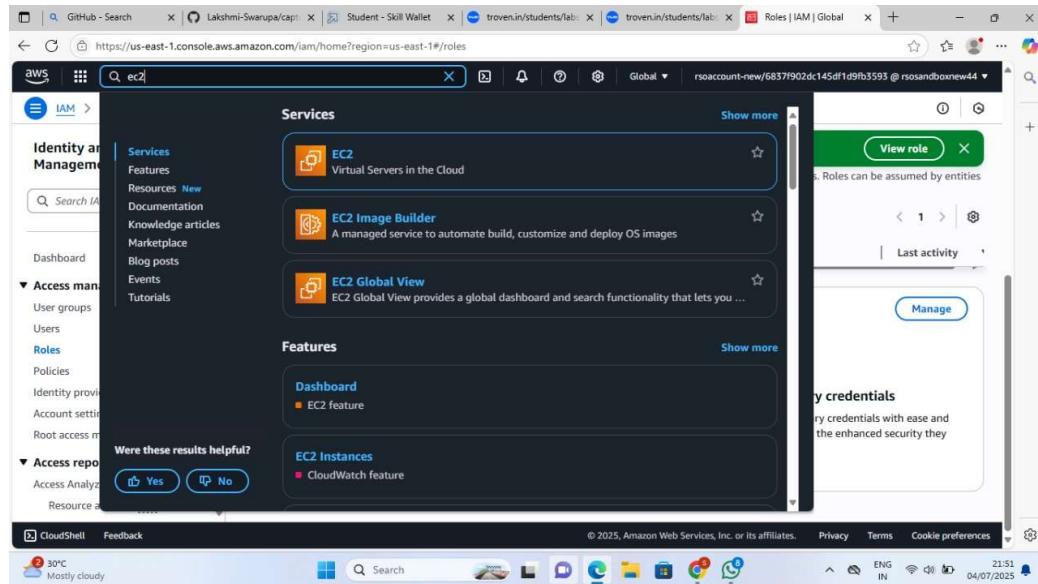
1 x [
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "sts:AssumeRole",
7       "Principal": [
8         {
9           "Service": [
10             "ec2.amazonaws.com"
11           ]
12         }
13       ]
14     }
15   ]
16 ]

```

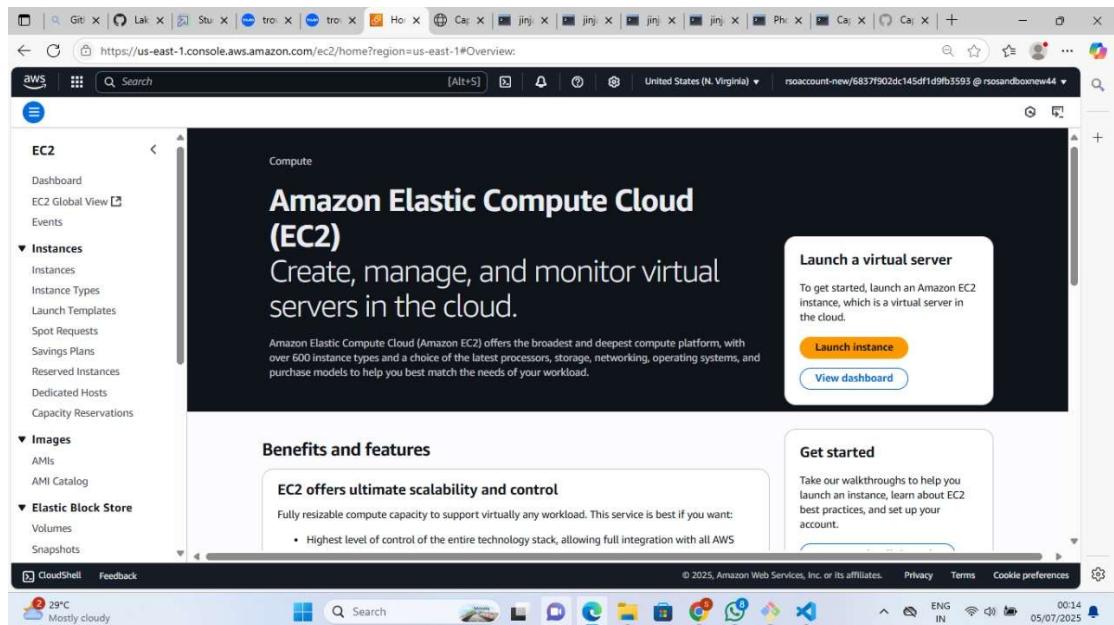
In Step 2: Add permissions, there is a placeholder for permissions.

Milestone 5. EC2 Instance Setup

5.1 Launch an EC2 instance to host the Flask application.

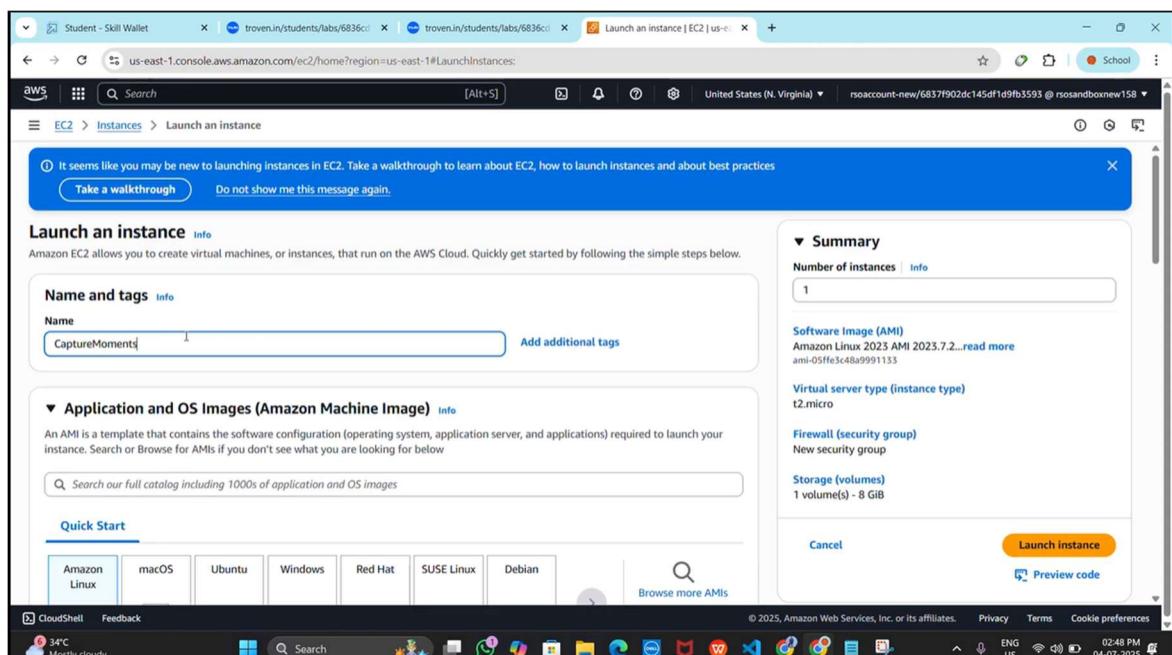


The screenshot shows the AWS EC2 service page. The main content area displays the 'EC2' feature under 'Virtual Servers in the Cloud'.

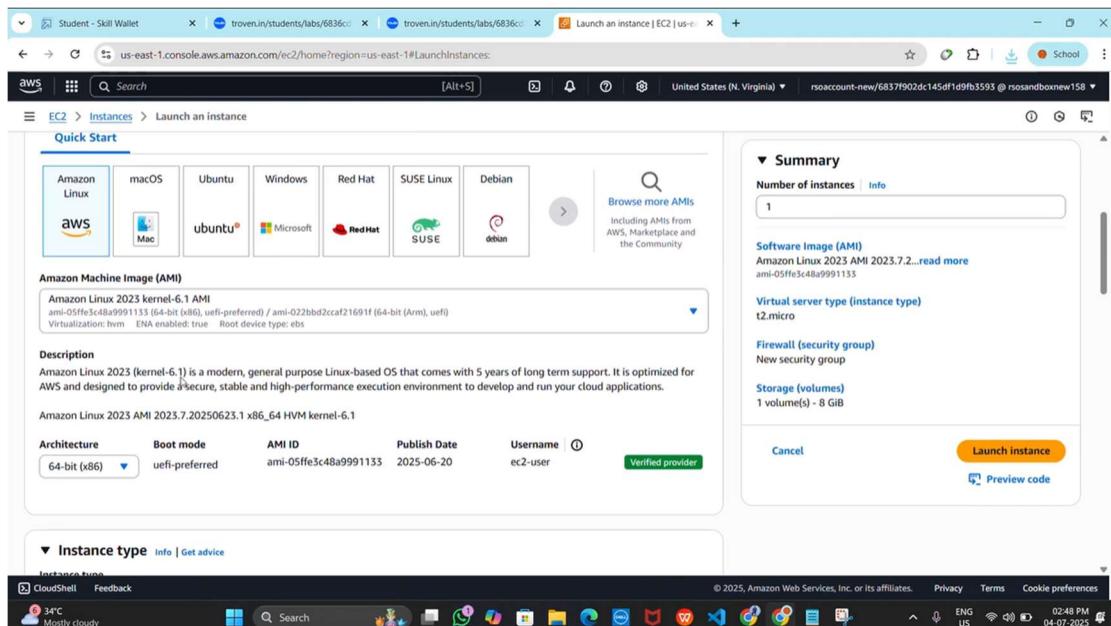


The screenshot shows the AWS EC2 home page. On the left, there's a sidebar with navigation links for EC2, Instances, Images, and Elastic Block Store. The main content area features a large heading "Amazon Elastic Compute Cloud (EC2)" and a sub-section "Create, manage, and monitor virtual servers in the cloud." Below this, there's a "Benefits and features" section with a box titled "EC2 offers ultimate scalability and control". To the right, there's a "Get started" section with a "Launch a virtual server" box containing a "Launch instance" button and a "View dashboard" link. At the bottom of the main content area, there's a "CloudShell" and "Feedback" link. The status bar at the bottom shows weather information (29°C, Mostly cloudy), system icons, and the date/time (05/07/2025).

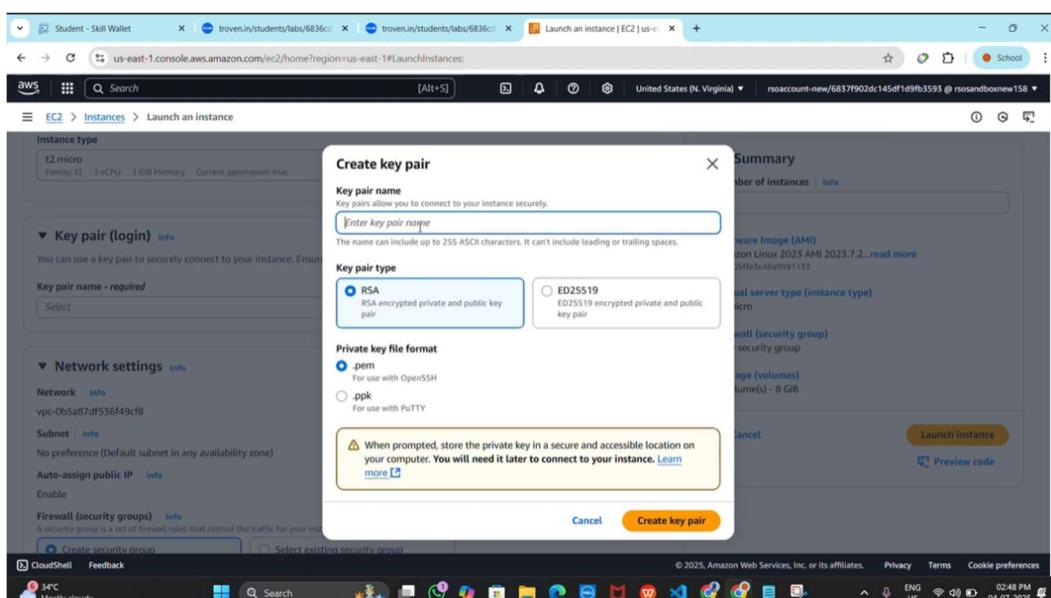
- Click on Launch instance to launch EC2 instance

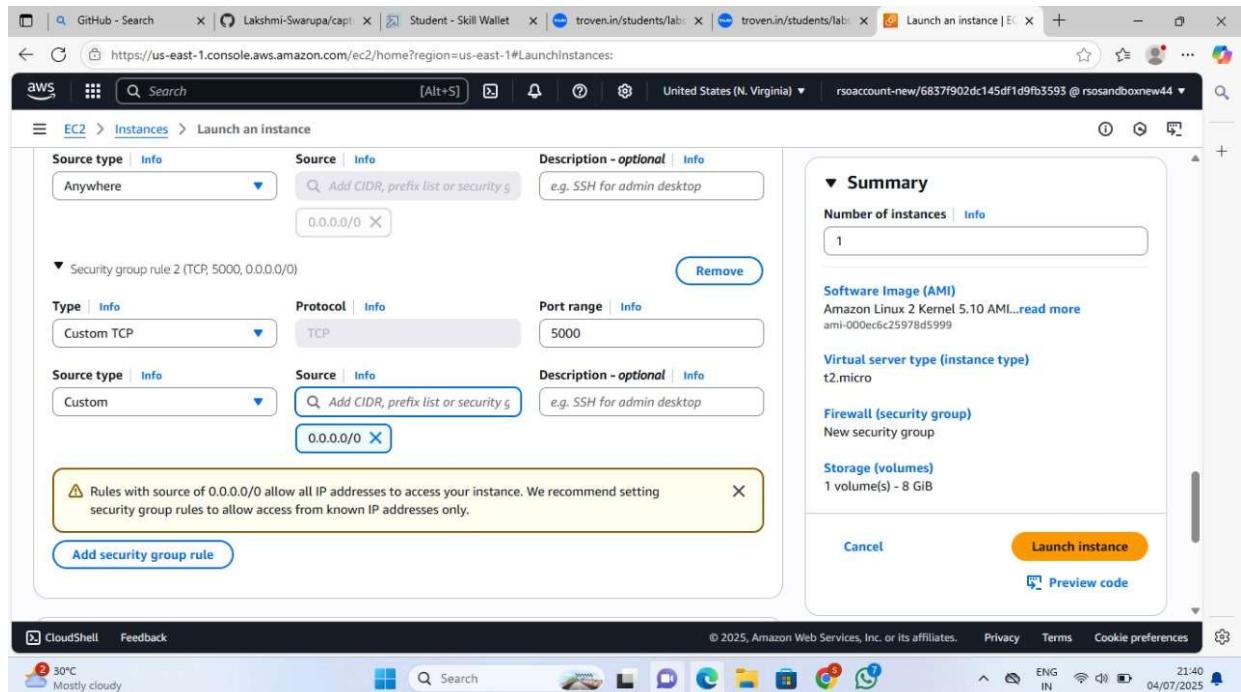


The screenshot shows the "Launch an instance" wizard. Step 1: Set instance details. It includes fields for Name and tags (with "CaptureMoments" entered), Application and OS Images (Amazon Machine Image) (with "Amazon Linux" selected), and Quick Start (with various operating system options like Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian). On the right, there's a "Summary" section showing 1 instance, Software Image (AMI) as Amazon Linux 2023 AMI 2023.7.2..., Virtual server type as t2.micro, Firewall as New security group, and Storage (volumes) as 1 volume(s) - 8 GiB. There are "Cancel", "Launch instance", and "Preview code" buttons.

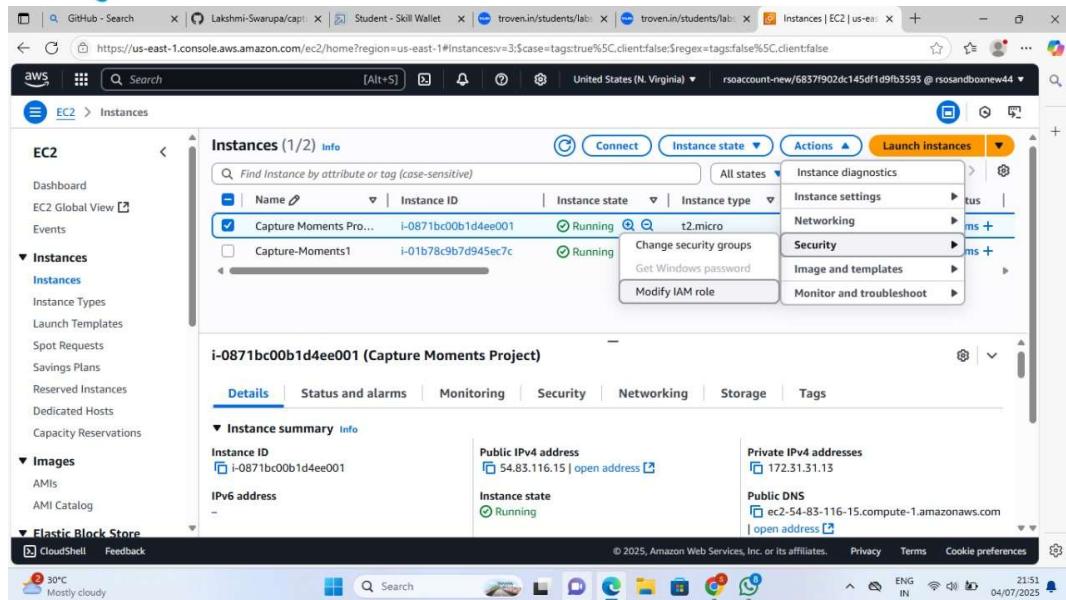


- Create a new key pair and save it with .pem extension





- 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 **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2InstanceConnect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.



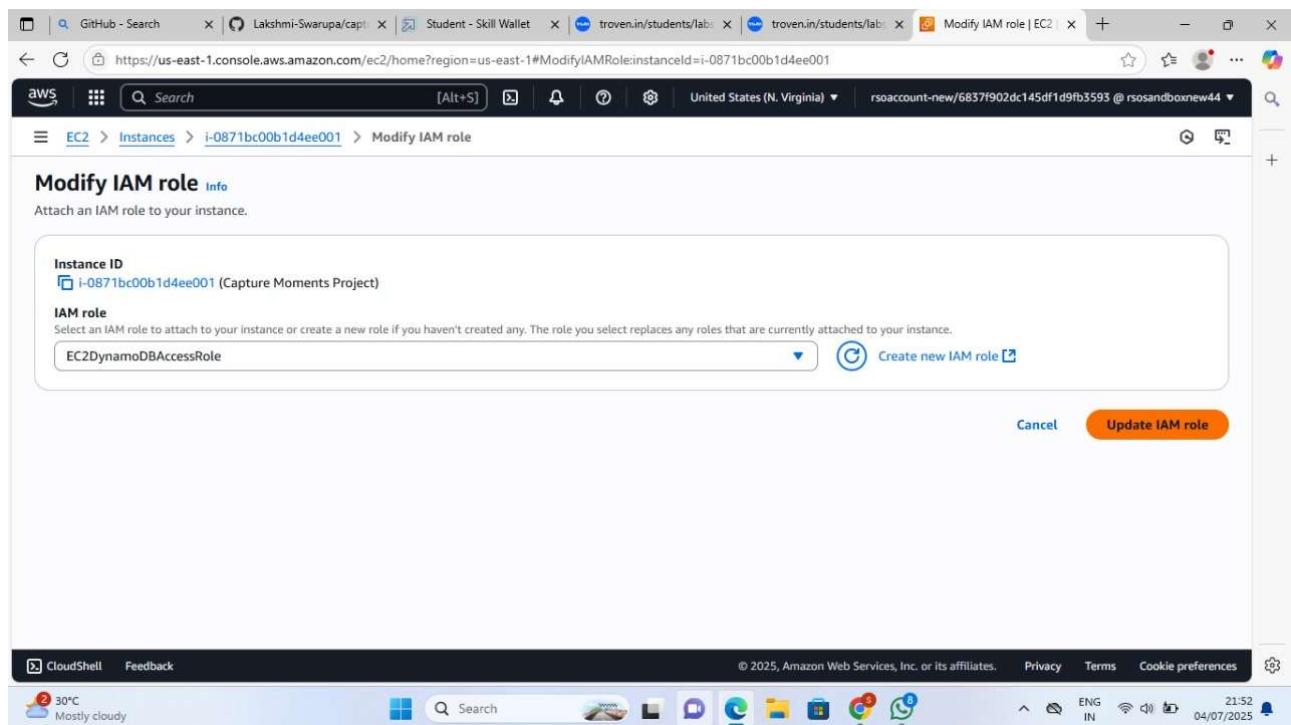
Instances (1/2) Info

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 | Private IPv4 addresses: 172.31.31.13
 Instance state: Running | Public DNS: ec2-54-83-116-15.compute-1.amazonaws.com



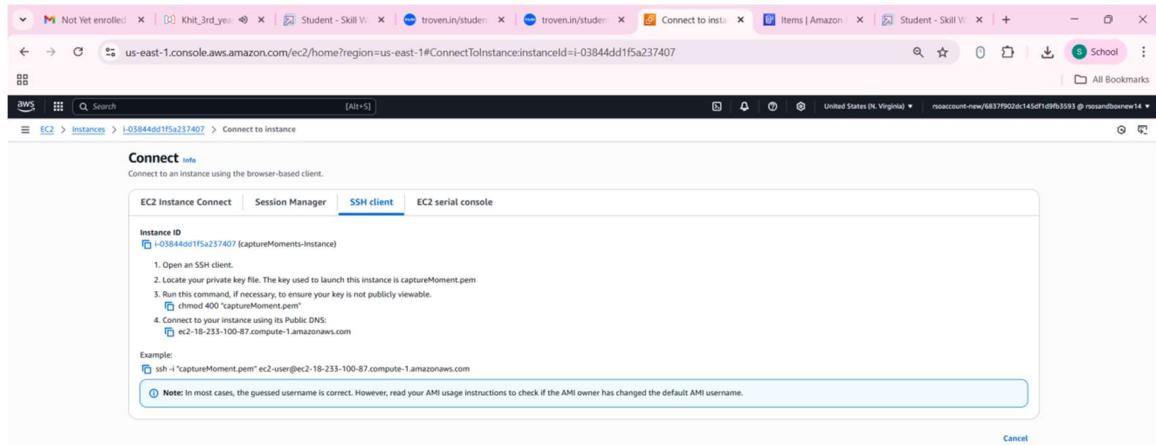
Modify IAM role

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

- Now connect the EC2 with the files



Milestone 6. Deployment on EC2

- Install Software on the EC2 Instance and copy the created .pem key in project folder

```

static
  css
    # style.css
  images
  templates
    about.html
    booking.html
    contact.html
    home.html
    index.html
    login.html
    photographers.html
    services.html
    signup.html
    success.html
  app.py
  mykeypair.pem
  README.md

```

- Install the

Install Python3, Flask, and Git, 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-172-31-31-13:~/capture-moments
$ cd "C:/Users/Admin/Desktop/Capture Moments"
$ ssh -i "Chinmayi.pem" ec2-user@ec2-54-83-116-15.compute-1.amazonaws.com
The authenticity of host 'ec2-54-83-116-15.compute-1.amazonaws.com (54.83.116.15)' can't be established.
ED25519 key fingerprint is SHA256:Xo/gZ6sMsChnI/mjL3LEkv/h0PUE+v19DL74RFnUa0s.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-83-116-15.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

# 
# Amazon Linux 2
# AL2 End of Life is 2026-06-30.
# A newer version of Amazon Linux is available!
# Amazon Linux 2023, GA and supported until 2028-03-15.
# https://aws.amazon.com/linux/amazon-linux-2023/
[m/]

[ec2-user@ip-172-31-31-13 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
No packages marked for update
[ec2-user@ip-172-31-31-13 ~]$ sudo yum install python3 git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package python3-3.7.16-1.amzn2.0.17.x86_64 already installed and latest version
Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.47.1-1.amzn2.0.3 will be installed
--> Processing Dependency: git-core = 2.47.1-1.amzn2.0.3 for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl-Git = 2.47.1-1.amzn2.0.3 for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Git) for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.47.1-1.amzn2.0.3.x86_64
--> Running transaction check
--> Package git-core.x86_64 0:2.47.1-1.amzn2.0.3 will be installed
--> Package git-core-doc.noarch 0:2.47.1-1.amzn2.0.3 will be installed
--> Package perl-Git.noarch 0:2.47.1-1.amzn2.0.3 will be installed
--> Processing Dependency: perl(Error) for package: perl-Git-2.47.1-1.amzn2.0.3.noarch
--> Package perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2 will be installed
--> Running transaction check
--> Package perl-Error.noarch 1:0.17020-2.amzn2 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

Transaction Summary
  0 updated
  0 new
  0 removed
  0 downgraded
  0 total

[ 3.6 kB  00:00:00

[ec2-user@ip-172-31-31-13 ~]$
```



```
ec2-user@ip-172-31-82-26:~$ 
Installing: git x86_64 2.47.1-1.amzn2023.0.3
Installing dependencies: git-core x86_64 2.47.1-1.amzn2023.0.3
git-core-doc noarch 2.47.1-1.amzn2023.0.3
perl-error noarch 1.0.17029-5.amzn2023.0.2
perl-file-find noarch 1.37-477.amzn2023.0.7
perl-term-readkey noarch 2.38-9.amzn2023.0.2
perl-lib noarch 0.65-477.amzn2023.0.7
Transaction Summary
Install 8 Packages
Total download size: 7.5 M
Installed size: 37 M
Downloading Packages:
(1/8): git-core-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 kB/s | 2.8 MB 00:00
(4/8): perl-file-find-1.37-477.amzn2023.0.2.noarch.rpm 1.7 MB/s | 100 kB 00:00
(5/8): git-core-2.47.1-1.amzn2023.0.3.x86_64.rpm 37 kB/s | 4.5 MB 00:00
(6/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm 973 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 876 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 : git-core-2.47.1-1.amzn2023.0.3.x86_64 1/1
Installing : git-core-doc-2.47.1-1.amzn2023.0.3.noarch 1/8
Installing : perl-error-0.17029-5.amzn2023.0.2.noarch 2/8
Installing : perl-file-find-1.37-477.amzn2023.0.2.noarch 3/8
Installing : git-core-2.47.1-1.amzn2023.0.3.x86_64 4/8
Installing : perl-Git-2.47.1-1.amzn2023.0.3.noarch 5/8
Installing : perl-TermReadkey-2.38-9.amzn2023.0.2.x86_64 6/8
Installing : perl-lib-0.65-477.amzn2023.0.7.x86_64 7/8
Running scriptlet: git-2.47.1-1.amzn2023.0.3.x86_64 8/8
Verifying : git-2.47.1-1.amzn2023.0.3.x86_64 1/1
Verifying : git-core-2.47.1-1.amzn2023.0.3.noarch 1/8
Verifying : perl-error-0.17029-5.amzn2023.0.2.noarch 2/8
Verifying : perl-file-find-1.37-477.amzn2023.0.2.noarch 3/8
Verifying : perl-Git-2.47.1-1.amzn2023.0.3.noarch 4/8
Verifying : perl-TermReadkey-2.38-9.amzn2023.0.2.x86_64 5/8
Verifying : perl-lib-0.65-477.amzn2023.0.7.x86_64 6/8
Installed: git-2.47.1-1.amzn2023.0.3.x86_64
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 ~]$ 
```

```
ec2-user@ip-172-31-82-26:~$ 
Install 2 Packages
Total download size: 1.9 M
Installed size: 1 M
Downloading packages:
(1/2): libxcrypt-compat-4.4.33-7.amzn2023.x86_64 2.4 MB/s | 92 kB 00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.11.noarch 23 kB/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 : libxcrypt-compat-4.4.33-7.amzn2023.x86_64 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 1/1
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)
Collecting werkzeug<3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
Collecting click<8.1.3
  Downloading Click-8.1.3-py3-none-any.whl (98 kB)
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-manylinux2014_x86_64.whl (20 kB)
Collecting jinja2<3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
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-3.7.0-py3-none-any.whl (27 kB)
Collecting 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 flask-3.1.1 importlib-metadata-3.7.0 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 ~]$ 
```

- 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>

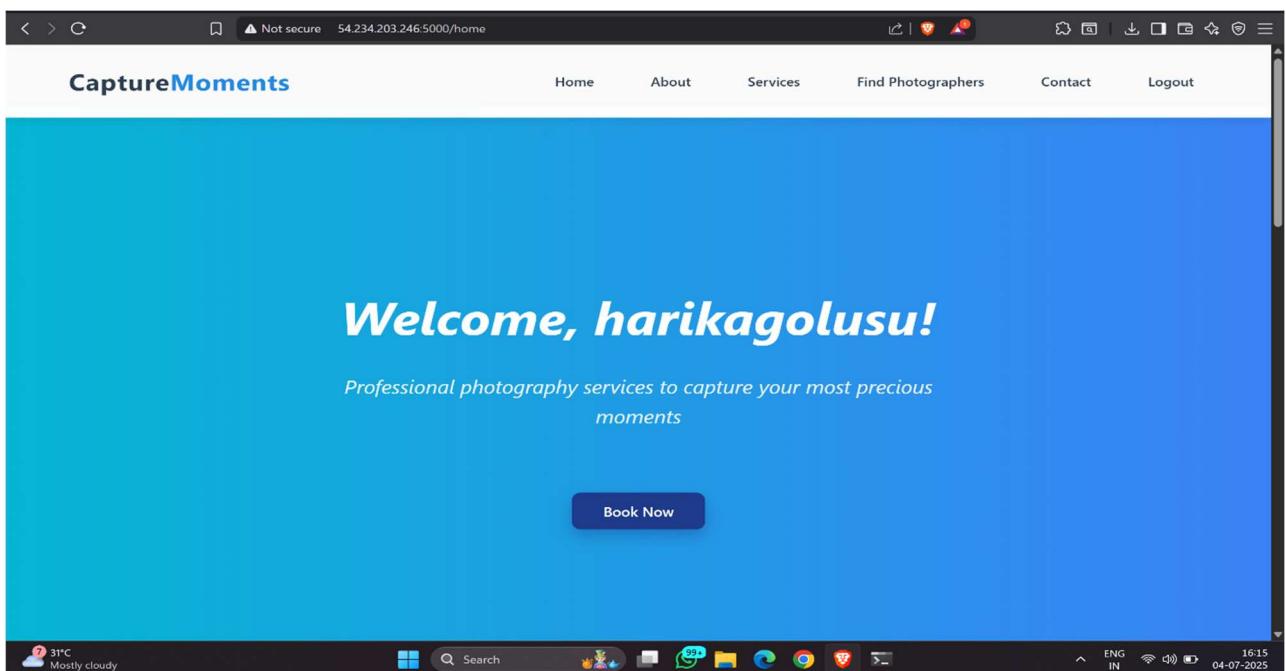
- Run the Flask app on the EC2 instance

```
ec2-user@ip-172-31-31-214:~/CaptureMoments-AWS$ bug https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn("Python deprecation warning")
INFO:boto3.core.credentials:Found credentials from IAM Role: EC2DynamoDBAccessRole
[?] Flask server starting on http://localhost:5000
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 332-625-655
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python3 app.py
/home/ec2-user/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates,
bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn("Python deprecation warning")
INFO:boto3.core.credentials:Found credentials from IAM Role: EC2DynamoDBAccessRole
[?] Flask server starting on http://localhost:5000
= serving Flask app 'app'
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python3 app.py
INFO:werkzeug: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
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
/home/ec2-user/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates,
bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn("Python deprecation warning")
INFO:boto3.core.credentials:Found credentials from IAM Role: EC2DynamoDBAccessRole
[?] Flask server starting on http://localhost:5000
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 332-625-655
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ git pull origin main
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Up-to-date! [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python app.py
From https://github.com/Harikagolusu/CaptureMoments-AWS
 * branch            main      -> FETCH_HEAD
  libbccf_fcc2bf4f_main      -> origin/main
Updated 10bccf..f0c2bf4
Fast-forward
 app.py | 4 +---
 1 file changed, 2 insertions(+), 2 deletions(-)
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python app.py
File "app.py", line 55
    logger.error("Login error: {}")
    ^
SyntaxError: invalid syntax
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python3 app.py
/home/ec2-user/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates,
bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn("Python deprecation warning")
INFO:boto3.core.credentials:Found credentials from IAM Role: EC2DynamoDBAccessRole
[?] Flask server starting on http://localhost:5000
= serving Flask app 'app'
[?] [ec2-user@ip-172-31-31-214 CaptureMoments-AWS]$ python3 app.py
INFO:werkzeug: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
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
/home/ec2-user/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates,
bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn("Python deprecation warning")
INFO:boto3.core.credentials:Found credentials from IAM Role: EC2DynamoDBAccessRole
[?] Flask server starting on http://localhost:5000
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 332-625-655
```

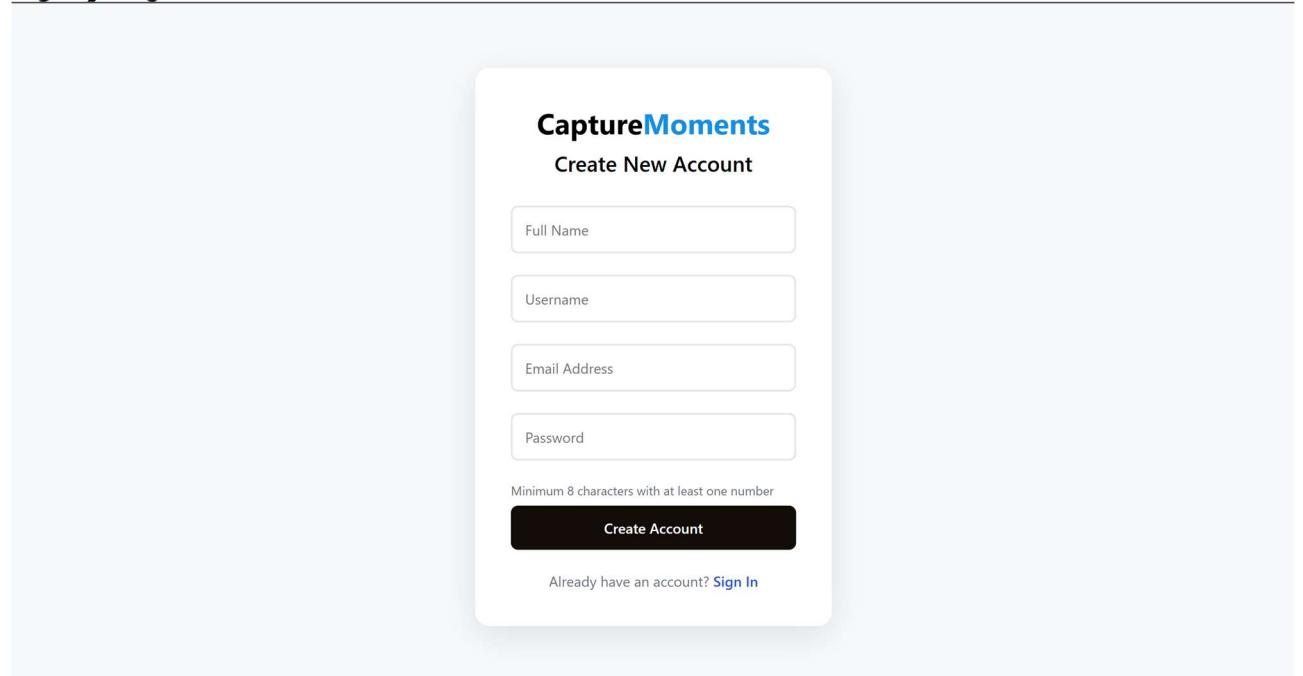
Milestone 7. Testing and Deployment

Conduct functional testing to verify user signup, login, home, booking, photographers, Contact us , about us pages.

Home Page :

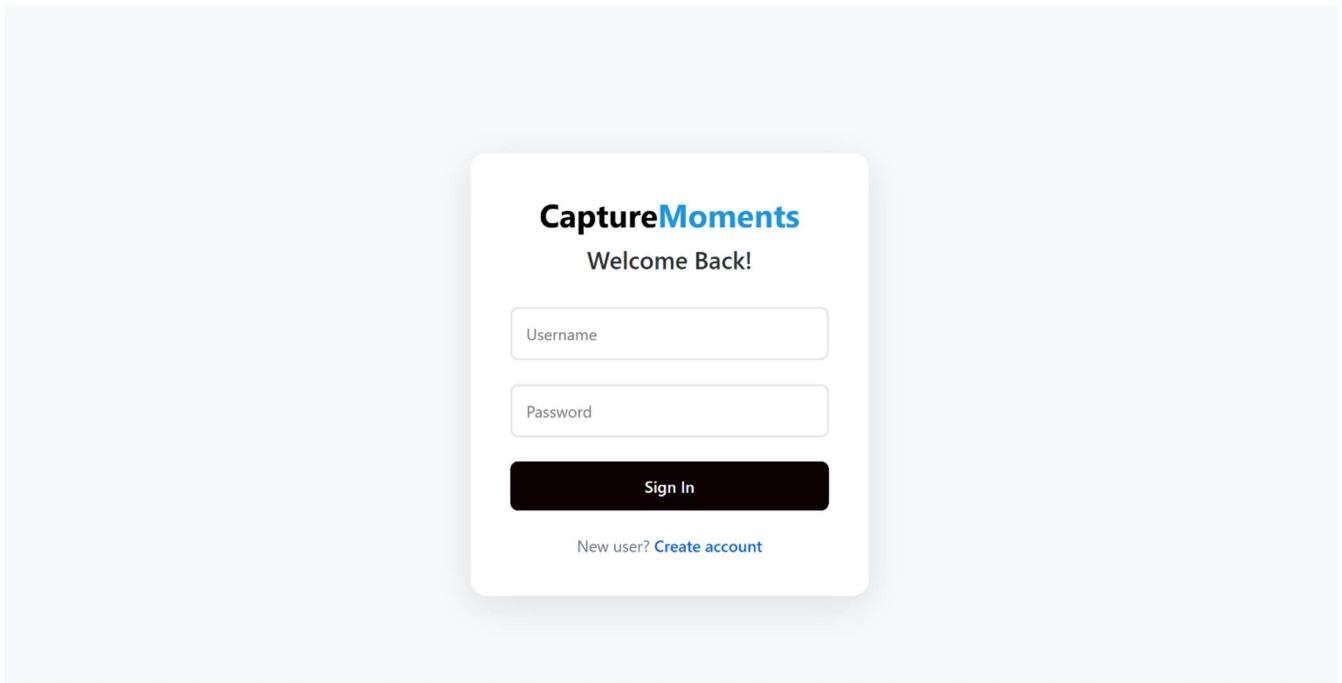


Signup Page :

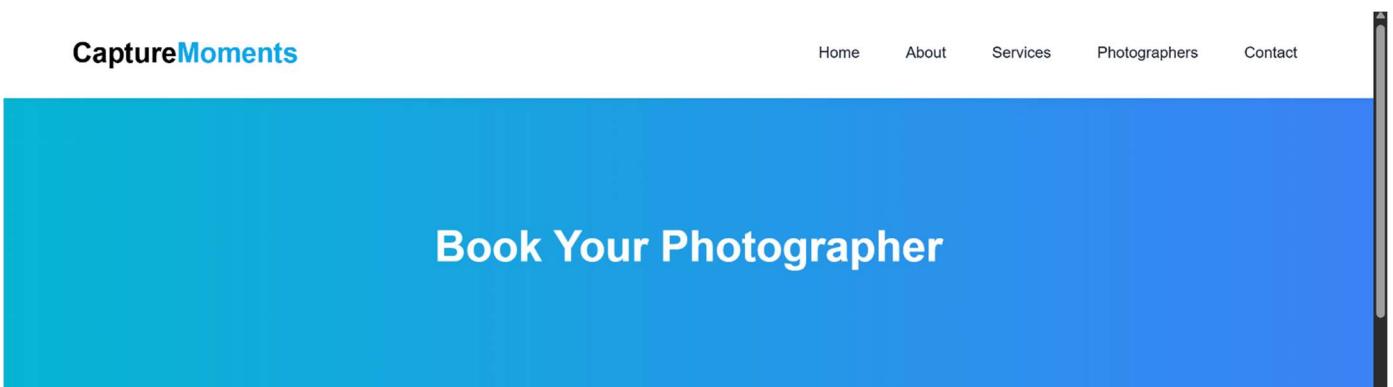


The screenshot shows a 'Create New Account' form for 'CaptureMoments'. The form includes fields for Full Name, Username, Email Address, and Password. Below the password field is a note: 'Minimum 8 characters with at least one number'. A 'Create Account' button is at the bottom, and a 'Sign In' link is below it. The background is white with a light gray shadow around the form.

Login Page



Booking Page



The booking page has a large blue header section with the text "Book Your Photographer" in white. Below this, a white form is titled "Booking Details". It includes a dropdown menu for "Event Type" with the placeholder "Select Event Type". Under "Start Date", there is a date input field showing "dd-mm-yyyy" with a calendar icon. Under "End Date", there is another date input field showing "dd-mm-yyyy" with a calendar icon. The "Booking Details" title has a blue underline.

Confirmation Page :

CaptureMoments



Booking Confirmed!

Your photography session has been successfully booked.

We've sent a confirmation email to your registered email address. Our team will contact you shortly to discuss further details.

[Back to Home](#)

Photographers page:

CaptureMoments

- [Home](#)
- [About](#)
- [Services](#)
- [Find Photographers](#)
- [Contact](#)

Featured Photographers


Featured

James

Wedding Photography

★ 4.9/5 8 years experience

Specializing in capturing the most precious moments of your special day with a romantic and elegant style. Sarah has photographed over



Aditya

Portrait Photography

★ 4.8/5 6 years experience

Expert in creating stunning individual and family portraits that capture personality and emotion. Michael's studio work is renowned for


Featured

Sushanth

Commercial Photography

★ 4.7/5 7 years experience

Professional product and architectural photography to elevate your brand's visual identity. Emma has worked with Fortune 500

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.