**Momade pickles & snacks: taste the best**

## Project Description:

The MOMade Pickles & Snacks project is a mini e-commerce web application developed using **Flask** (Python-based web framework), aimed at showcasing and selling home-made pickles and snacks. The application provides users with an interactive shopping experience, allowing them to browse a product catalog, add items to a cart, and place orders using **Cash on Delivery (COD)**.

The project is fully **AWS-ready**, integrated with key cloud services to ensure scalability, reliability, and real-world deployment capabilities.

**Scenario 1: Smooth Shopping Experience During Festival Rush**

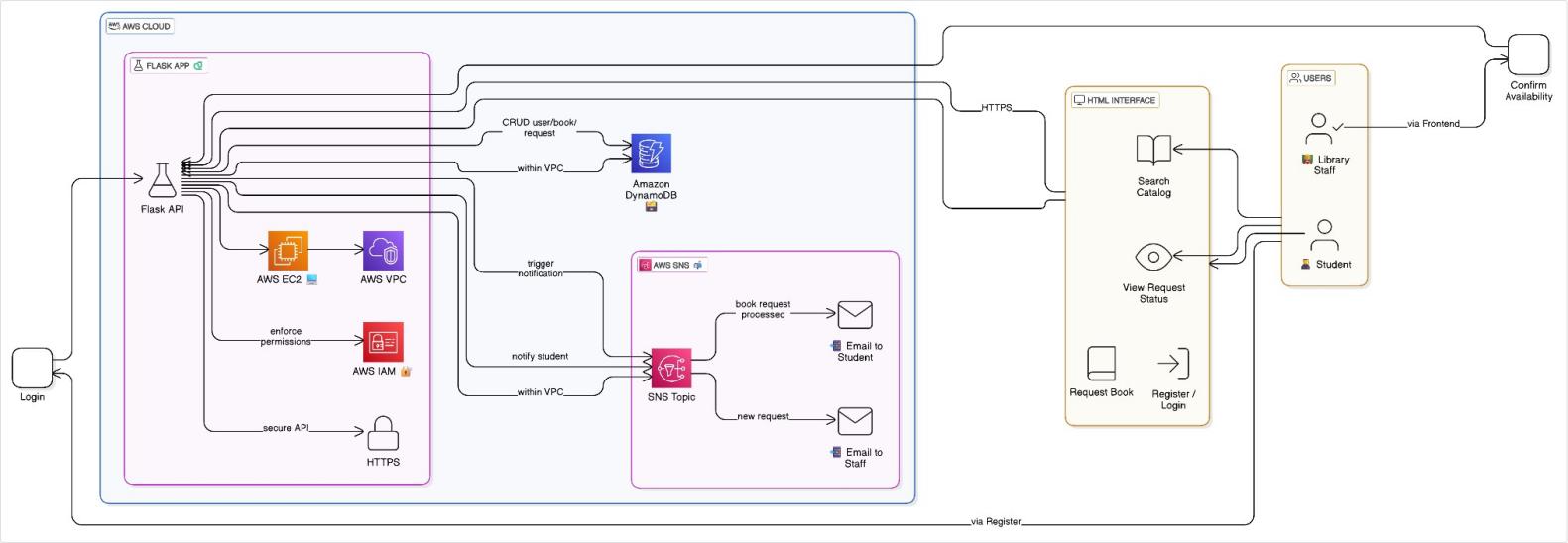
During festive seasons like Diwali or Sankranti, MOMade Pickles sees a spike in orders. Thanks to **AWS EC2**, the website can handle hundreds of users browsing and adding items to their cart at the same time.For example, a customer logs in to the website and selects Mango Pickle (500g) and Banana Chips (1kg). Flask processes this request, calculates the price based on quantity and weight, and adds it to the user's cart. Even with heavy traffic, the website continues to run smoothly without downtime, offering a seamless shopping experience.

### Scenario 2:Order Confirmation with Email Alerts

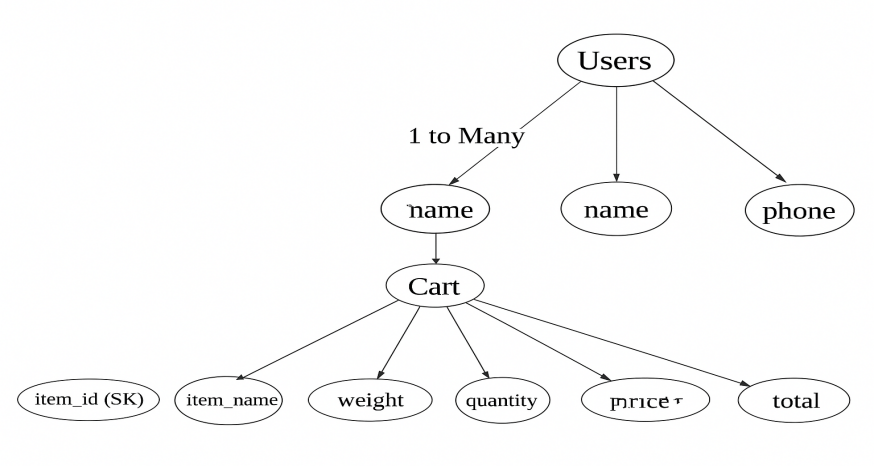
Once a customer finalizes the order, the application provides instant order confirmation through email. When a user submits their delivery address and clicks "Order Now", Flask handles the backend operations by fetching all cart items, calculating the total, clearing the cart, and triggering an automated email. This email confirmation, sent using AWS Simple Email Service (SES) or Gmail SMTP configured via environment variables, contains the delivery details and payment information. This seamless integration ensures customers receive immediate communication about their purchase, enhancing trust and user experience.

**Scenario 3: Customer Feedback and Review Submission**  
To build community trust and product credibility, MOMade Pickles allows registered users to submit reviews after trying the products. After enjoying the Lemon Pickle, for instance, a user can log in and submit a review like, “Tangy and delicious! Reminds me of my grandma’s recipe.” Flask validates the user's login session, processes the review form, and stores the review in the AWS DynamoDB Reviews table along with a timestamp. These reviews are later displayed on the reviews page, along with a few dummy testimonials, helping new users gain confidence in the product quality and service reliability.

**AWS ARCHITECTURE:**



**Entity Relationship:**



## 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. **SNS Overview**: [SNS Documentation](https://docs.aws.amazon.com/sns)
6. **Git Version Control**: [Git Documentation](https://git-scm.com/doc)

## Project WorkFlow:

**1. AWS Account Setup and Login**

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

**Activity 1.2:** Log in to the AWS Management Console.

### 2.DynamoDB Database Creation and Setup

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

**Activity 2.2**:Configure attributes:

* Users: email (PK), name, phone, password
* Cart: email (PK), item\_id (SK), item\_name, weight, quantity, price, total, timestamp
* Reviews: email (PK), timestamp (SK), name, message

### SNS Notification Setup

· **Activity 3.1:** Create an SNS topic for order confirmation.

· **Activity 3.2:** Subscribe user email IDs for notifications upon placing orders.

### 4.Backend Development and Application Setup

· **Activity 4.1:** Develop the backend using **Flask** to manage routing, session, and logic.

· **Activity 4.2:** Integrate AWS services (DynamoDB, SNS, SES) using **boto3** library.

### 5.IAM Role Setup:

· **Activity 5.1:** Create an IAM Role (EC2\_DynamoDB\_SES\_Role) to allow EC2 access to AWS services.

· **Activity 5.2:** Attach policies like AmazonDynamoDBFullAccess, AmazonSNSFullAccess, AmazonSESFullAccess.

### 6.EC2 Instance Setup

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

· **Activity 6.2:** Configure Security Groups to allow inbound traffic on ports:

* **80** (HTTP)
* **22** (SSH)
* **5000** (Flask default, for testing)

**7.Deploymet on EC2**

· **Activity 7.1:** Upload Flask project files to EC2 using Git.

· **Activity 7.2:** Run the Flask app using python app.py

### 8.Testing and Deployment

**Activity 8.1:** Conduct full functional testing:

User registration and login

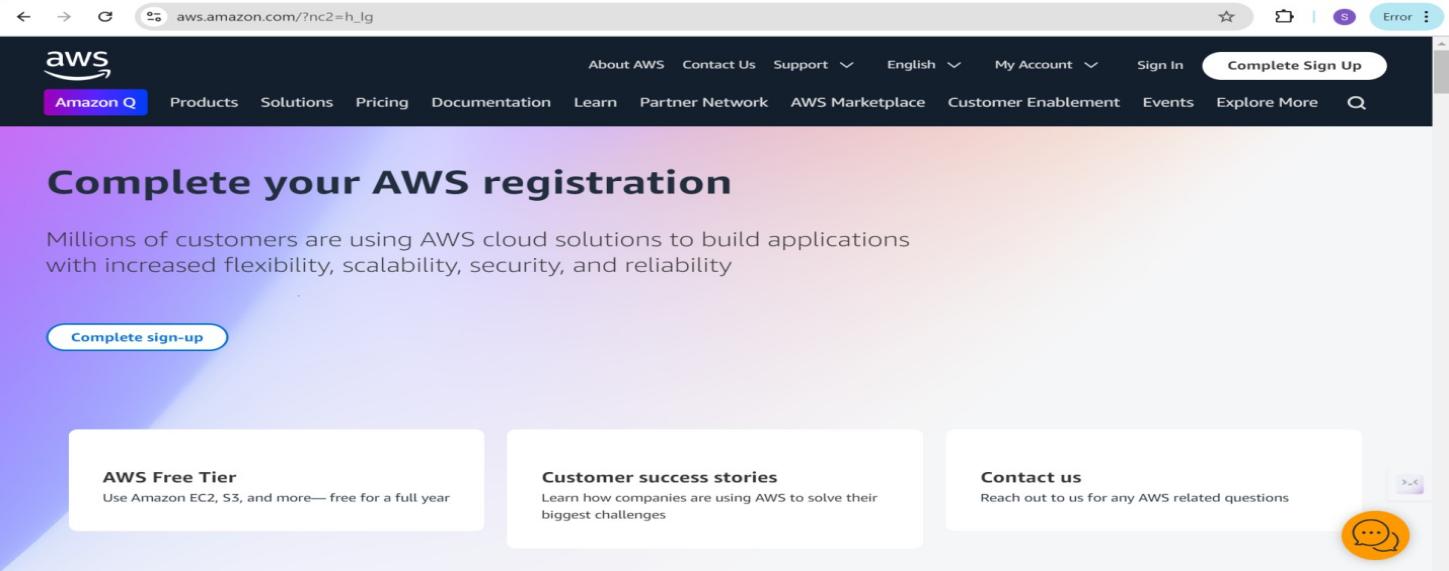
Add to cart and review

Place orders and receive SNS/SES notification.

# Milestone 1: AWS Account Setup and Login

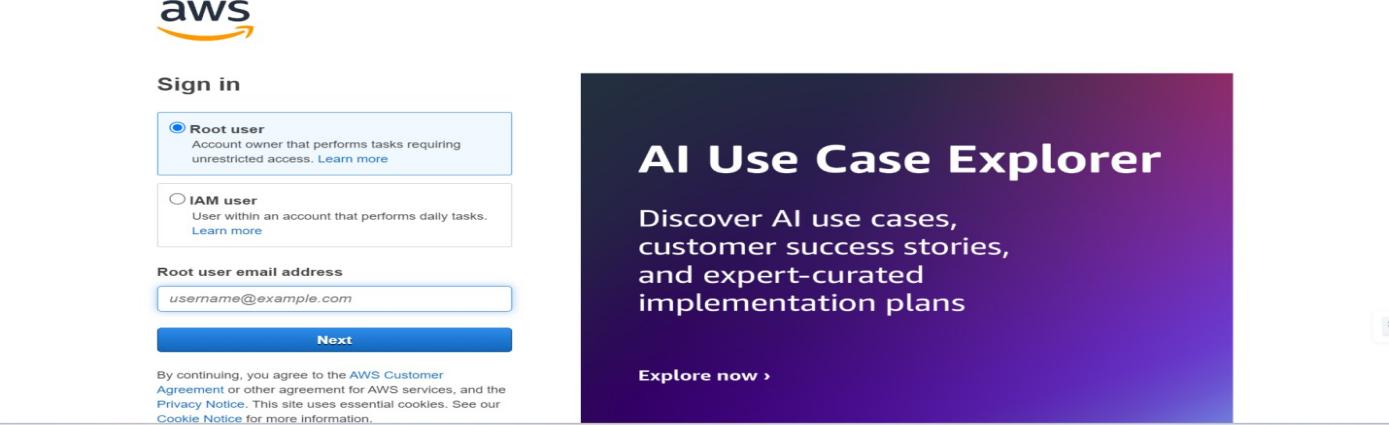
### Activity 1.1: Set up an AWS account if not already done.

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



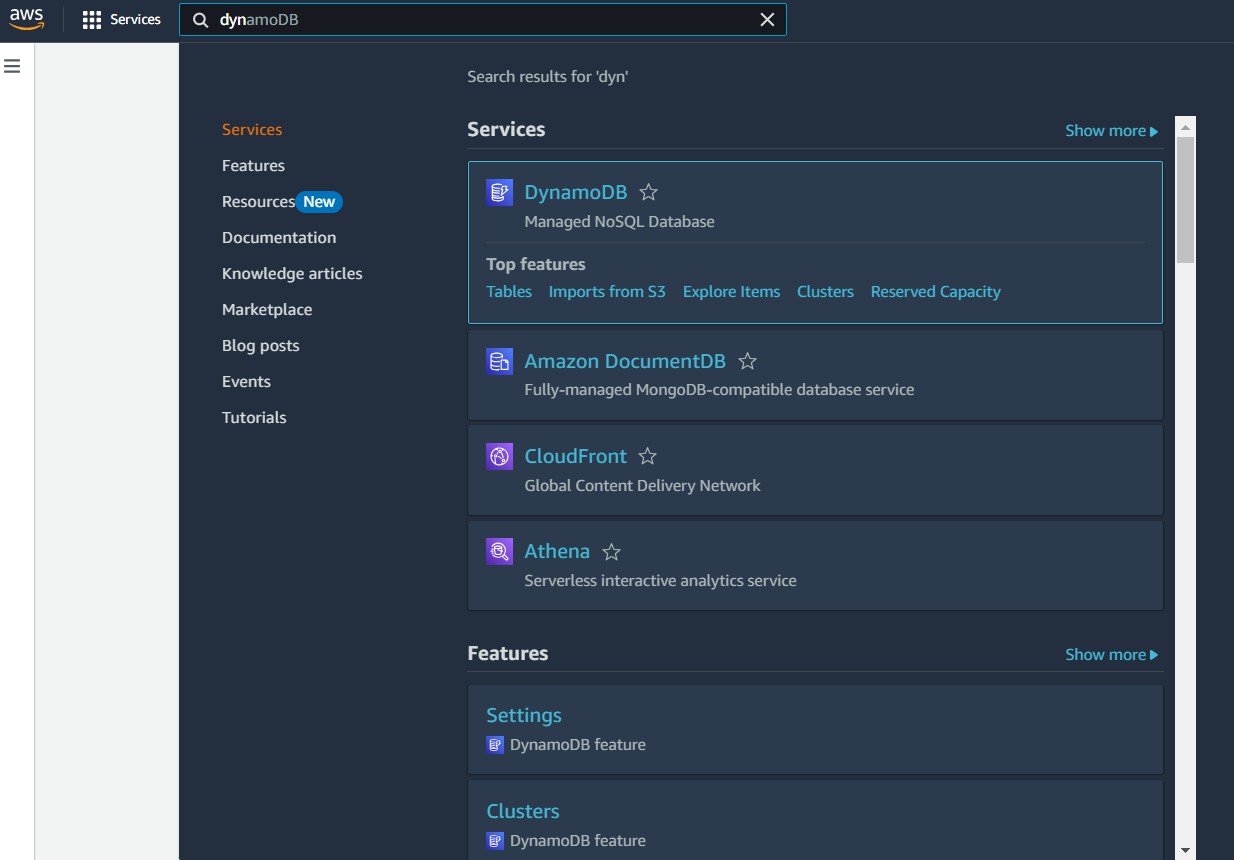
### Activity 1.2: Log in to the AWS Management Console

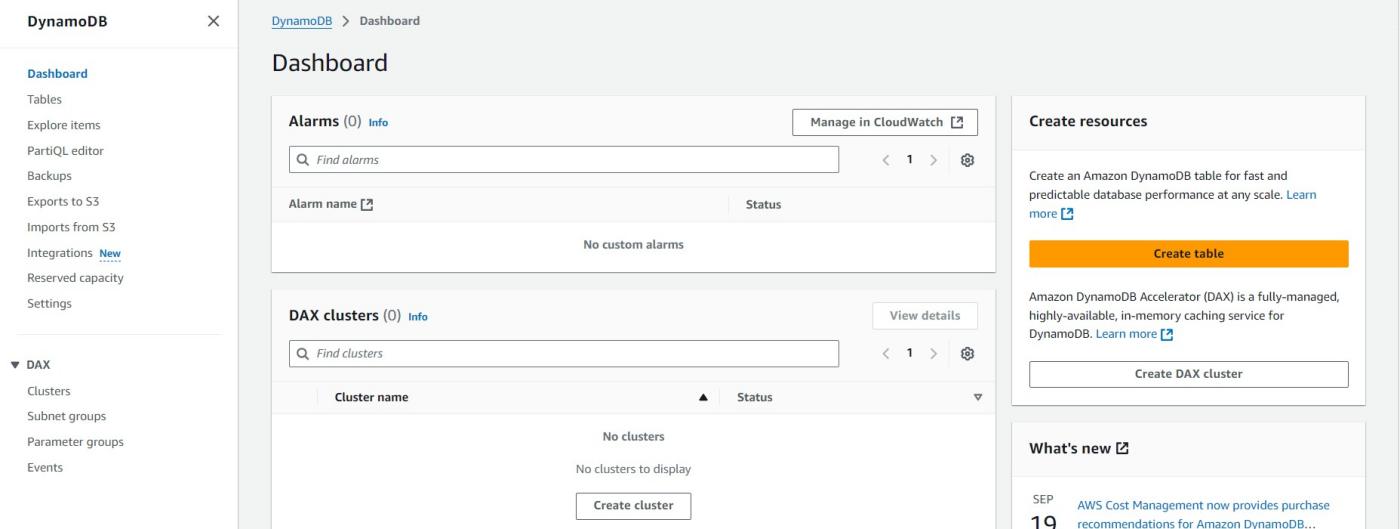
* + - After setting up your account, log in to the [AWS Management Console](https://aws.amazon.com/console/).



# Milestone 2: DynamoDB Database Creation and Setup

### Activity 2.1:Navigate to the DynamoDB

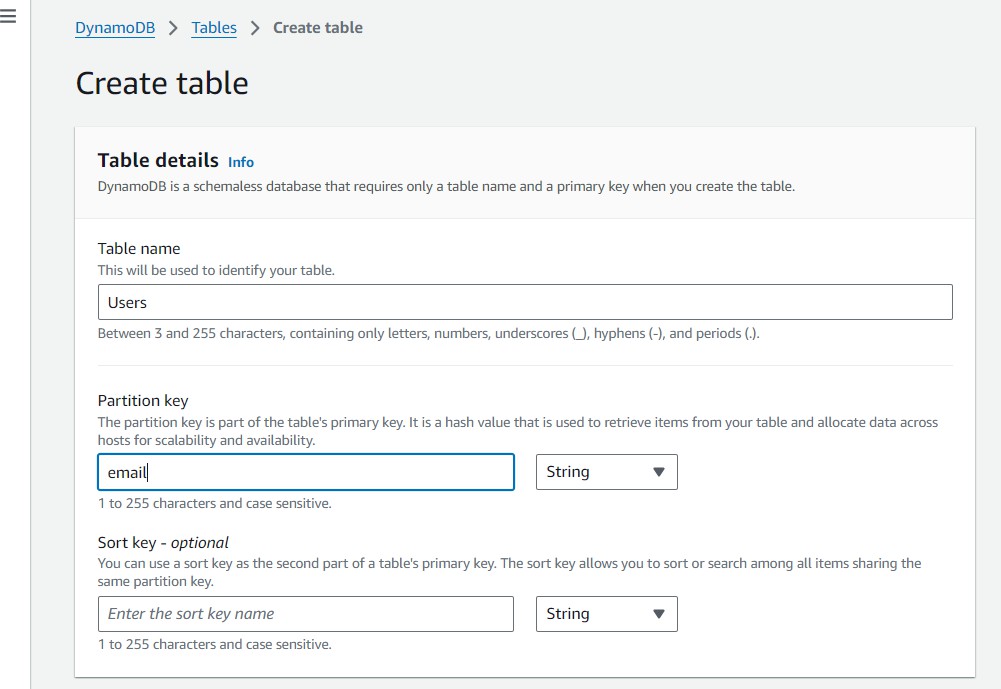
* + - In the AWS Console, navigate to DynamoDB and click on create tables.
    - 

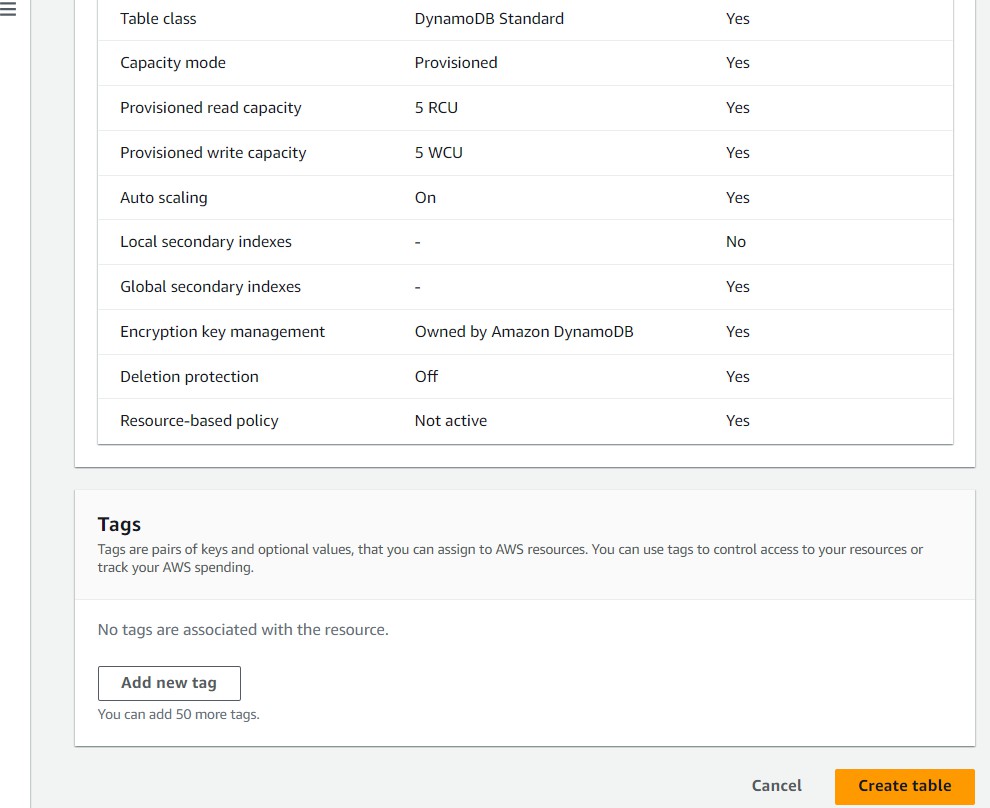


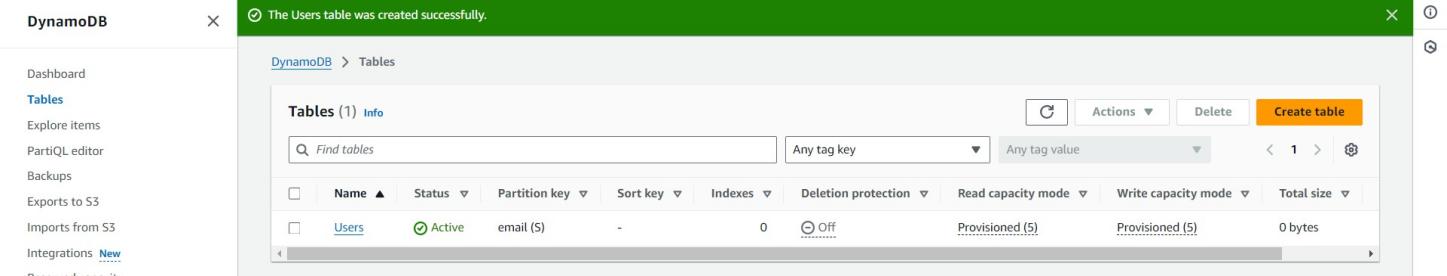


Activity 2.2:Create a DynamoDB table for storing user registration details, cart data, and customer reviews.

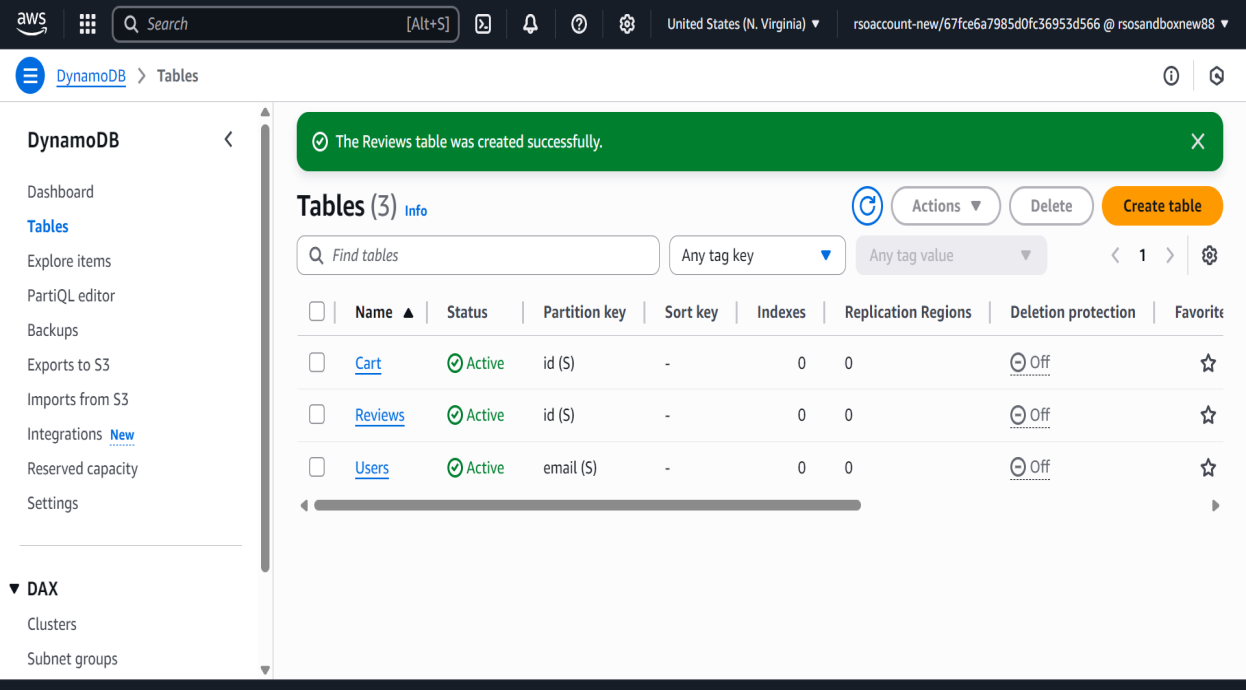
Create Users table with partition key “Email” with type String and click on create tables.





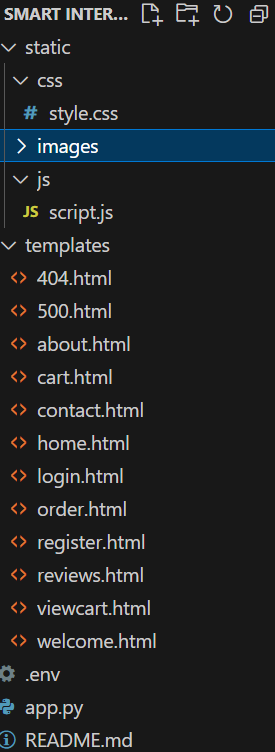


* + - Follow the same steps to create a reviews table with id as the primary key for customer reviews data and cart table to store the items in the cart.

\_--

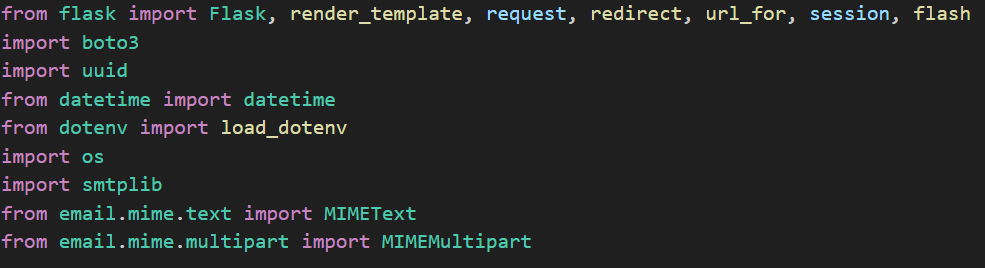
Milestone 4:Backend Development and Application Setup

### Activity 4.1: Develop the backend using Flask

* + File Explorer Strucure
  + 

**Description:**The project directory is organized into key folders and files essential for a Flask-based web application integrated with AWS. The app.py file is the core backend script that handles routing, session management, user authentication, and communication with AWS services like DynamoDB and SNS. The static folder contains subfolders for css (for styling), images (for visual assets like logos or product pictures), and js (for any frontend interactivity using JavaScript). The templates folder holds HTML files rendered by Flask, including home.html (the main page after login), login.html (for user authentication), register.html (for new user sign-up), and welcome.html (the initial landing page shown before login). This structure ensures a clean separation of frontend and backend components, enabling efficient development and deployment of the web application.

### Flask App Initialization

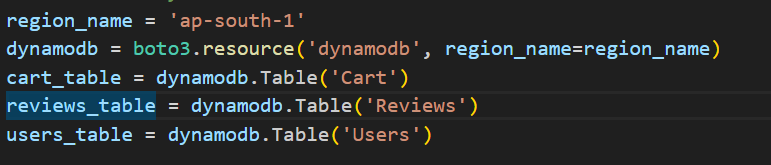


**Description:**This image displays the import section of your app.py file, which includes all the essential libraries required for your Flask web application integrated with AWS and email services. It starts by importing core Flask modules such as Flask, render\_template, request, redirect, url\_for, session, and flash for routing, rendering HTML templates, handling form submissions, and managing user sessions. The boto3 library is used to interact with [AWS services](https://chatgpt.com?q=AWS services), particularly [DynamoDB](https://chatgpt.com?q=DynamoDB) and [SNS](https://chatgpt.com?q=SNS). The uuid module generates unique identifiers for user data and requests, while datetime handles timestamps. dotenv is used to load environment variables securely via the .env file. The os module accesses environment variables and file paths. Finally, smtplib, MIMEText, and MIMEMultipart are imported to facilitate sending email notifications, such as order confirmations, using SMTP. Together, these imports enable your app to handle backend logic, database interactions, and communication functionalities securely and efficiently.



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

### Dynamodb Setup:



**Description:**This snippet from your app.py file demonstrates how you're connecting to three different [DynamoDB](https://chatgpt.com?q=DynamoDB) tables in the AWS **ap-south-1** region using the boto3 library. The line dynamodb = boto3.resource('dynamodb', region\_name=region\_name) establishes a resource-level connection to DynamoDB. Then, the tables are assigned to variables:

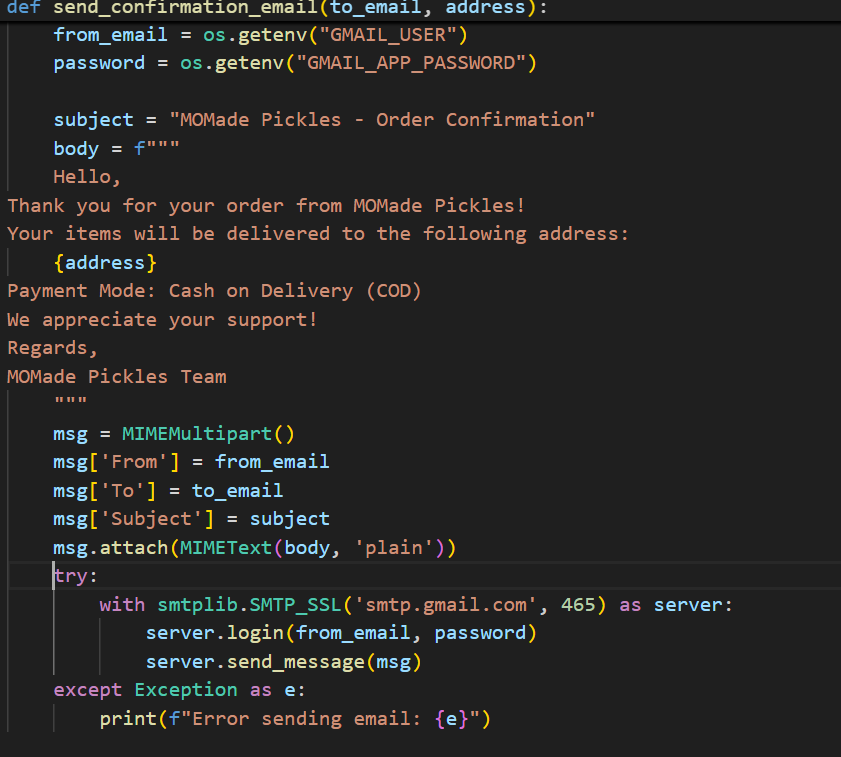
cart\_table = dynamodb.Table('Cart'): Handles items added to the cart by users.

reviews\_table = dynamodb.Table('Reviews'): Stores customer reviews and feedback.

users\_table = dynamodb.Table('Users'): Manages user registration and login data.

This structure ensures modular and clear access to each table, allowing smooth database operations throughout your Flask app.

### Email Confirmation:



**Description:** The send\_confirmation\_email(to\_email, address) function is responsible for sending a personalized order confirmation email to users after they successfully place an order on the MOMade Pickles platform. It retrieves the sender's Gmail credentials securely using environment variables (GMAIL\_USER and GMAIL\_APP\_PASSWORD) to protect sensitive information. The function constructs a message that includes a subject and a plain-text body containing order details and the delivery address, ensuring that customers receive all necessary information. Using Python’s smtplib and email.mime modules, the function creates a secure connection to Gmail’s SMTP server (smtp.gmail.com on port 465) via SSL. After logging in with the provided credentials, it sends the composed email to the recipient. Additionally, the function handles any errors that may occur during the process, providing feedback through exception handling. This ensures a smooth and secure communication channel with the customer, reinforcing trust and enhancing the overall user experience.

## Routes for Web Pages

### View Cart Route:

### 

**Description:** The view\_cart() function is mapped to the /viewcart route and is designed to allow authenticated users to view items in their shopping cart. It begins by checking if the user’s email is present in the session; if not, it flashes a message prompting them to log in and redirects them to the login page. If the email is available in the session, the function queries the Cart table in [AWS DynamoDB](https://chatgpt.com/c/w) using the email as the key to retrieve all cart items associated with that user. The query uses the boto3.dynamodb.conditions.Key condition to fetch only the relevant records. The retrieved items are then stored in the cart\_items variable, which defaults to an empty list if no items are found. Finally, the function renders the viewcart.html template, passing the cart\_items as context so they can be displayed to the user. This logic ensures secure and dynamic retrieval of cart data based on the logged-in user's session.

### Order Route:



**Description:**Here's a slightly longer version that's still concise but more detailed:

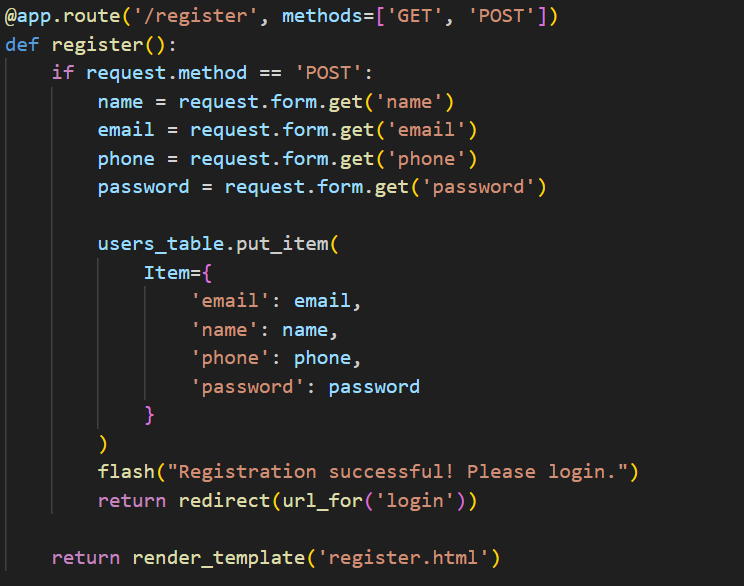
The order() function is responsible for handling the order placement process. It begins by checking if the user is logged in using session data. If the user is not authenticated, they are redirected to the login page. When the method is POST, the function retrieves the user's email from the session and the entered delivery address from the form. It then queries the Cart table in DynamoDB to fetch all cart items associated with that email. For each item found, it deletes the item from the cart to simulate order processing. After clearing the cart, it calls the send\_confirmation\_email() function to notify the user with the delivery address and order confirmation. Finally, it displays a flash message indicating successful order placement and redirects the user to the home page.

### Reviews Route :

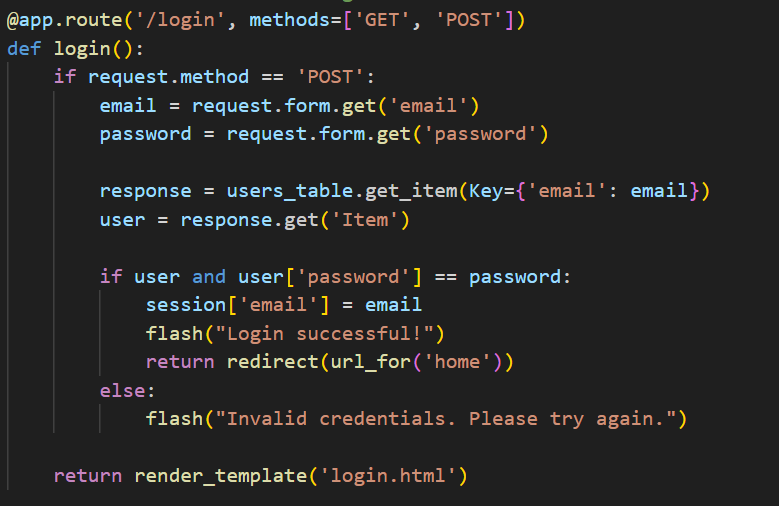


**Description:** The reviews() function manages the submission of customer reviews. When a user submits a POST request, it first checks if the user is logged in by verifying the session. If not, the user is redirected to the login page. Upon successful authentication, the function retrieves the user's name and message from the form, along with their email from the session. It then stores this information—along with a timestamp—into the Reviews table in DynamoDB. After saving the review, it flashes a thank-you message and redirects back to the reviews page. Additionally, the page displays a list of dummy reviews using the dummy\_reviews list for demonstration purposes.

### Register Route:



**Description:**This Flask Python code handles user registration at the /register route, supporting both GET and POST methods. For POST requests, it extracts 'name', 'email', 'phone', and 'password' from the submitted form, then stores this data as a new item in a users\_table (likely a database). Upon successful data storage, it flashes a "Registration successful!" message and redirects the user to the login page; otherwise, for GET requests, it renders the register.html template to display the registration form.

* **Login Routes:**
* 

**Description:**This Python Flask code defines a /login route that handles user authentication. When a POST request is received, it retrieves the 'email' and 'password' submitted in the form. It then attempts to fetch a user record from the users\_table using the provided email as the key. If a user is found and the stored password matches the provided password, the user's email is stored in the session, a "Login successful!" message is flashed, and the user is redirected to the 'home' page. If the credentials do not match, an "Invalid credentials" message is flashed. For GET requests, the code renders the login.html template, which likely displays the login form.

### Other Routes:

### 

**Description:** This Python Flask code establishes four distinct routes to serve static web pages: the root URL (/) renders welcome.html as the landing page, /home displays home.html for the main content, /contact shows contact.html for contact information, and /about presents about.html for details about the application or organization. These routes collectively define the foundational navigation and content delivery for a Flask-based web application.

**Deployment Code:**

****

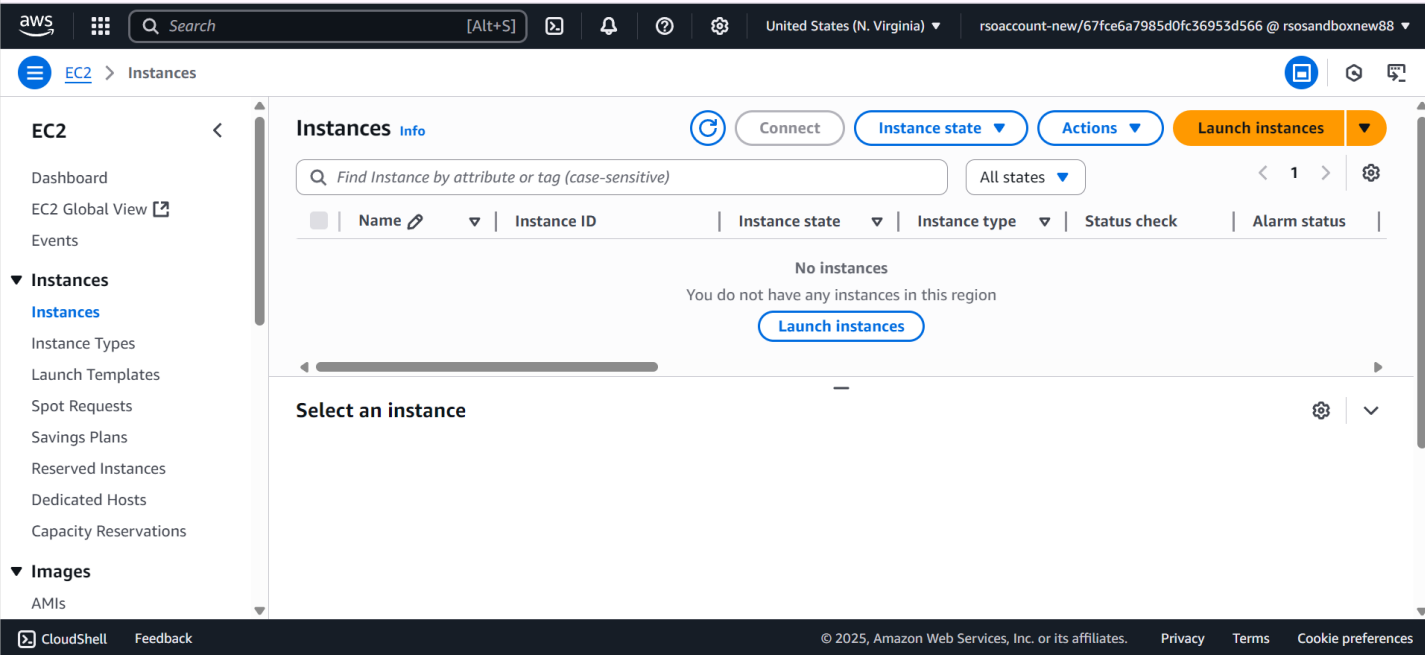
**Description:** start the Flask server to listen on all network interfaces (0.0.0.0) at port 80

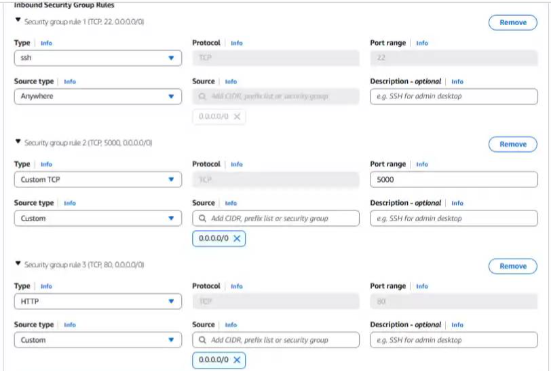
with debug mode enabled for development and testing.

**Milestone 3:**

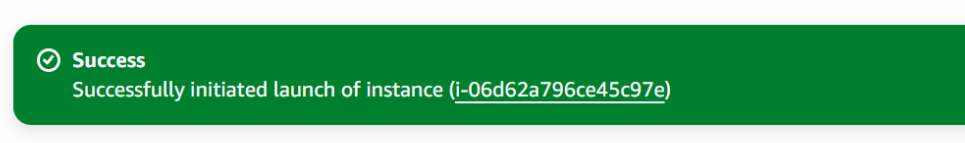
Create an instance in EC2.

3.1:click on the launch instance button



3.2: 

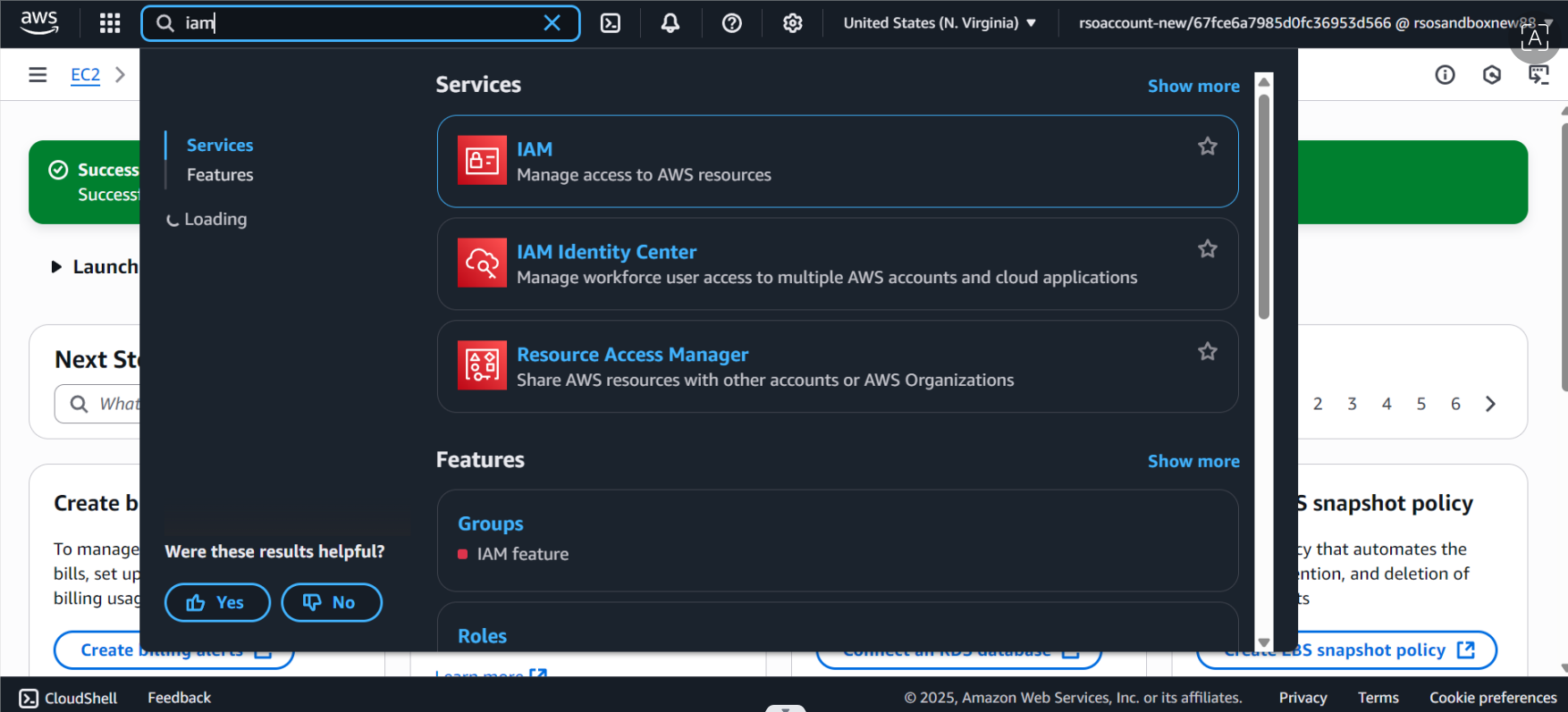
Verification :



**Milestone 4:IAM**

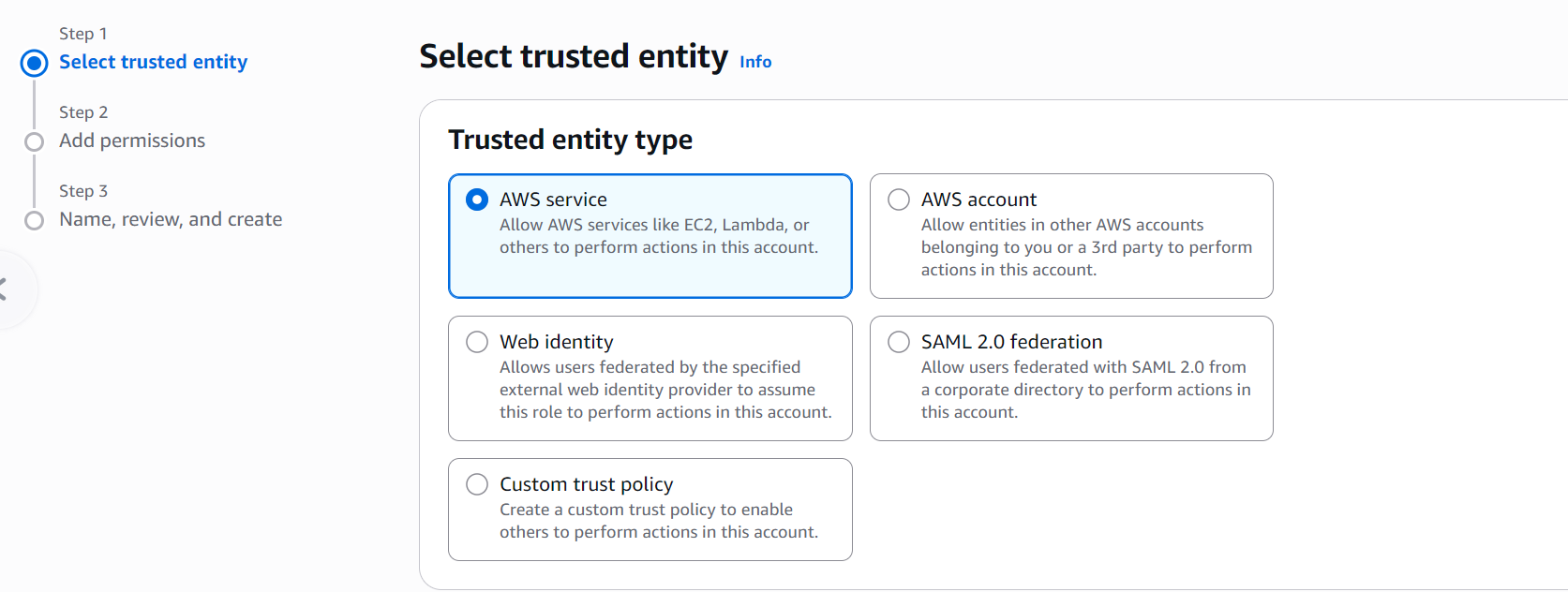
**Activity 4.1:**

**Search for IAM in AWS console.**

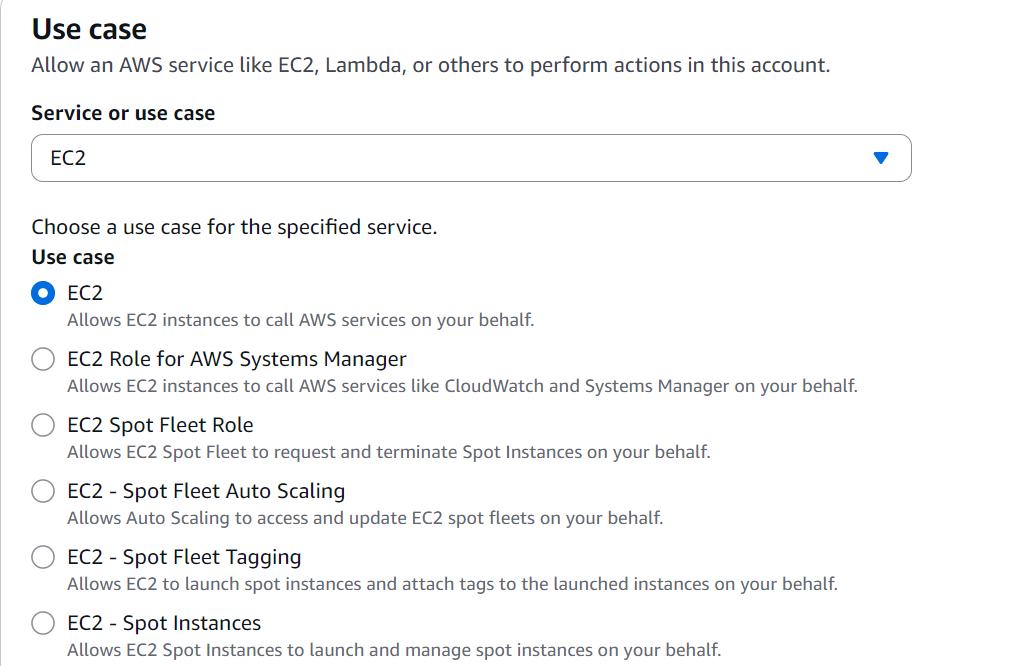


**Activity 4.2:**

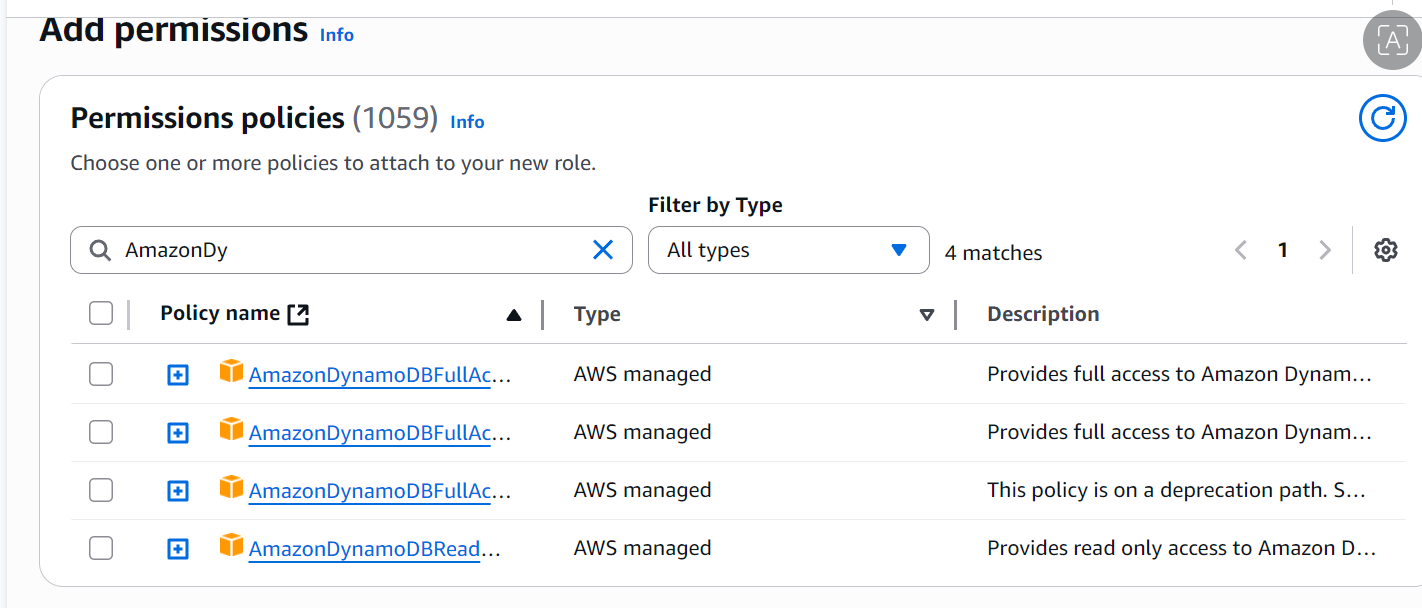
**Select trusted entity as AWS service.**



Activity 4.3:select ec2 as use case

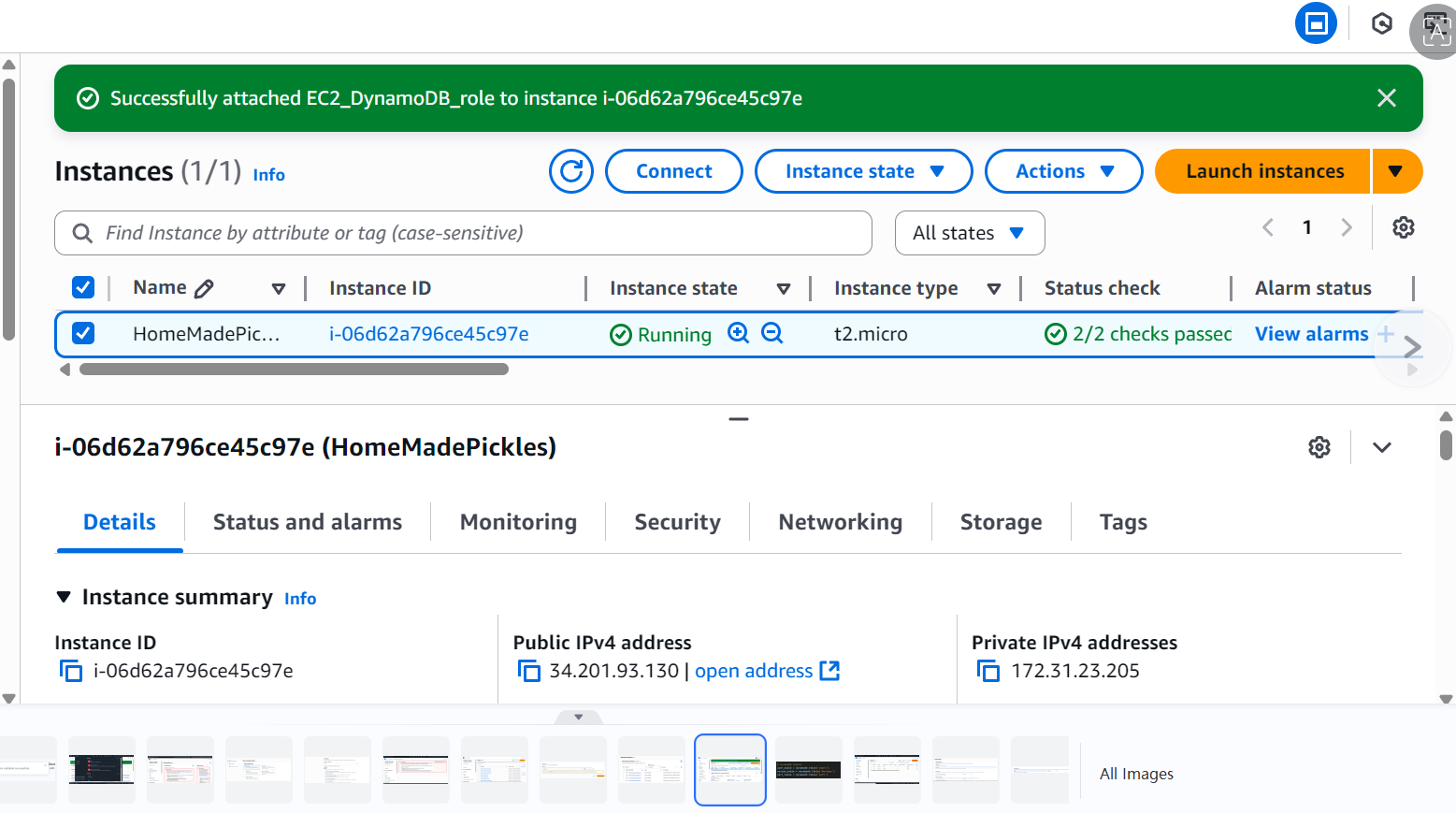


Activity 4.4:

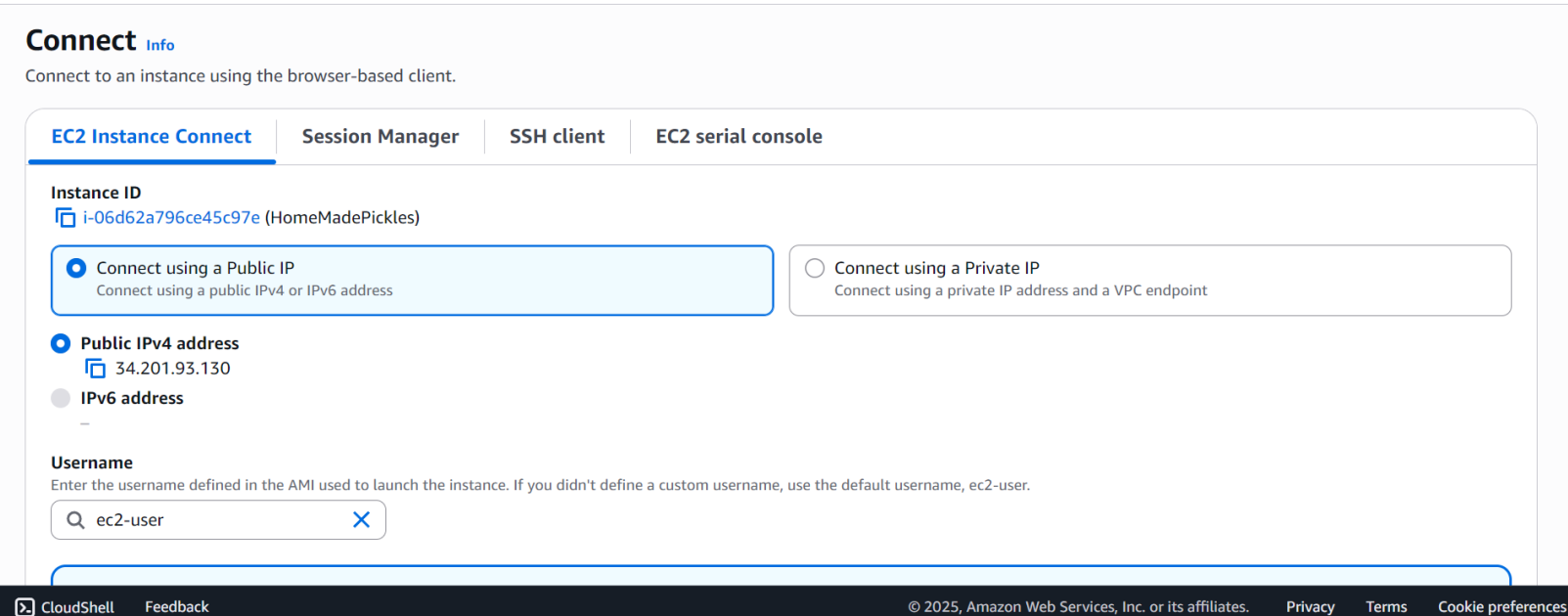


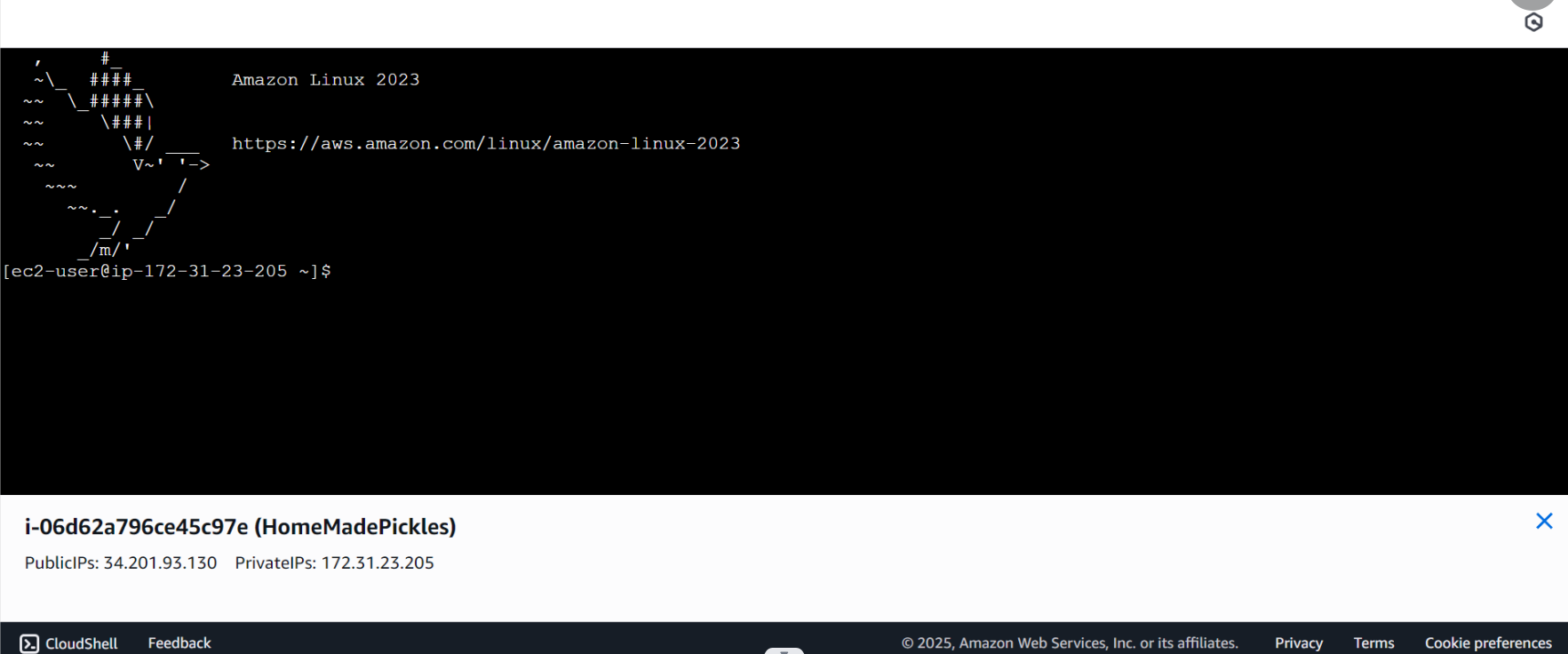
**MileStone 5:**

Open aws console, search for ec2 and modify the role to EC2\_DynamoDB\_role.



**MileStone 6: click on connect and open the terminal**





**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: ‘git clone <https://github.com/your-github-username/your-repository-name.git>’

Note: change your-github-username and your-repository-name with your credentials here: ‘git clone https://github.com/HarshithaPandranki/smart\_interns\_project.git

* This will download your project to the EC2 instance.

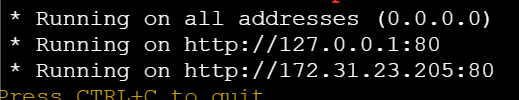
### To navigate to the project directory, run the following command:

cd smart\_interns\_project.git

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

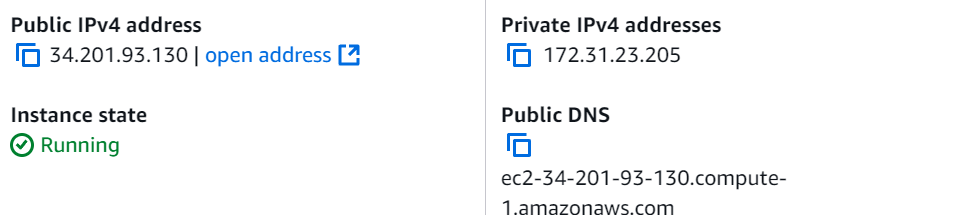
**Run the Flask Application**

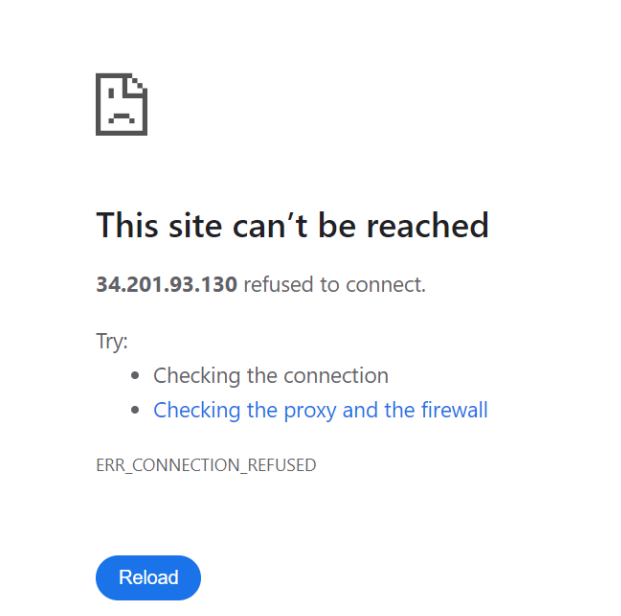
sudo flask run --host=0.0.0.0 --port=5000



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

* Run the Flask app on the EC2 instance





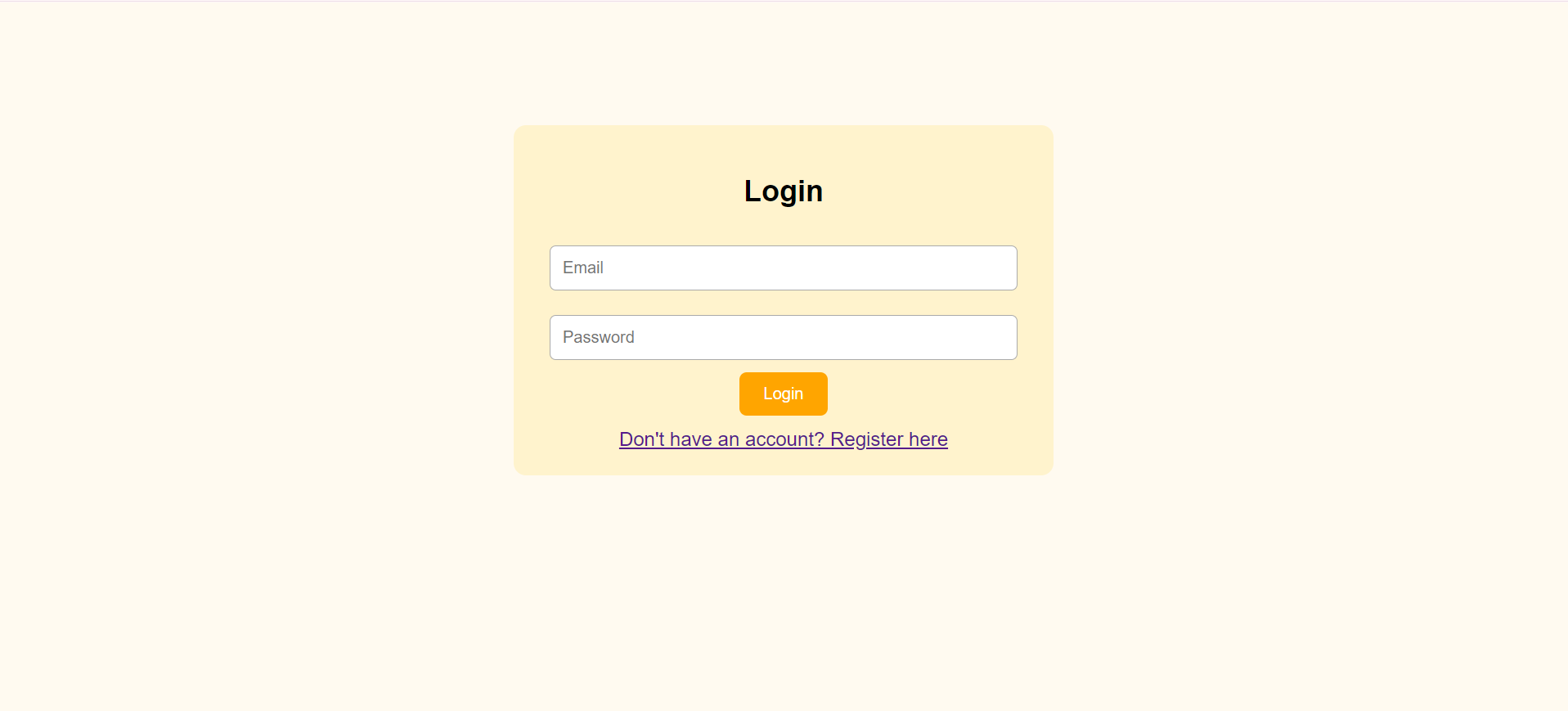
### Access the website through(locally deployed):

http://127.0.0.1:5000

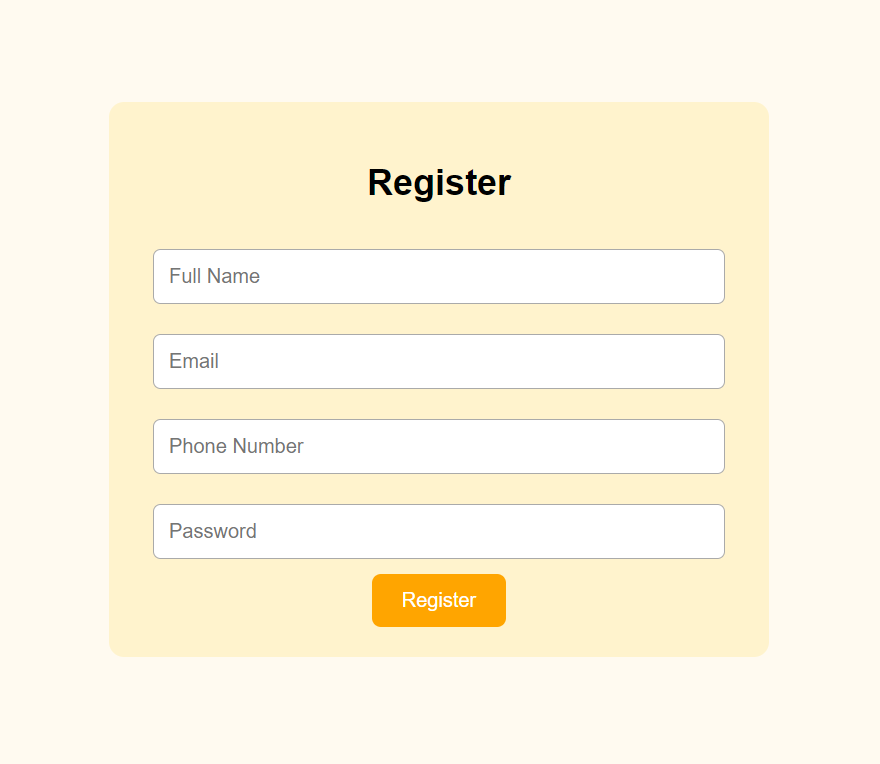
## Milestone 8: Testing and Deployment

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

**Login Page:**



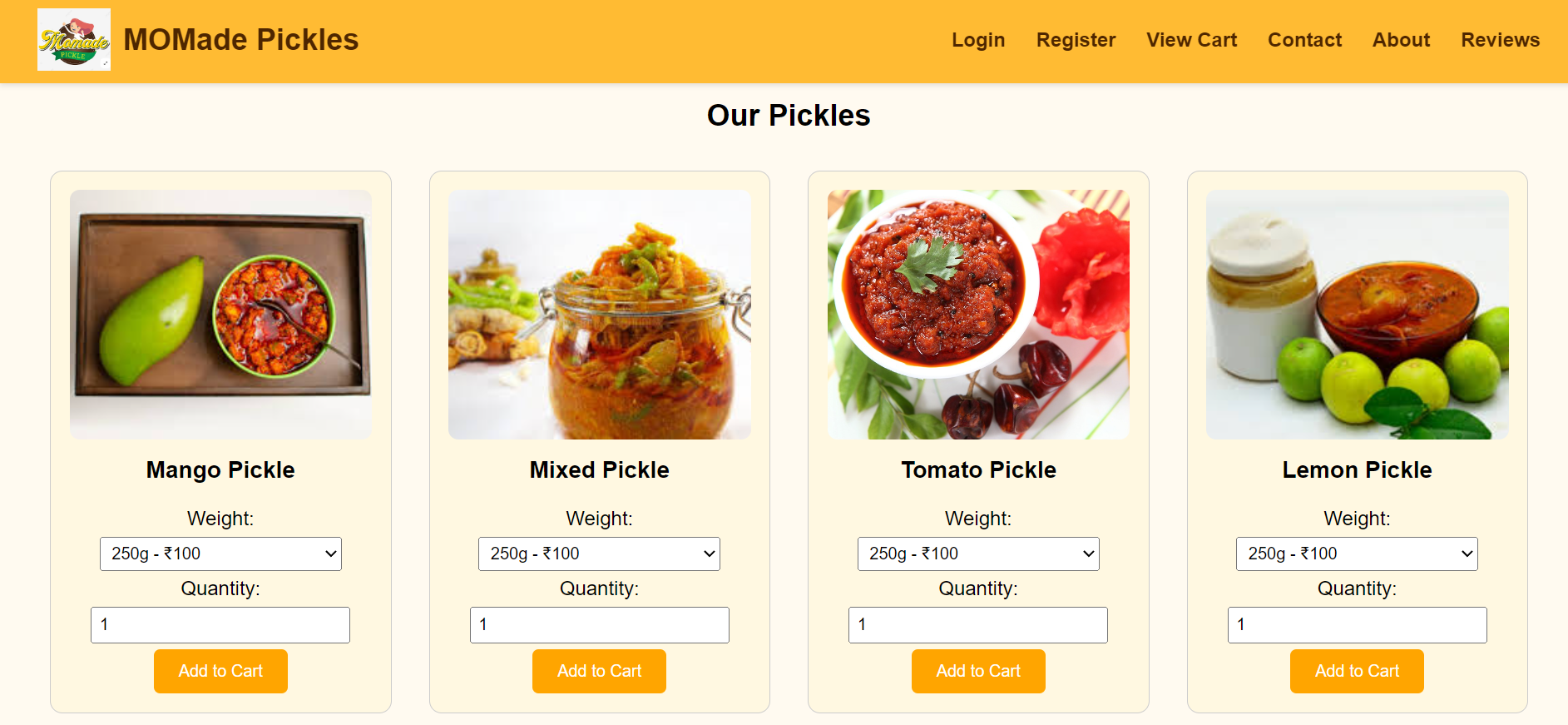
**Register Page:**



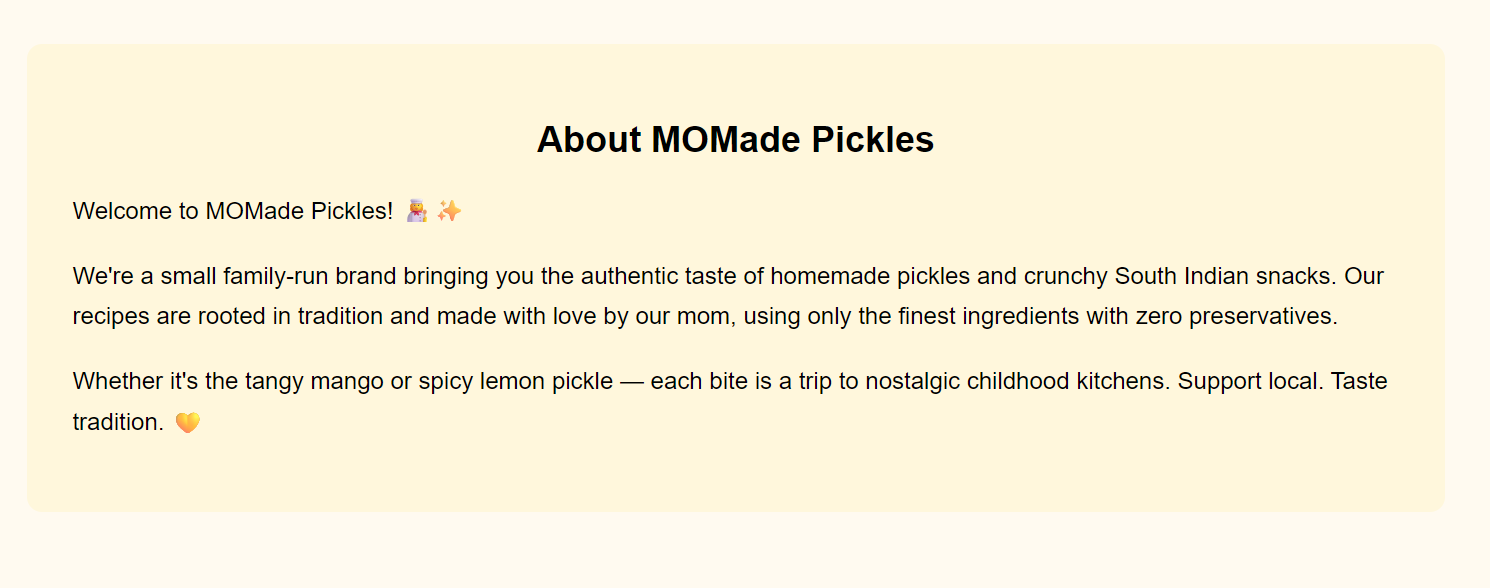
**Welcome page :**



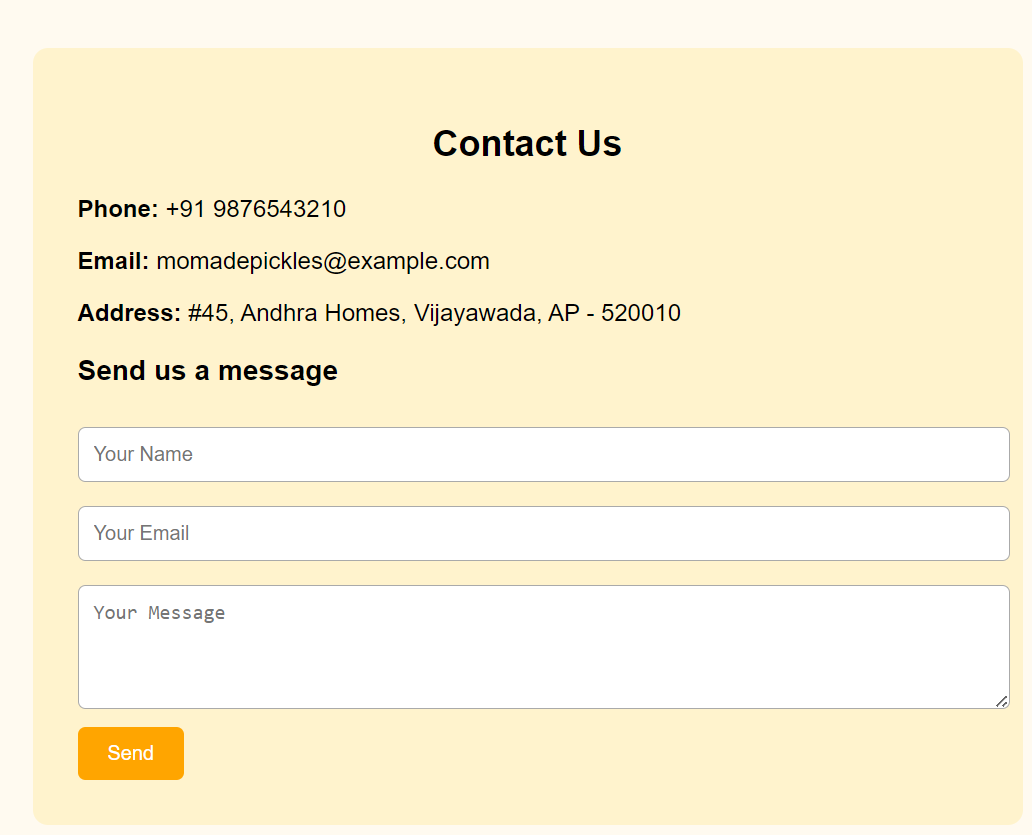
**Home page:**



**About Us page:**



**Contact Page:**



**Review page:**

