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.

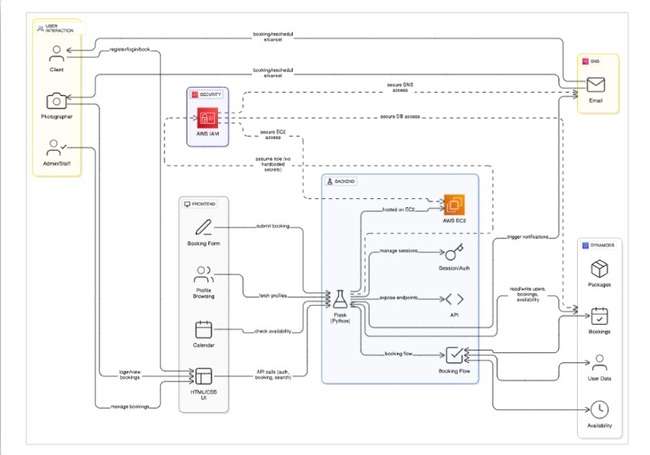
**Scenario 2: Centralized Booking Management**

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.

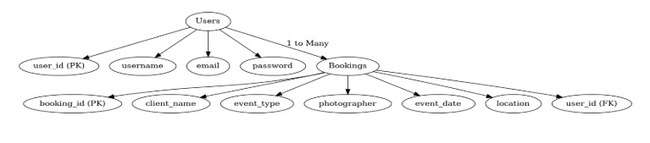
**Scenario 3: Easy Access to Photography Services**

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**: [AWS Account Setup](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)
2. **Understanding IAM**: [IAM Overview](https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR)
3. **Amazon EC2 Basics**: [EC2 Tutorial](https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz)
4. **DynamoDB Basics**: [DynamoDB Introduction](https://docs.aws.amazon.com/dynamodb)
5. **Git Version Control**: [Git Documentation](https://git-scm.com/doc)

## Project WorkFlow:

### AWS Account Setup and Login

**Activity 1.1:** Set up an AWS account if not already done**.** **Activity 1.2:** Log in to the AWS Management Console

### DynamoDB Database Creation and Setup

**Activity 2.1**: Create a DynamoDB Table.

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

### Backend Development and Application Setup

**Activity 3.1**:Develop the Backend Using Flask.

**Activity 3.2**: Integrate AWS Services Using boto3.

### IAM Role Setup

**Activity 4.1**: Create IAM Role

**Activity 4.2**: Attach Policies

### EC2 Instance Setup

**Activity 5.1**: Launch an EC2 instance to host the Flask application.

**Activity 5.2**: Configure security groups for HTTP, and SSH access.

1. **Deployment on EC2 Activity 6.1**:Upload Flask Files **Activity 6.2**: Run the Flask App

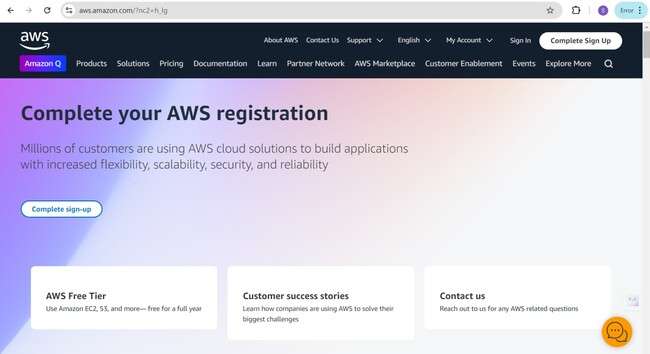
### 7.Testing and Deployment

**Activity 7.1**: Conduct functional testing to verify user registration, login, book requests, and notifications.

# Milestone 1: AWS Account Setup and Login

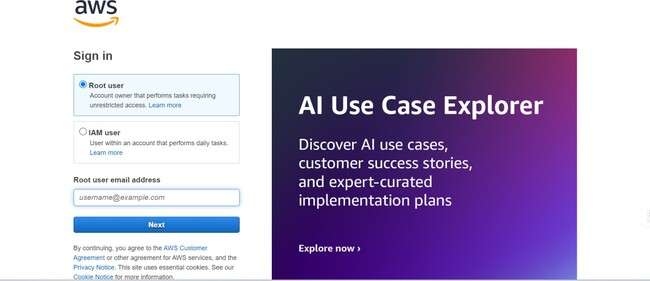
### Activity 1.1: Set up an AWS account if not already done.

* + - Sign up for an AWS account and configure billing settings.



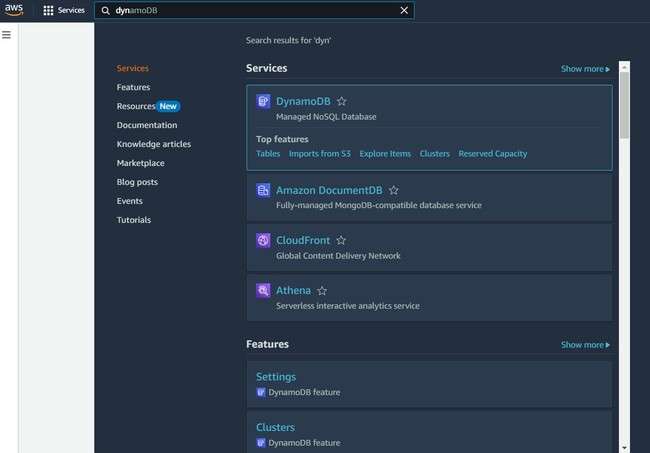
### Activity 1.2: Log in to the AWS Management Console

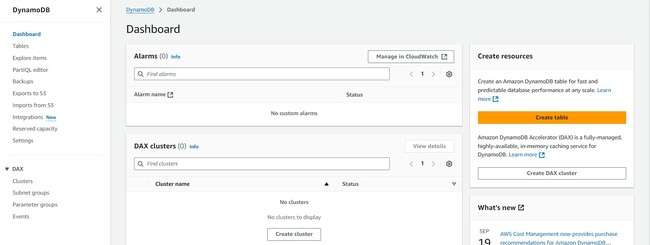
* + - After setting up your account, log in to the [AWS Management Console](https://aws.amazon.com/console/).

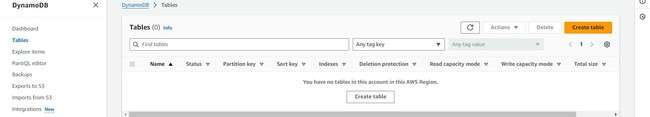


# Milestone 2: DynamoDB Database Creation and Setup

### Activity 2.1:Navigate to the DynamoDB

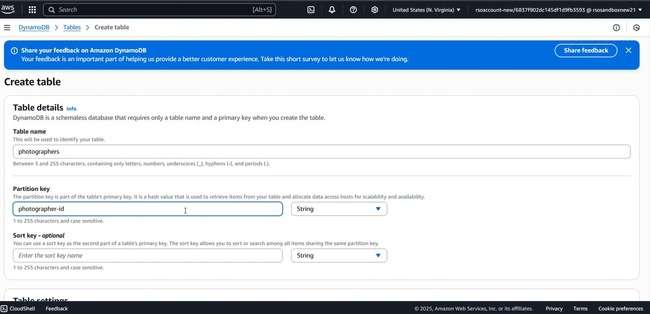
* + - In the AWS Console, navigate to DynamoDB and click on create tables.
    - 

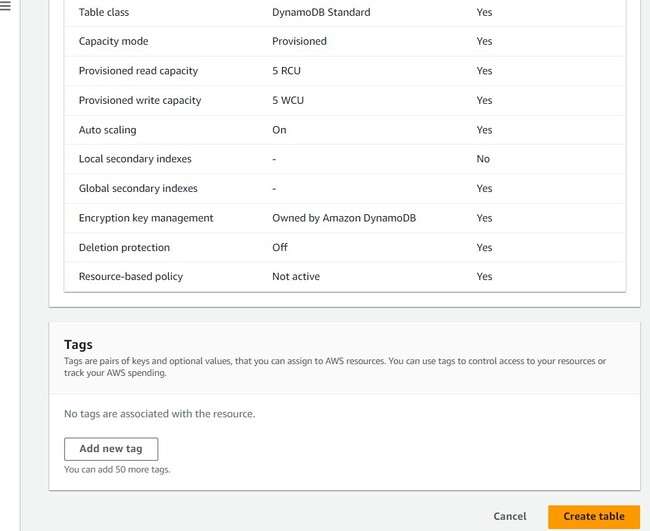


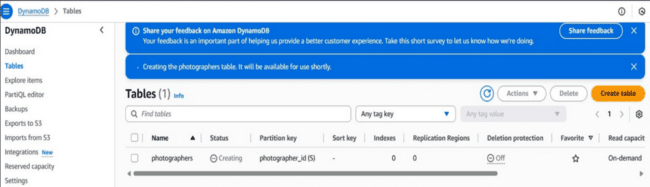


### Activity 2.2:Create a DynamoDB table for storing registration details and book requests.

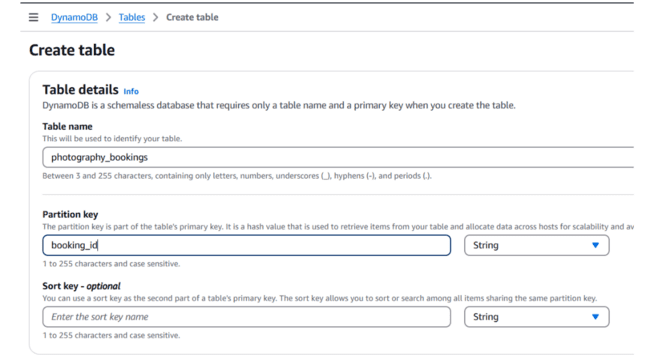
* + - Create photographers table with partition key “photographer\_id” with type String and click on create tables.

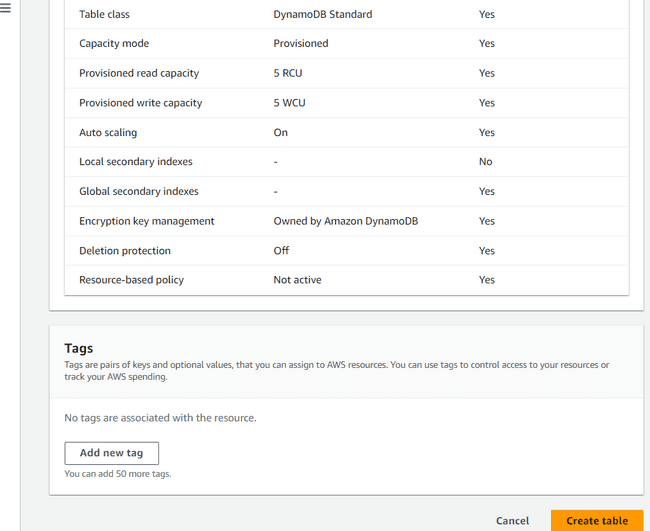


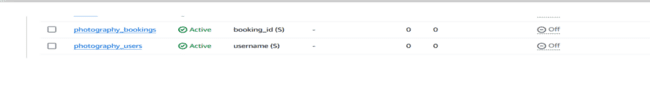




* + - Follow the same steps to create a requests table with photographer\_id as the primary key for book requests data.



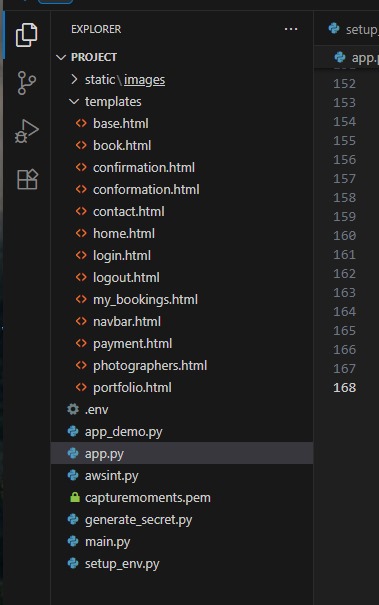




# Milestone 3:Backend Development and Application Setup

### Activity 3.1: Develop the backend using Flask

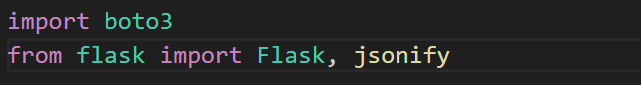
* + File Explorer Structure



**Description**: The templates/ folder contains HTML pages like home, login, register, book, and event-specific pages (baby, fashion, wedding, etc.). Python files like app.py and app\_demo.py handle the backend logic, while awsint.py likely manages AWS integration. The static/ folder stores assets like CSS and images. A database file and README are also included for data storage and project guidance.

## Description of the code:

### Flask App Initialization

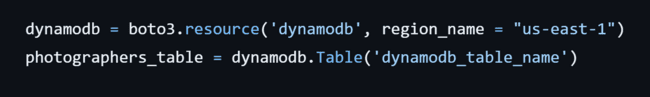
****

**Description:** This code imports boto3 for interacting with AWS services. It also imports Flask to create a web application and jsonify to return JSON responses from routes**.**



**Description:** initialize the Flask application instance using Flask( \_name\_\_) to start building the web app.

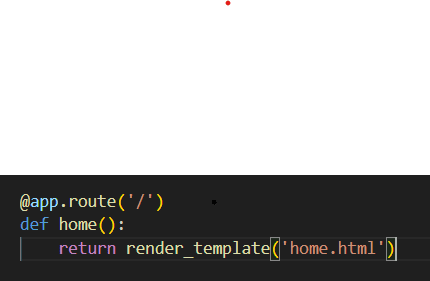
### Dynamodb Setup:

****

**Description:** initialize the DynamoDB resource for the us-east-1 region and set up access to the photographers table for storing details.

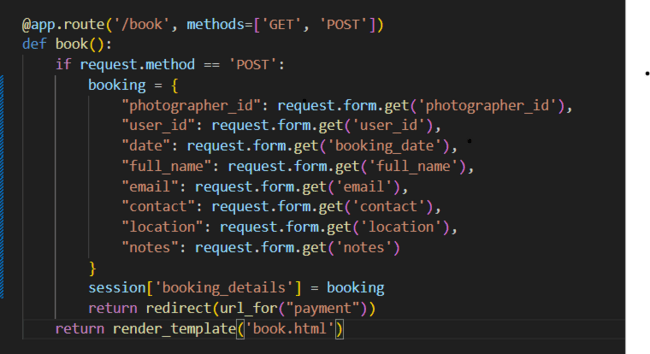
## Routes for Web Pages

### Home Route:

****

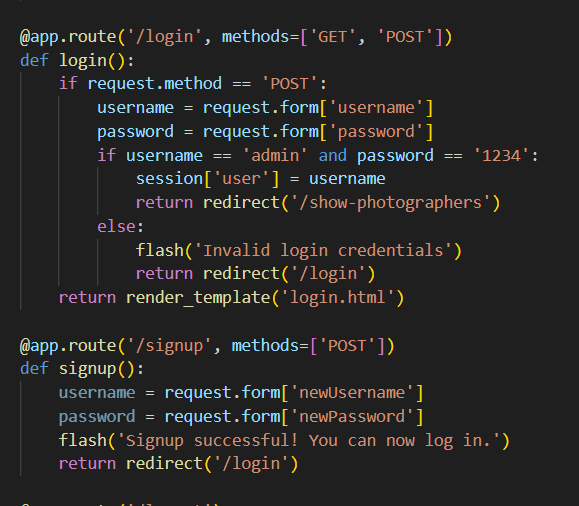
**Description:** define the home route / to automatically redirect users to the register page when they access the base URL.

### Book Route:

****

**Description:** This Flask route handles user registration with both GET and POST methods. It collects username, email, and password from a form and performs basic validation. If fields are missing or the username/email already exists, it shows an error. On success, it stores the user data with a hashed password and redirects to the login page. The register.html template is rendered for the GET request.

### login Route (GET/POST):



**Description**: This Flask code handles user login and logout functionality. The /login route checks if the entered username or email and password match a registered user. If valid, it stores session data and redirects to the home page; otherwise, it shows an error. The /logout route clears the session and redirects back to the login page with a success message.

Payment Route:

****

**Description:** This code defines the /home and /book routes in a Flask app. The home() route ensures only logged-in users can access the home page. The book() route allows logged-in users to submit a booking via POST, storing details like date, price, and photographer ID. Bookings are saved in a list and a success message is shown. The book.html template is rendered for GET reques

**Deployment Code:**

****

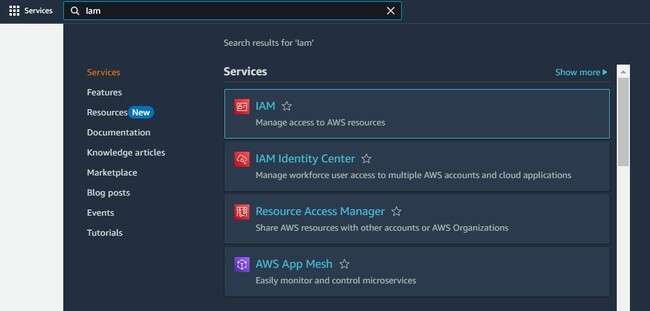
**Description:** start the Flask server to listen on all network interfaces (0.0.0.0) at port 80

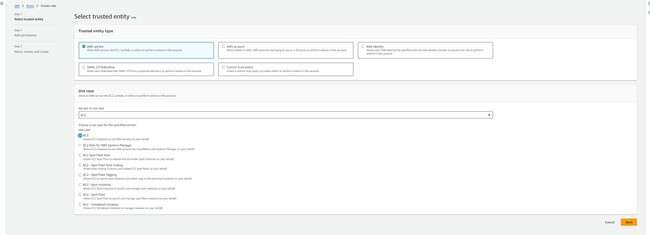
with debug mode enabled for development and testing.

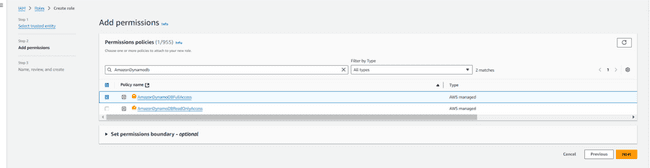
# Milestone 5: IAM Role Setup

### Activity 5.1:Create IAM Role.

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



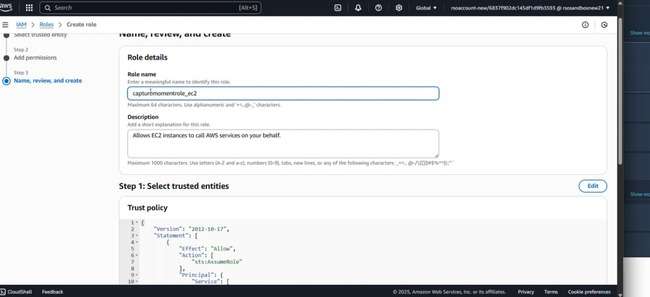




### Activity 5.2: Attach Policies.

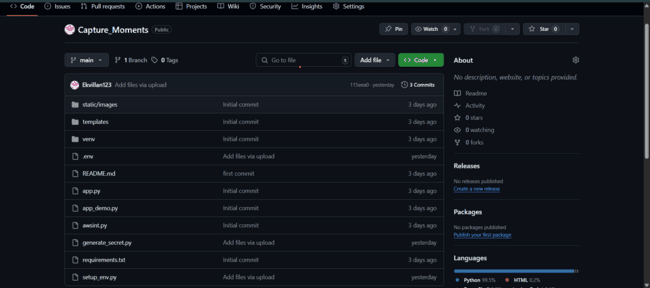
Attach the following policies to the role:

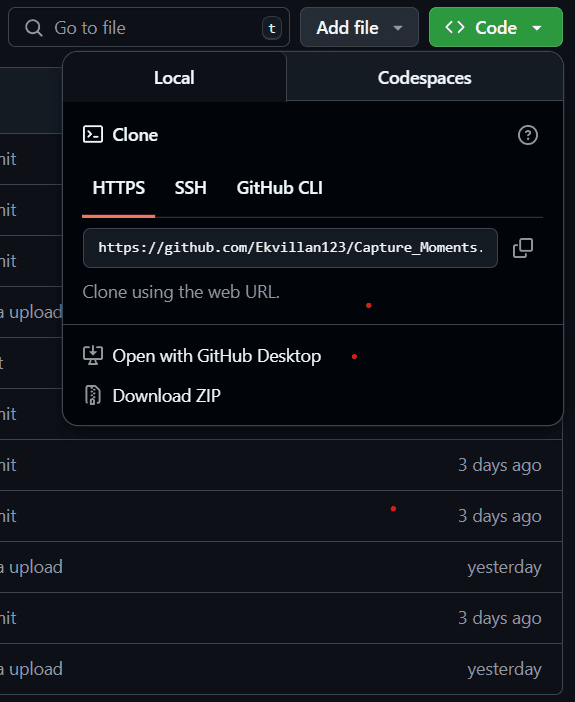
* AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.



# Milestone 6: EC2 Instance Setup

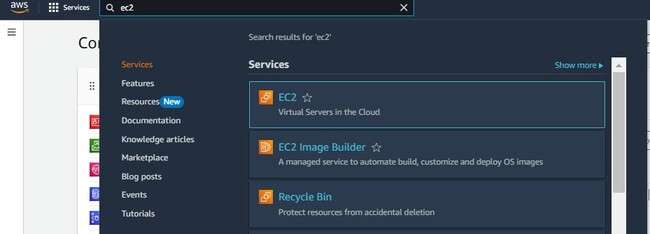
* **Note: Load your Flask app and Html files into GitHub repository.**



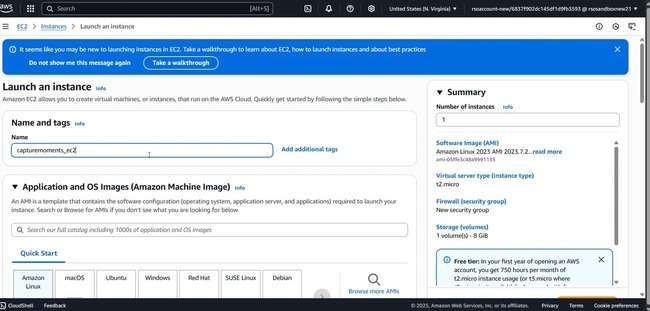
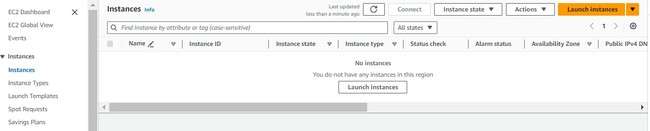


### Activity 6.1: Launch an EC2 instance to host the Flask application.

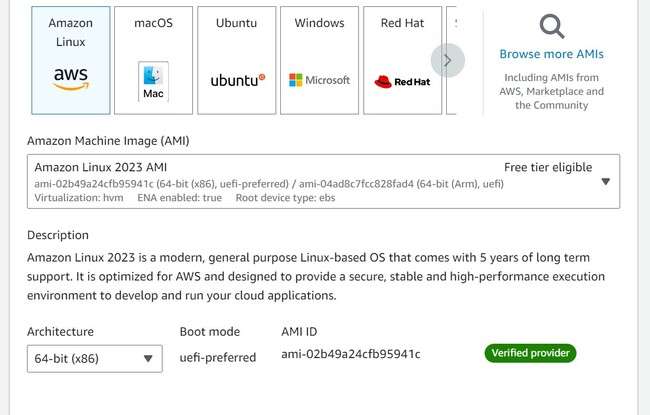
* + **Launch EC2 Instance**
    - In the AWS Console, navigate to EC2 and launch a new instance.



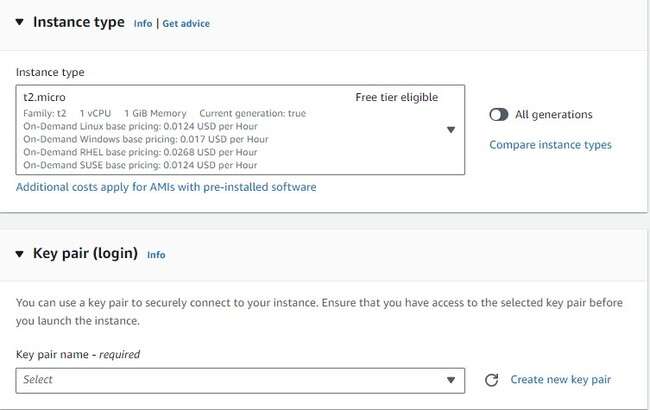
* Click on Launch instance to launch EC2 instance

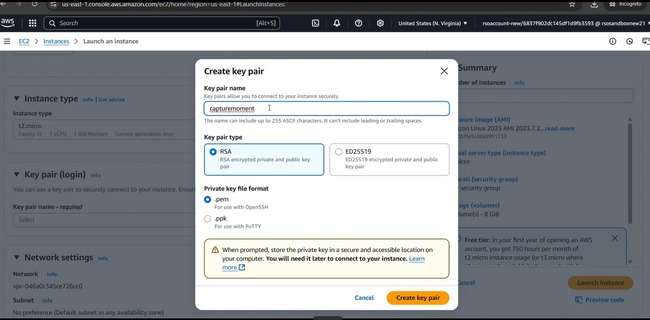


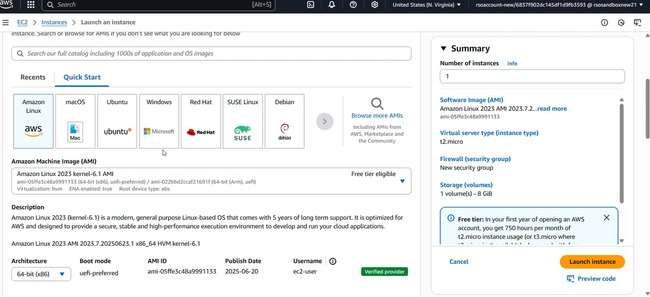
* + Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



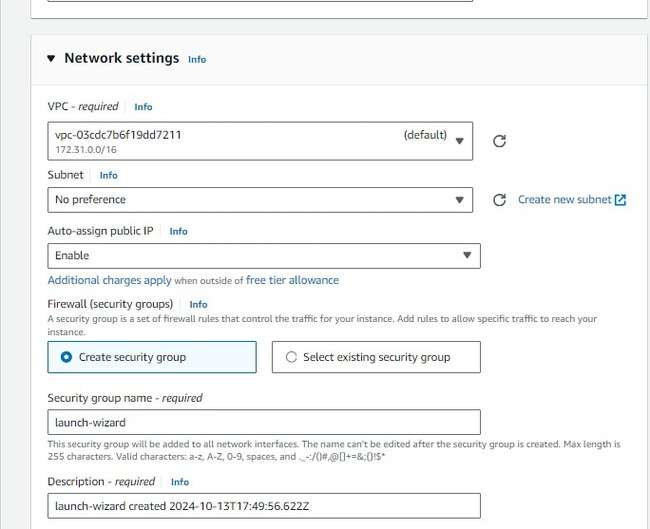
* Create and download the key pair for Server access.

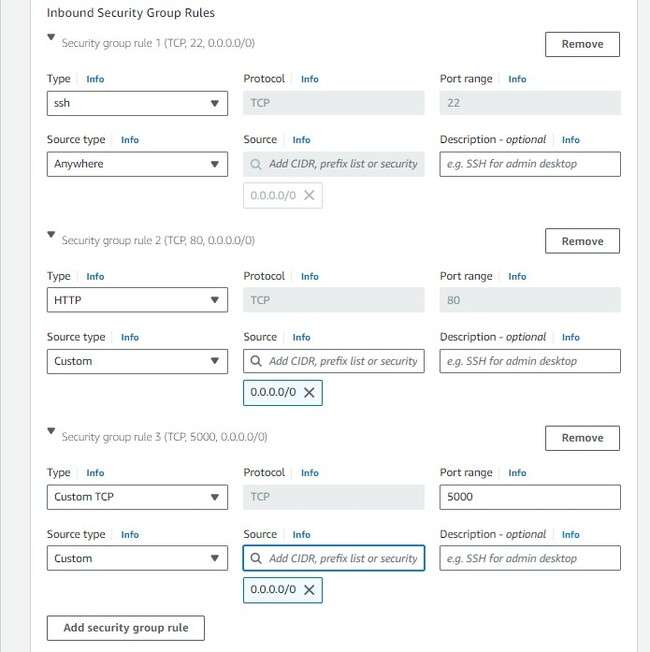


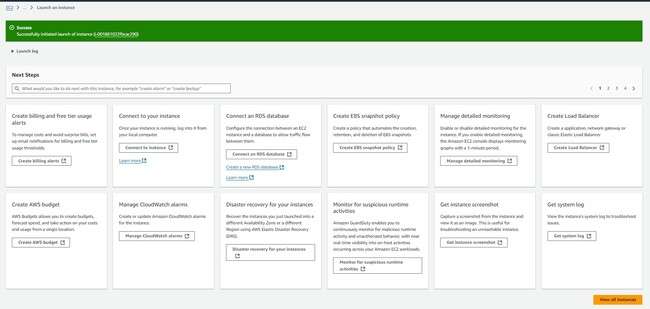




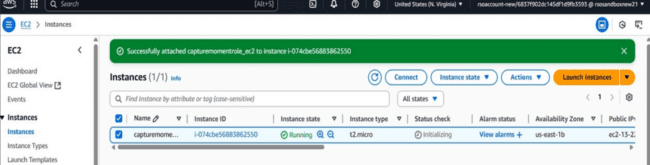
### Activity 6.2:Configure security groups for HTTP, and SSH access.

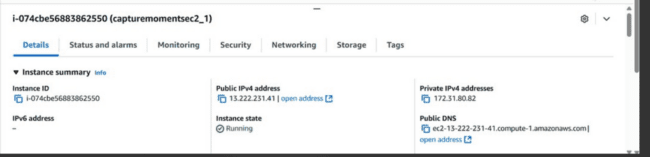
****

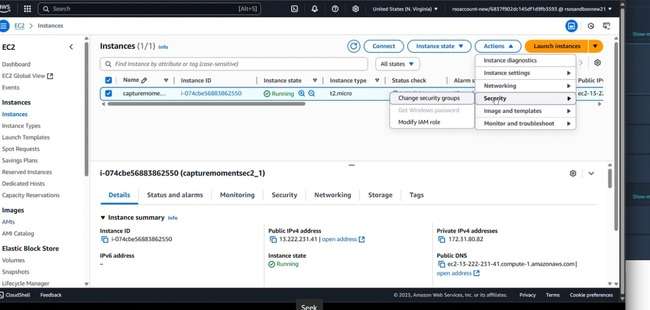
****

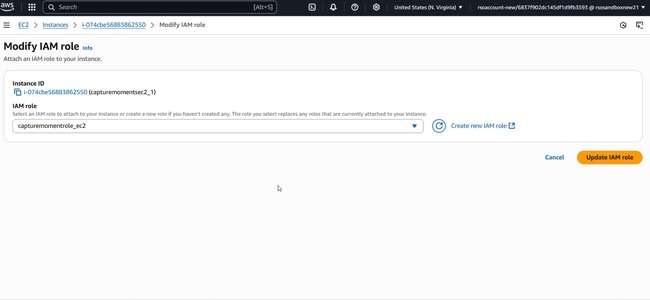


* To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the 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 **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

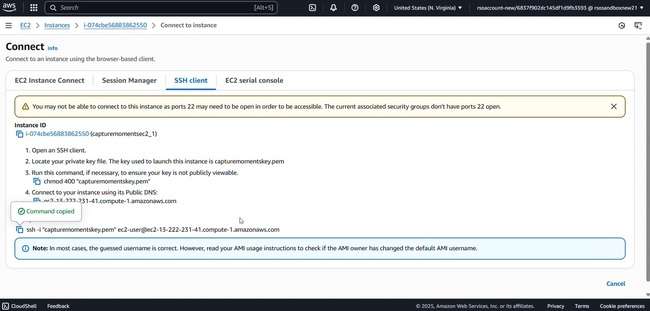








* Now connect the EC2 with the files





**Milestone 7: Deployment on EC2**

### Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git: On Amazon Linux 2:

sudo yum update -y

sudo yum install python3 git sudo pip3 install flask boto3

Verify Installations:

flask --version git --version

### Activity 7.2:Clone Your Flask Project from GitHub

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

Run: https://github.com/Sivadattakonni/capturemoments.git

Note: change your-github-username and your-repository-name with your credentials here: ‘https://github.com/Sivadattakonni/capturemoments.git’

* This will download your project to the EC2 instance.

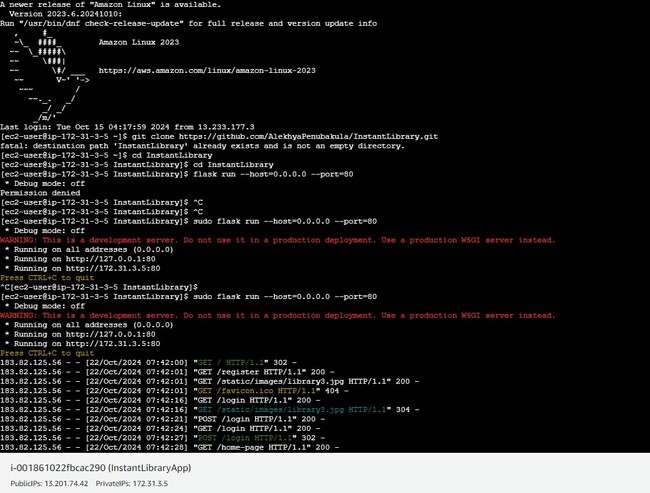
### To navigate to the project directory, run the following command:

C:\Users\sshiv\OneDrive\Desktop\project

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

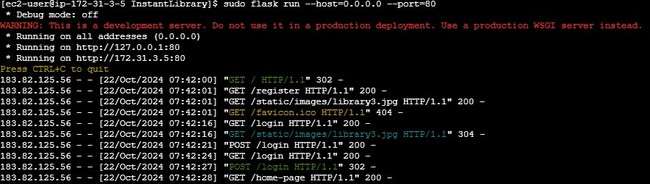
**Run the Flask Application**

sudo flask run --host=0.0.0.0 --port=80



**Verify the Flask app is running**: [http://your-ec2-public-ip](http://your-ec2-public-ip/)

* Run the Flask app on the EC2 instance



### Access the website through:

**PublicIP:** http://127.0.0.1:5000

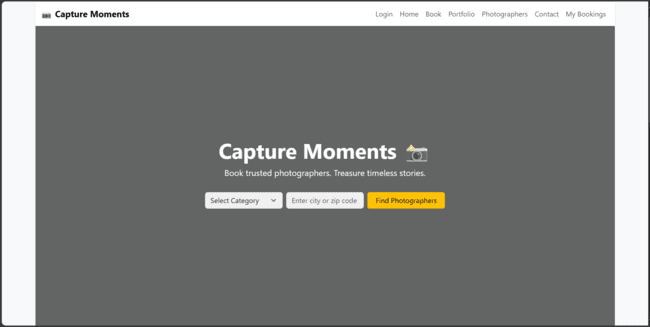
## Milestone 8: Testing and Deployment

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

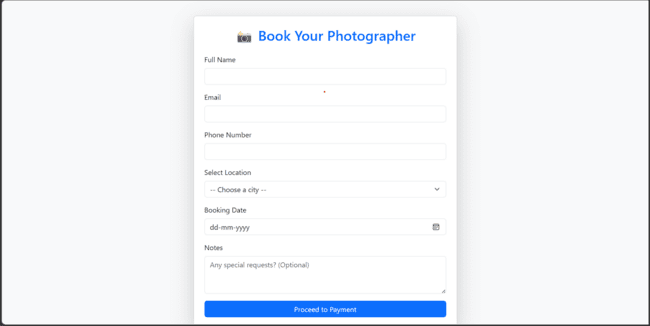
**Login Page:**

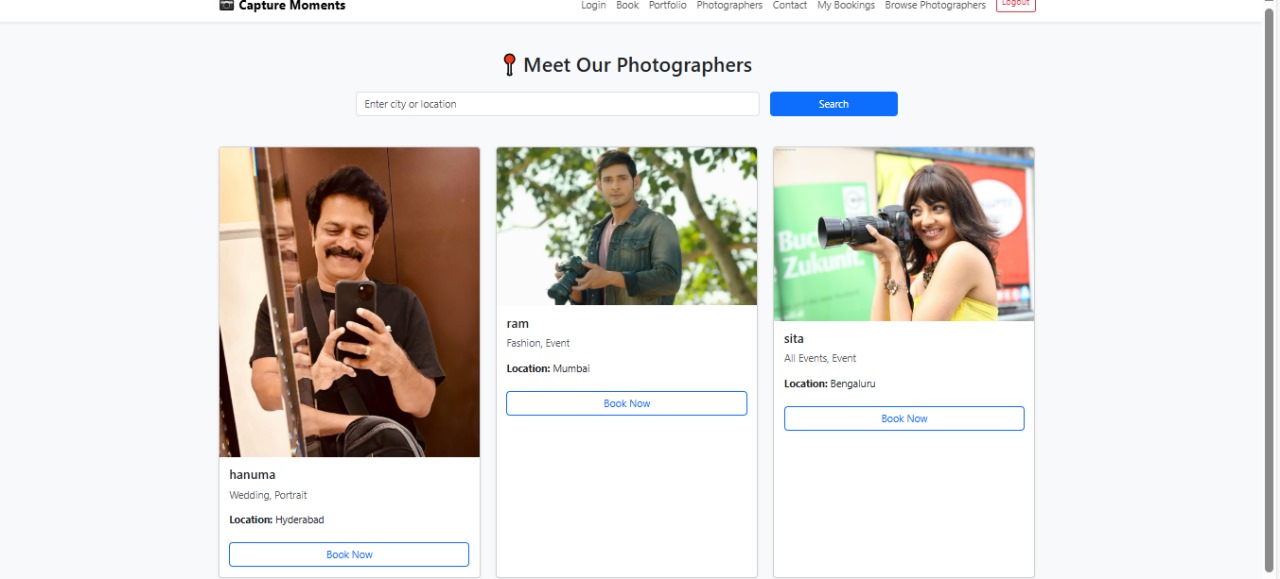
****

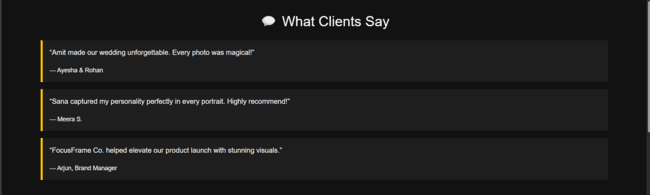
**Home page:**

****

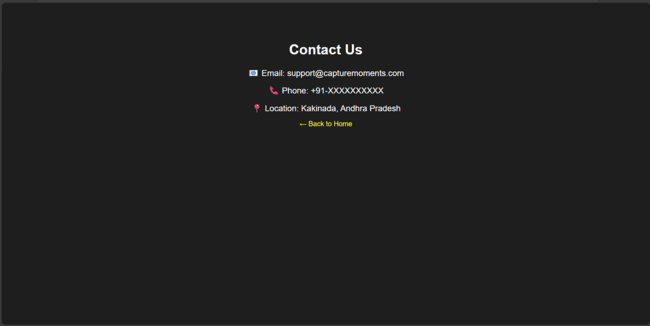
**Booking page:**

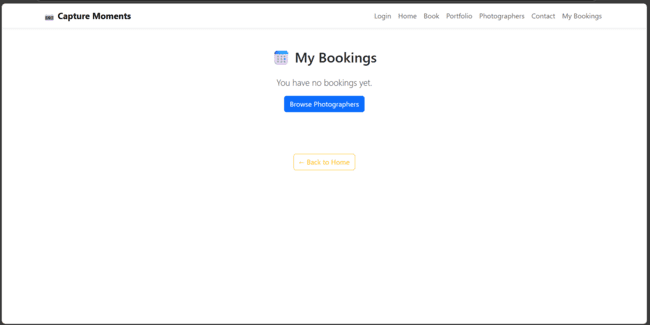
****

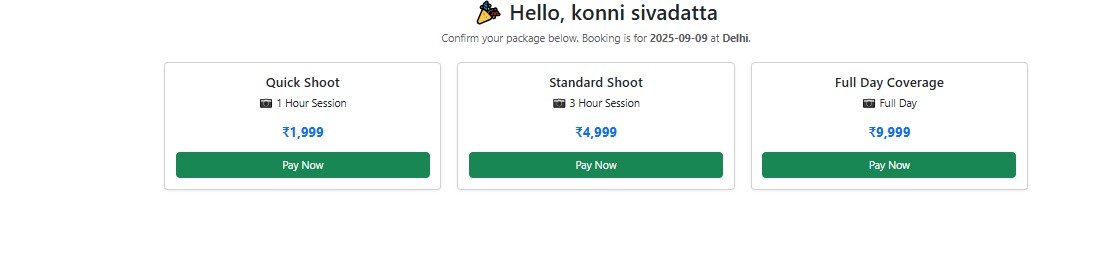
**Photographers Page:**  ****

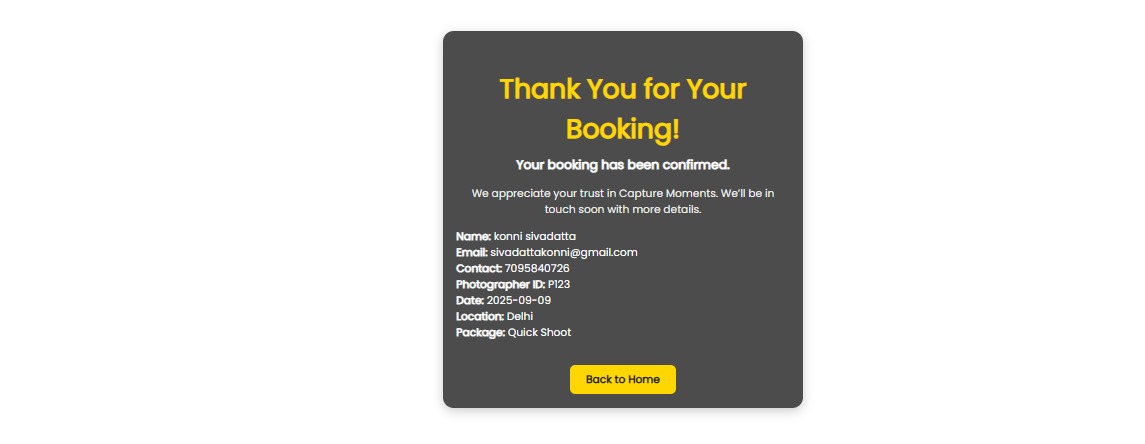
**Portfolio Page:**  ****

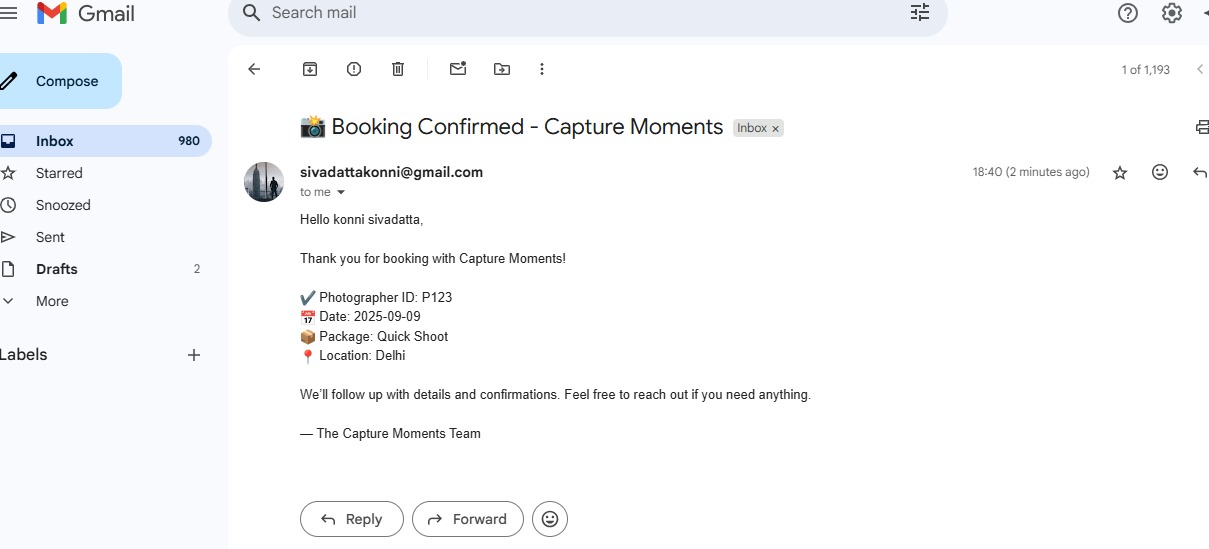
**Contact Page:**

****

**My\_Bookings page:**  ****

**Payment Page:**  ****

**Confirmation Page:**  ****

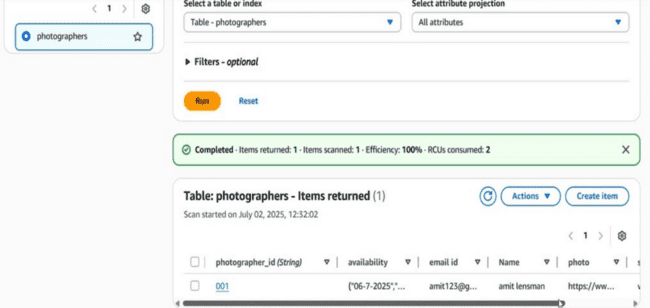
****

**Exit:**

****

**Dynamodb Database updations :**

1. **Users table :**

****

# 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 quickaccess to media content. Designed to address

the common challenges of disorganized photo sharing and limited storage, the platform in

offers a centralized and user-friendly interface for uploading, organizing, and accessing

event photos. Flask powers the backend operations, enabling smooth workflows for was

user authentication, album creation, and secure photo management. Throughout the in

testing phase, all core functionalities—from user registration to photo uploads and can

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 setting