HomeMade Pickles & Snacks: Taste The Best

Project Description:

HomeMade Pickles & Snacks Taste the Best is a cloud-based culinary platform revolutionizing access to authentic, handcrafted pickles and snacks. Addressing the growing demand for preservative-free, traditional recipes, this initiative combines artisanal craftsmanship with cutting-edge technology to deliver farm-fresh flavors directly to consumers. Built on Flask for backend efficiency and hosted on AWS EC2 for scalable performance, the platform offers seamless browsing, ordering, and subscription management. DynamoDB ensures real-time inventory tracking and personalized user experiences, while fostering sustainability through partnerships with local farmers and eco-friendly packaging. From tangy regional pickles to wholesome snacks, every product celebrates heritage recipes, nutritional integrity, and convenience-proving that tradition and innovation can coexist deliciously. "Preserving Traditions, One Jar at a Time."

Scenario 1: Scalable Order Management for High Demand

A cloud-based system ensures seamless order processing during peak user activity. For instance, during a promotional event, hundreds of users simultaneously access the platform to place orders. The backend efficiently processes requests, updates inventory in real-time, and manages user sessions. The cloud infrastructure handles traffic spikes without performance degradation, ensuring smooth transactions and minimizing wait times.

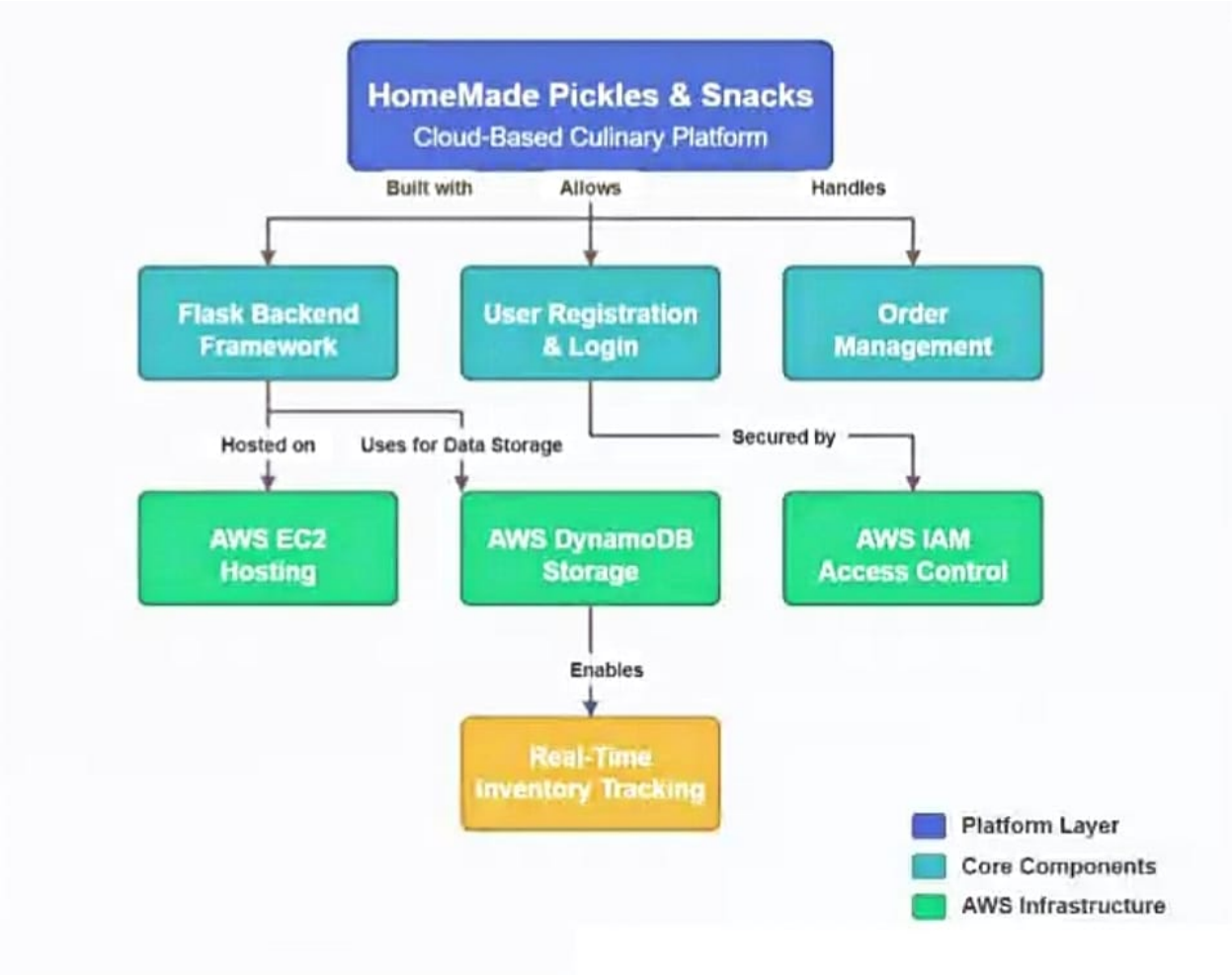
Scenario 2: Real-Time Inventory Tracking and Updates

When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and fulfillment progress, ensuring timely restocking and minimizing overselling risks.

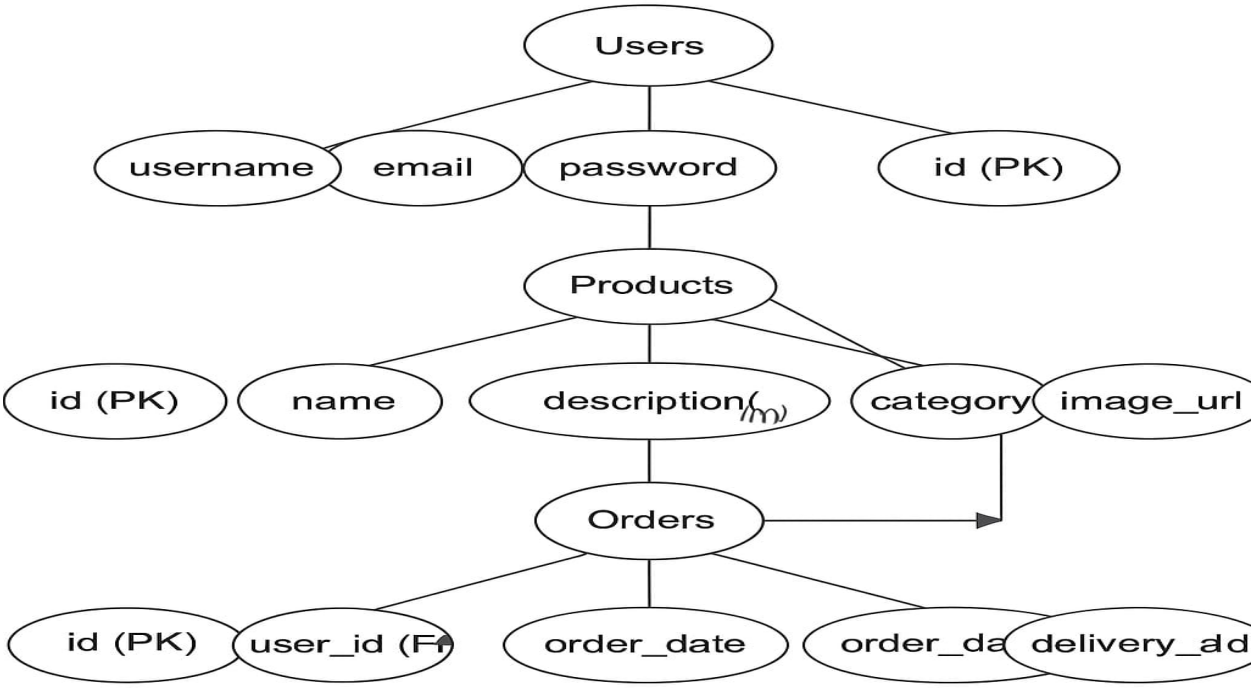
Scenario 3: Personalized User Experience and Recommendations:

The platform leverages user behavior data to enhance engagement. A returning customer, for instance, views tailored recommendations based on past purchases and browsing history. The system dynamically adjusts suggestions in real-time, while maintaining fast response rates even during high traffic, creating a frictionless and intuitive shopping experience.

AWS ARCHITECTURE :



Entity Relationship(ER) Diagram :



Pre-requisites :

1. AWS Account Setup: AWS Account Setup
2. Understanding IAM: IAM Overview
3. Amazon EC2 Basics: EC2 Tutorial
4. DynamoDB Basics: DynamoDB Introduction
5. Git Version Control: Git Documentation

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 DatabaseCreation and Setup

Activity 2.1: Create a DynamoDB Table.

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

3.SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

4.Backend Development and Application Setup

Activity 4.1:Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

5.IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

6.EC2 Instance Setup

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

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

7.Deployment on EC2

Activity 7.1:Upload Flask Files

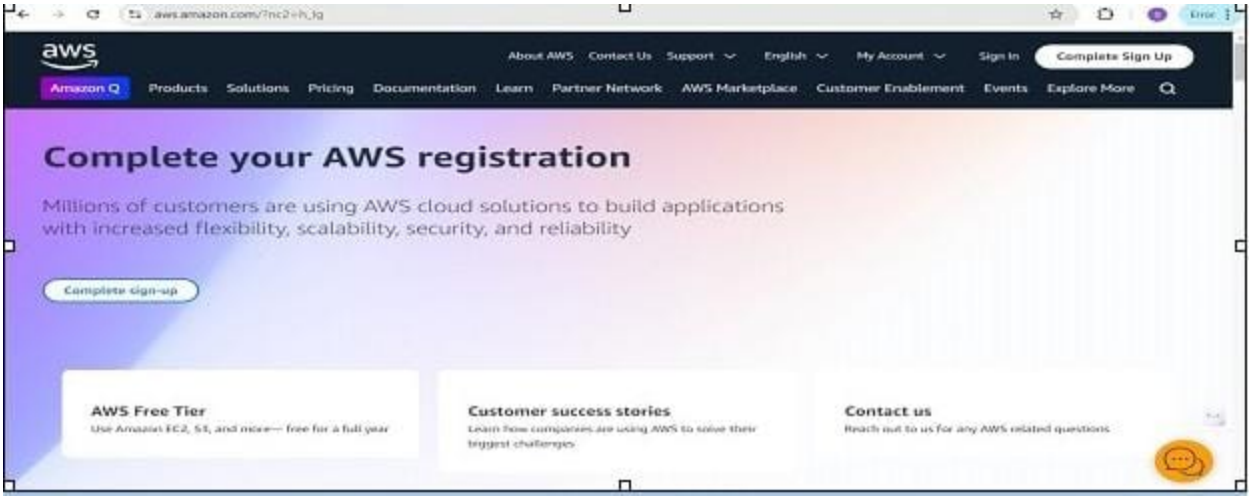
Activity 7.2: Run the Flask App

8.Testing and Deployment

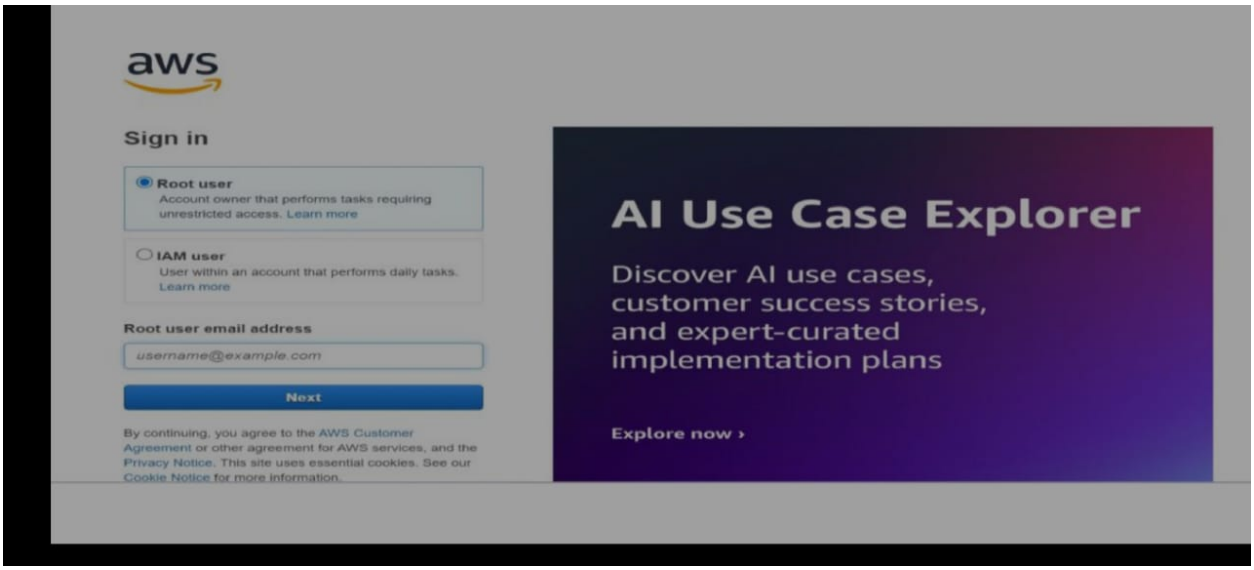
Activity 8.1: Conduct functional testing to verify user registration, login, Pickles and snacks orders, and notifications.

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

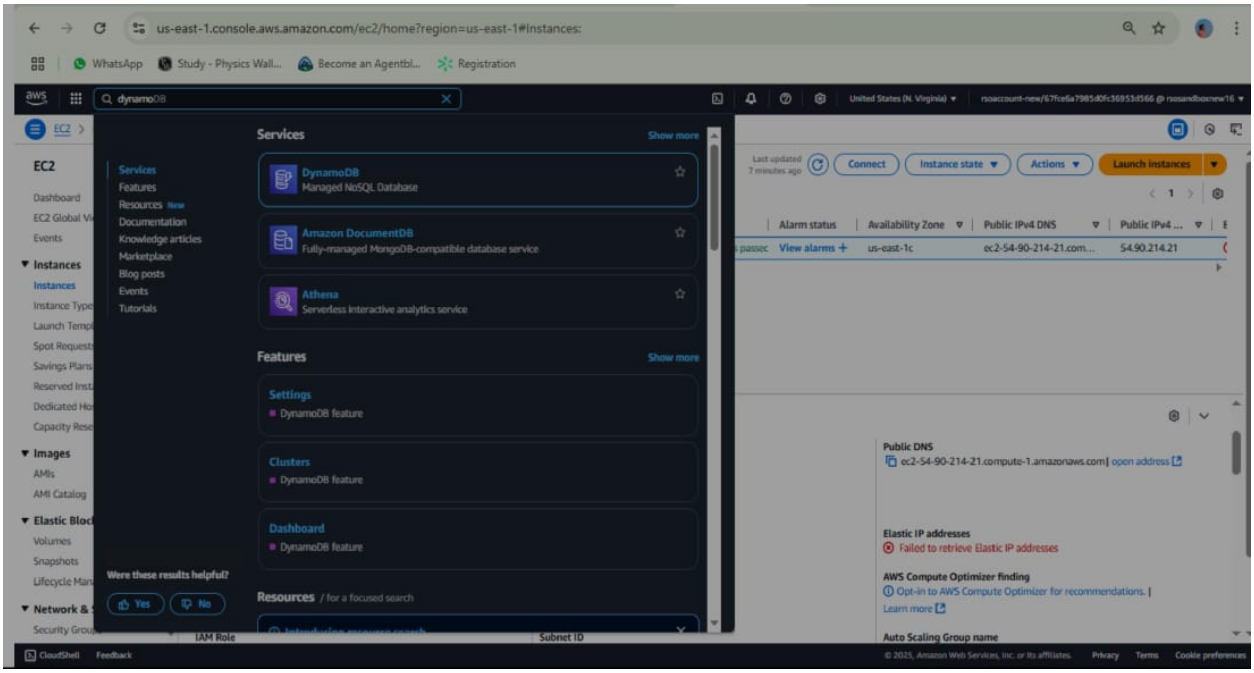


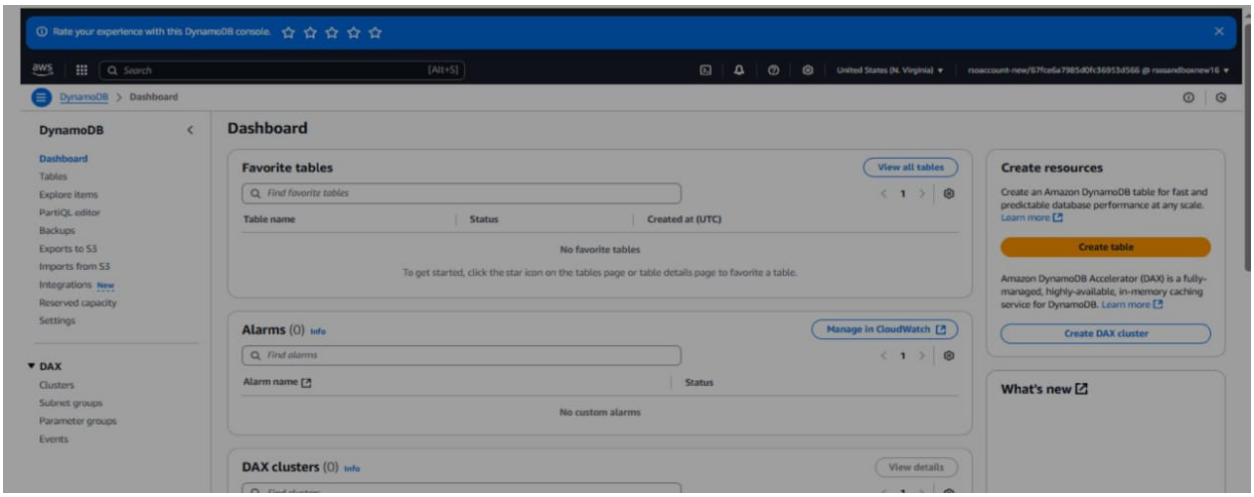
- **Activity 1.2: Log in to the AWS Management Console**
- After setting up your account, log in to the AWS Management Console.

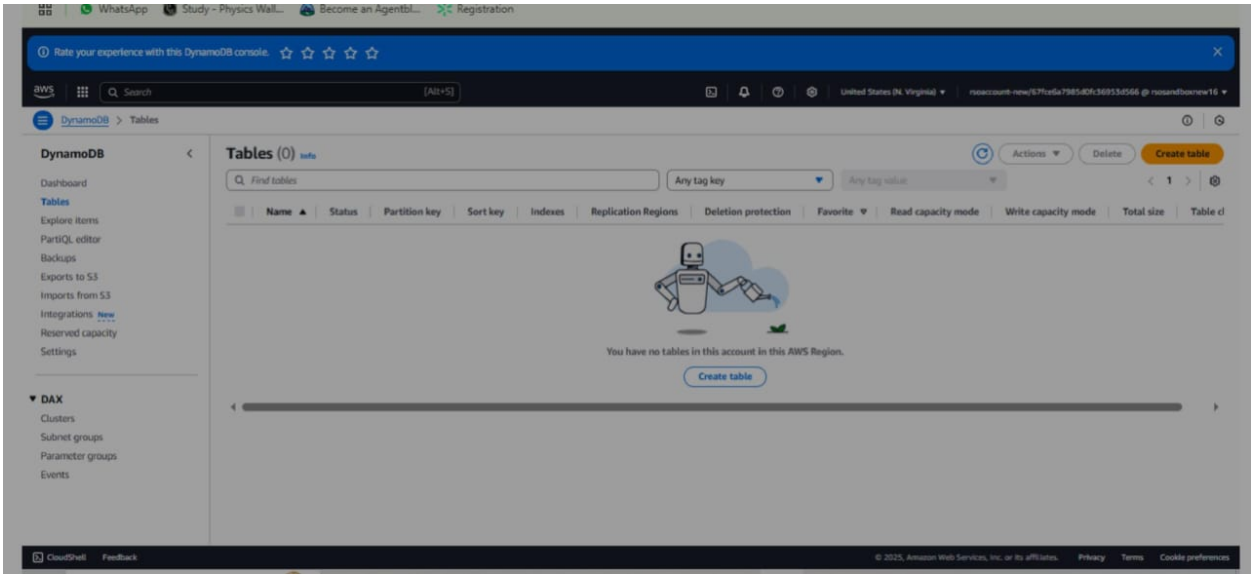


Milestone 2: DynamoDB Database Creation and Setup

- [ ] **Activity 2.1:Navigate to the DynamoDB**
- [ ] In the AWS Console, navigate to DynamoDB and click on create tables.

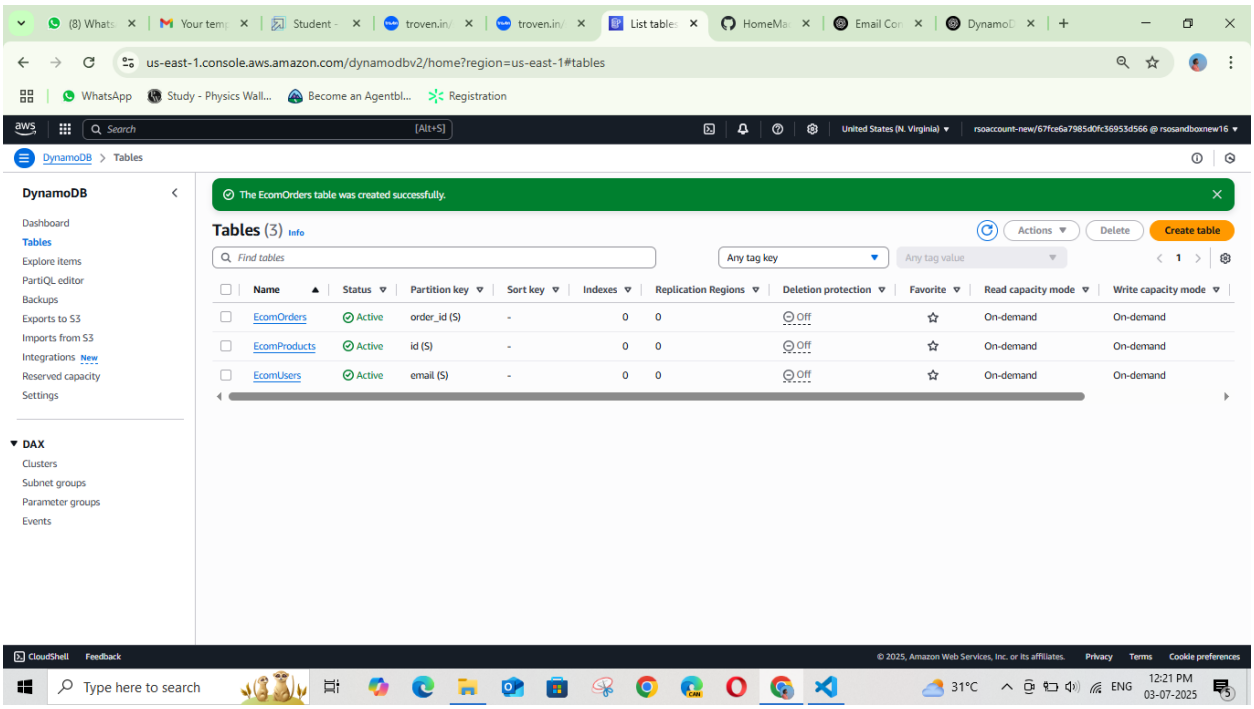






☐ **Activity 2.2: Create a DynamoDB table for storing registration details and pickles and snacks request.**

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

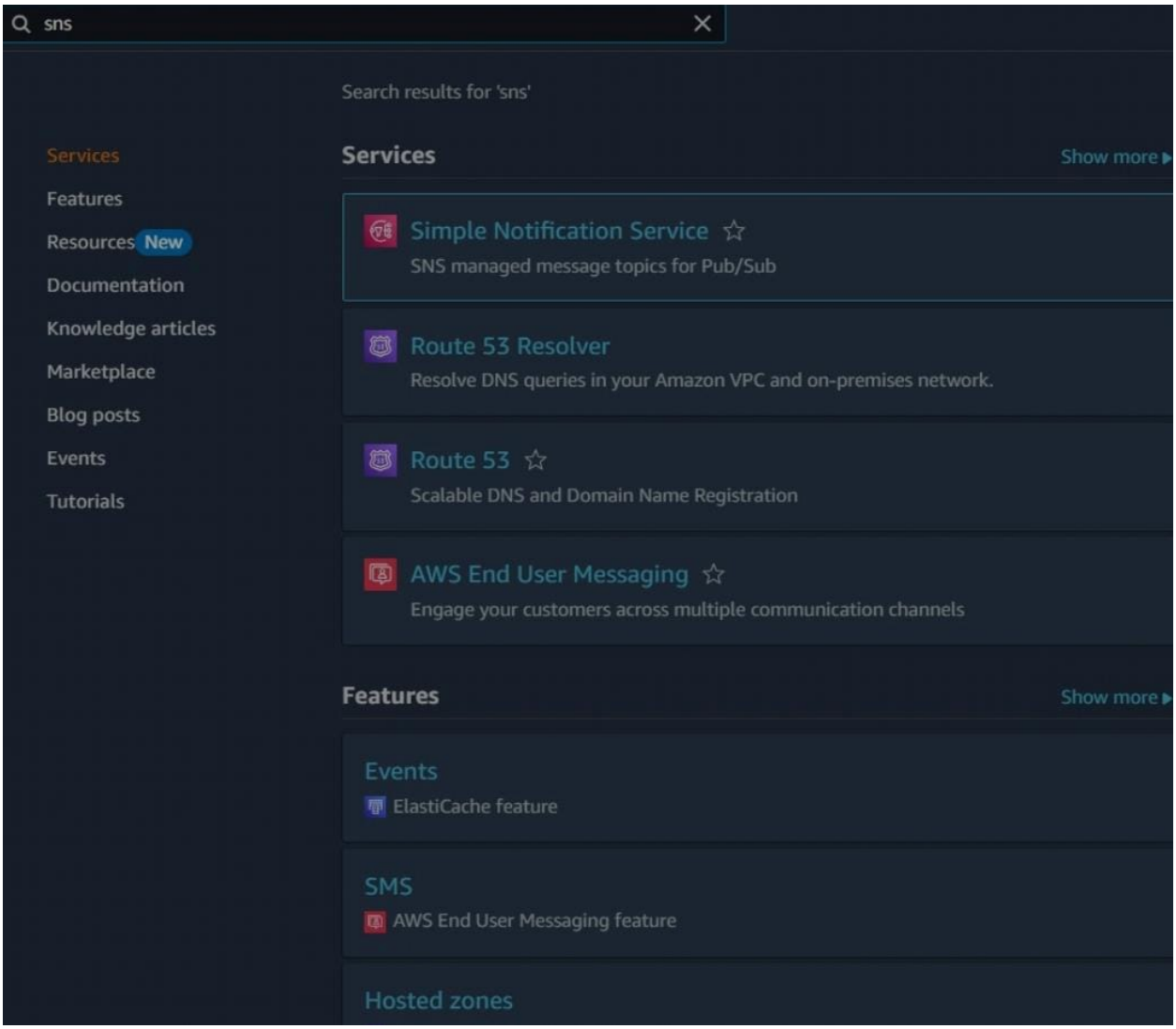


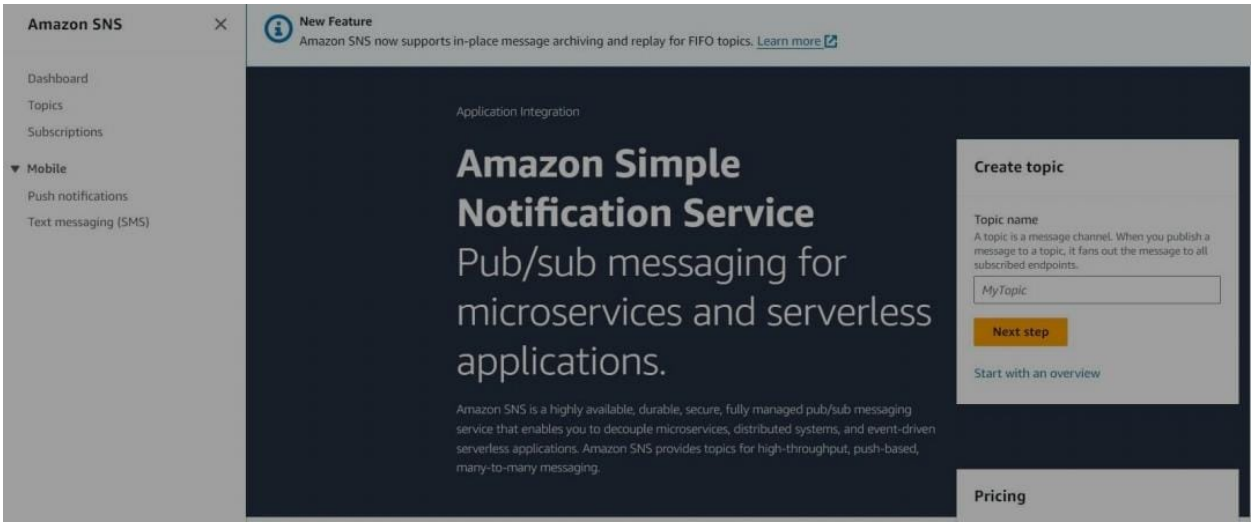
☐ Follow the same steps to create a orders table with Email as the primary key for pickles order data.

Milestone 3: SNS Notification Setup

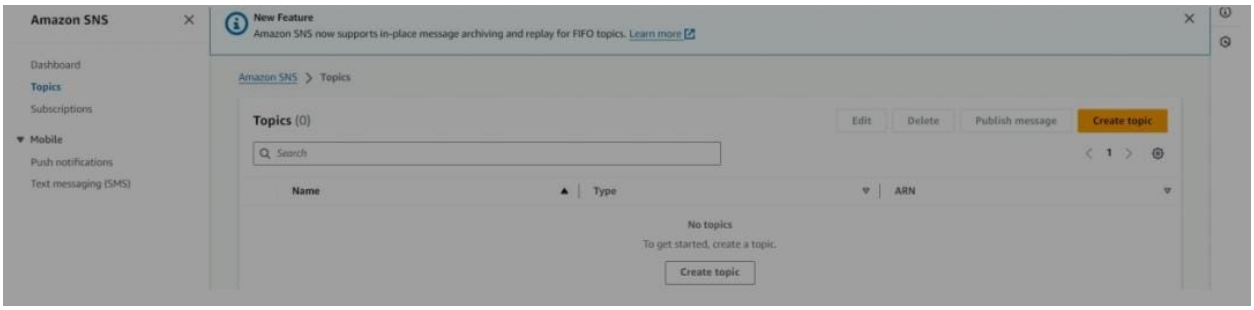
☐ **Activity 3.1**: **Create SNS topics for sending email notifications to users and Customers .**

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

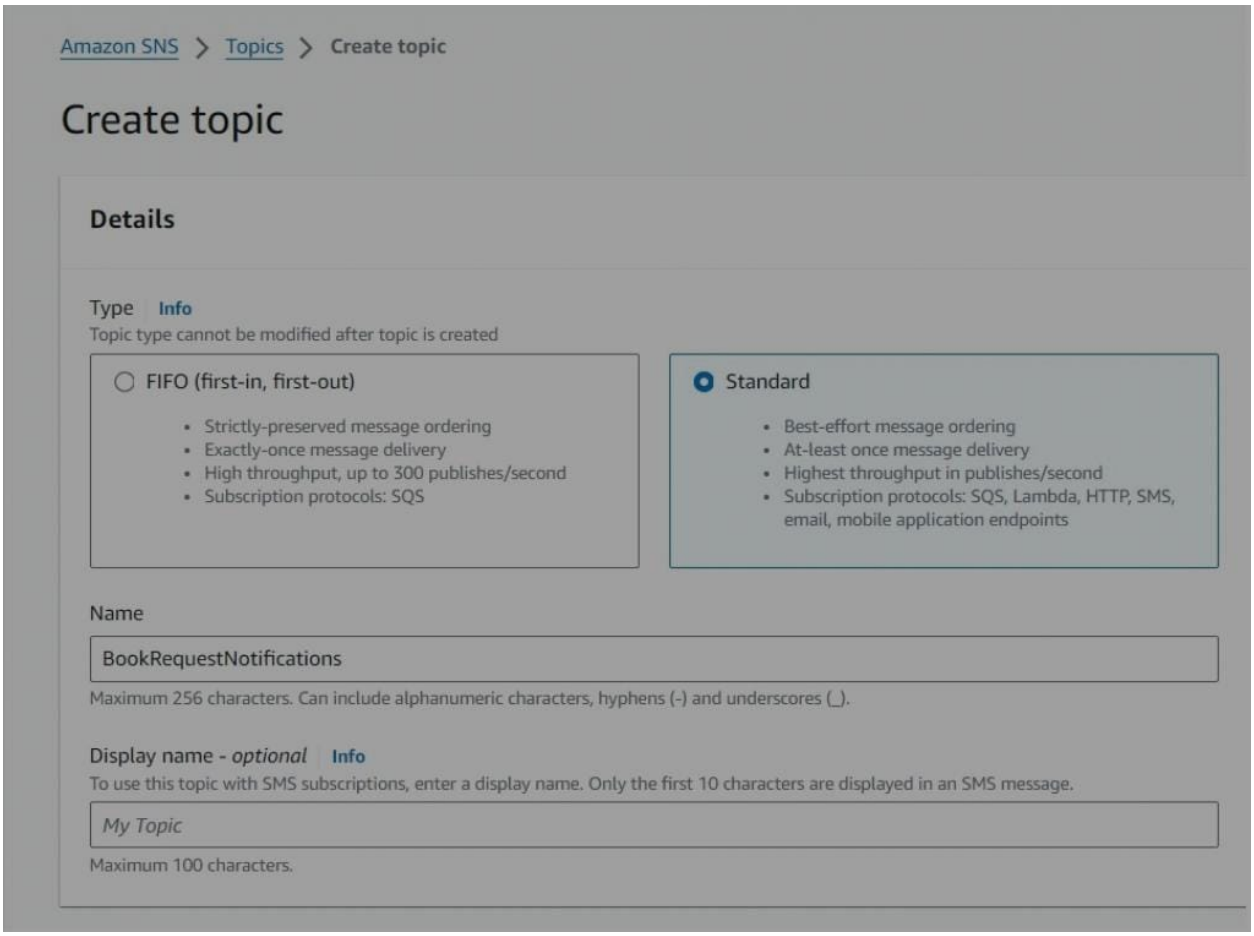


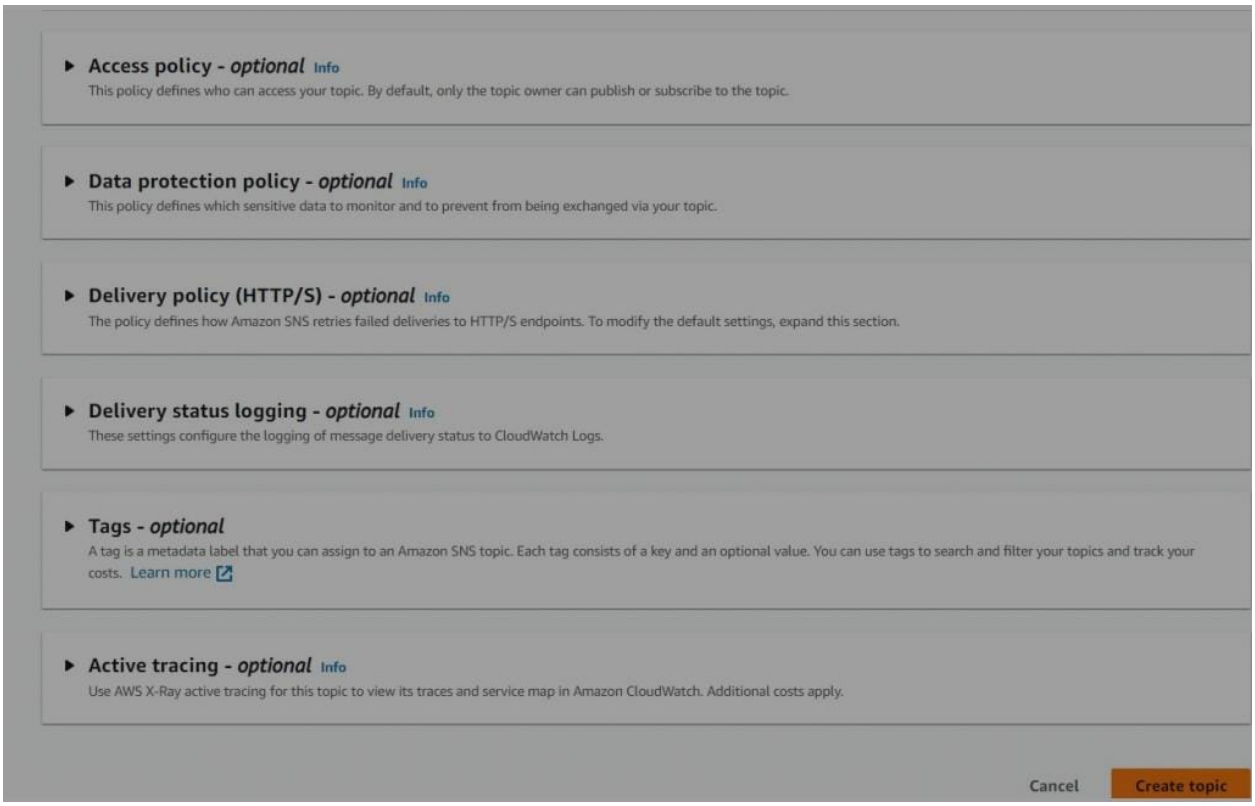


- [ ] Click on Create Topic and choose a name for the topic.

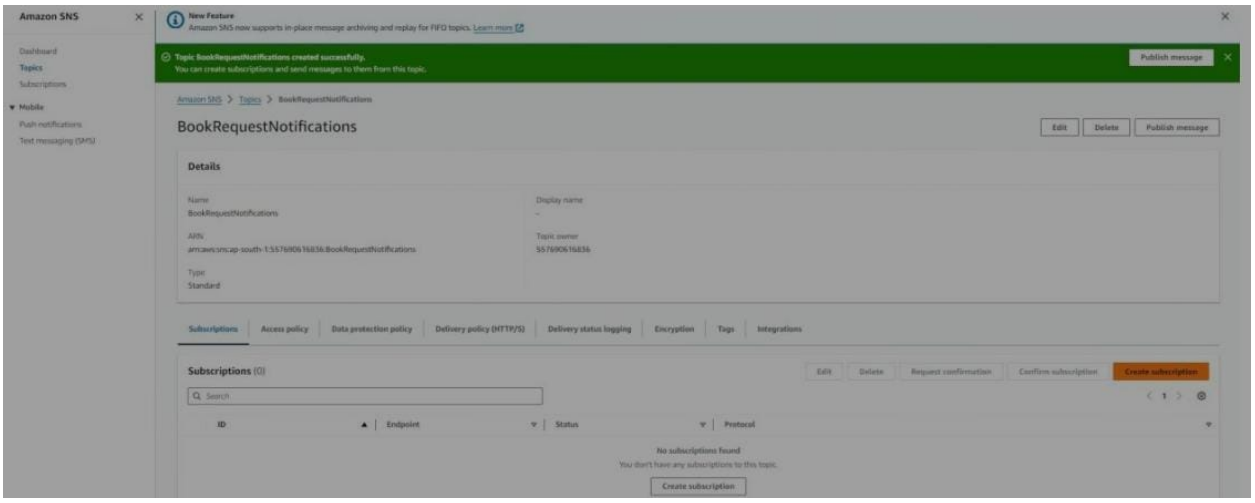


☐ Choose Standard type for general notification use cases and Click on Create Topic.



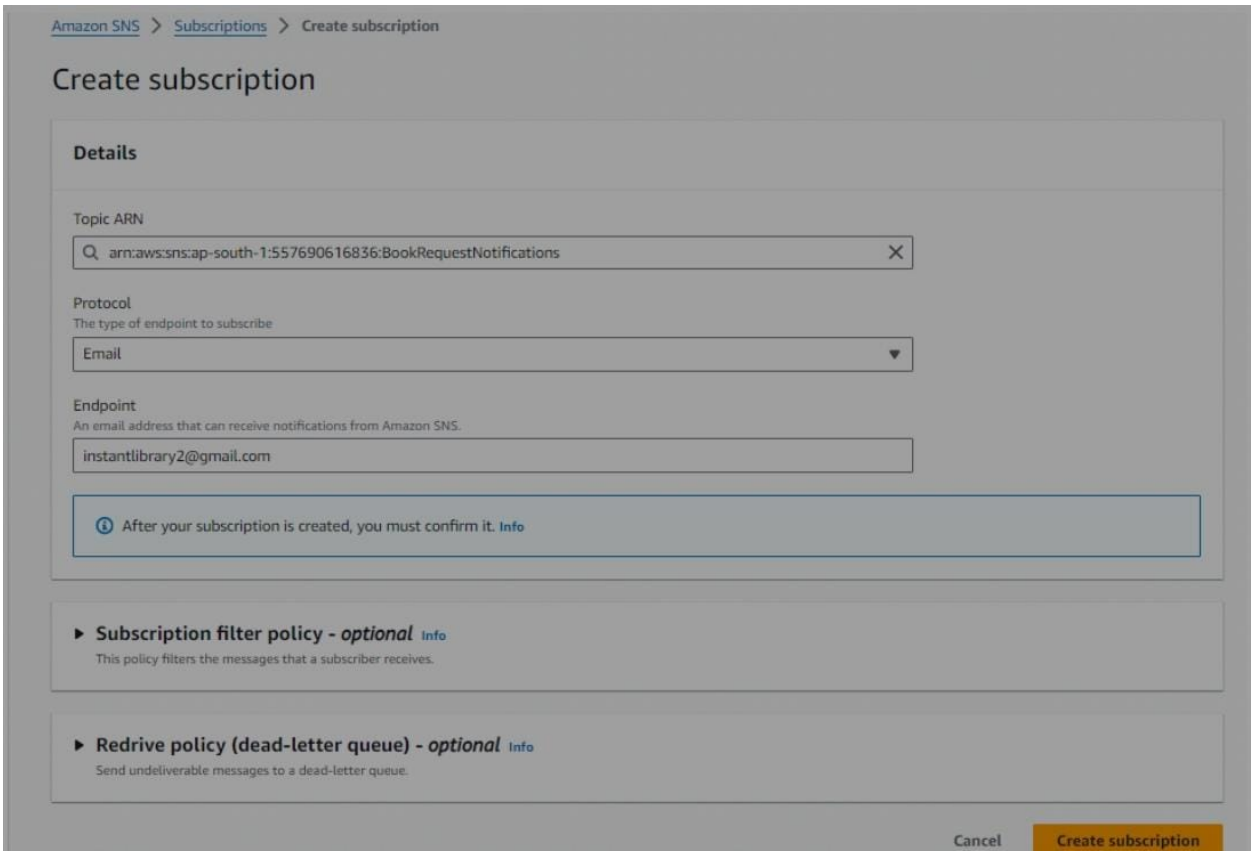


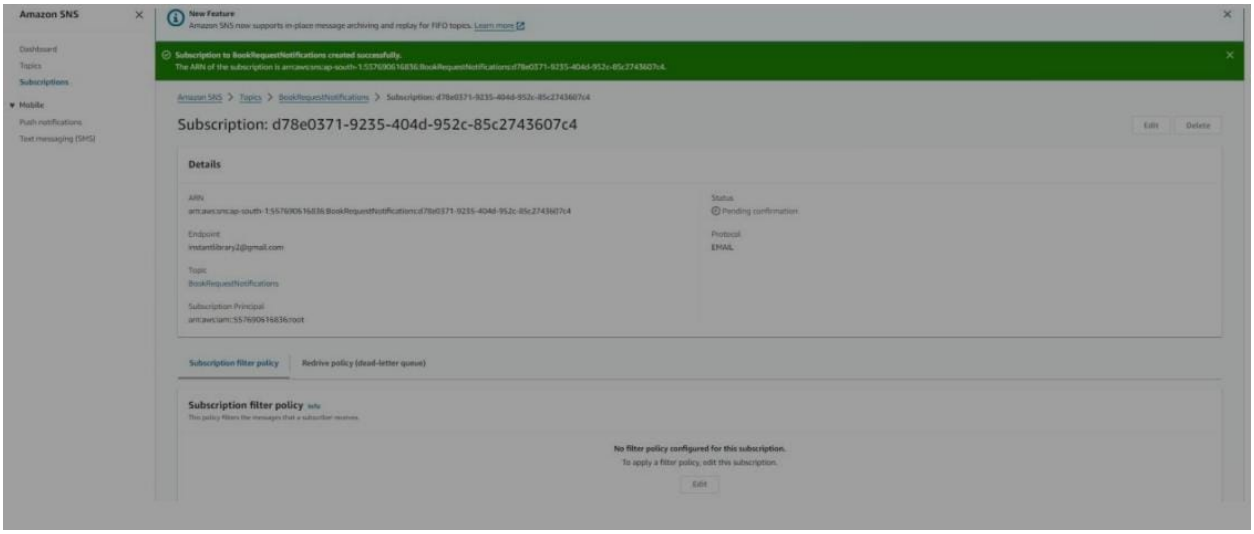
- Configure the SNS topic and note down the Topic ARN.



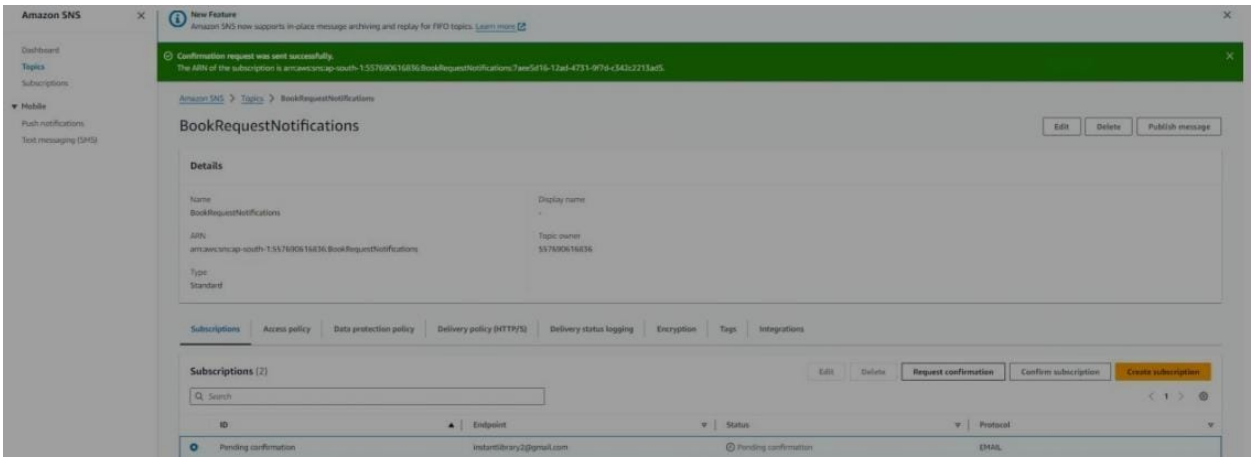
- **Activity 3.2: Subscribe users and customers to relevant SNS topics to receive real-time notifications when a snacks order is made.**

- Subscribe users (or admin) to this topic via Email. When a snacks reque is made, notifications will be sent to the subscribed emails.

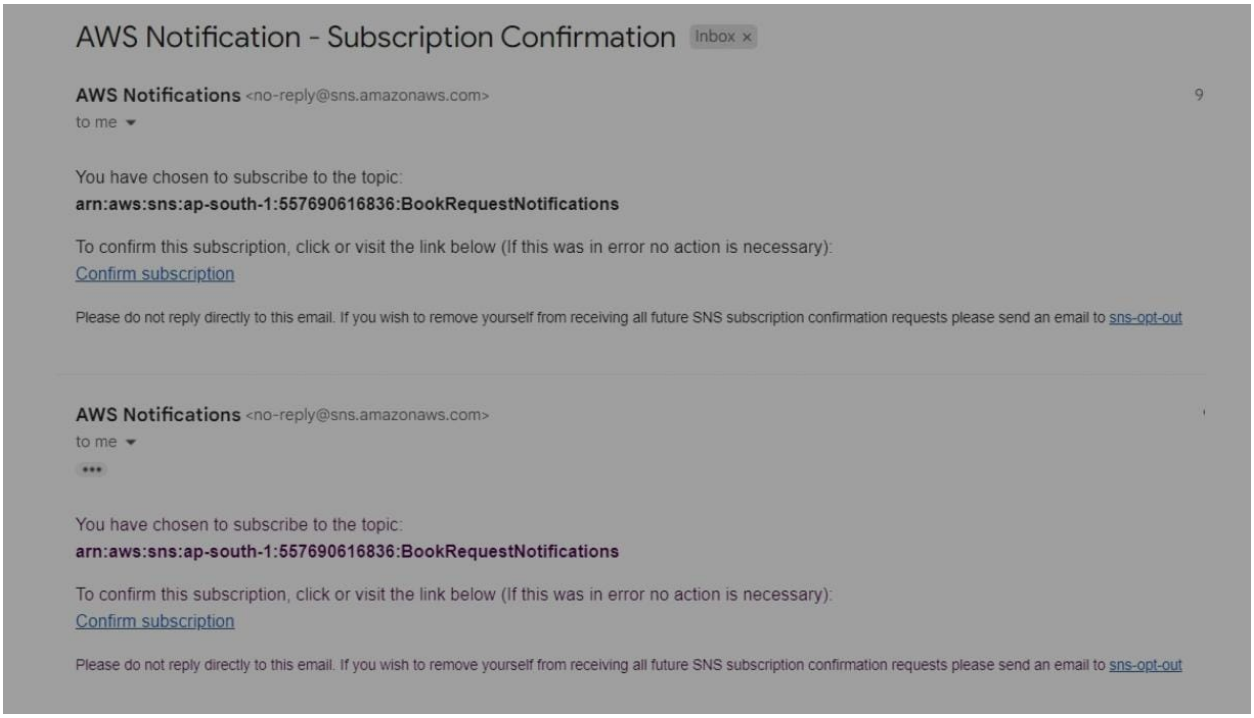


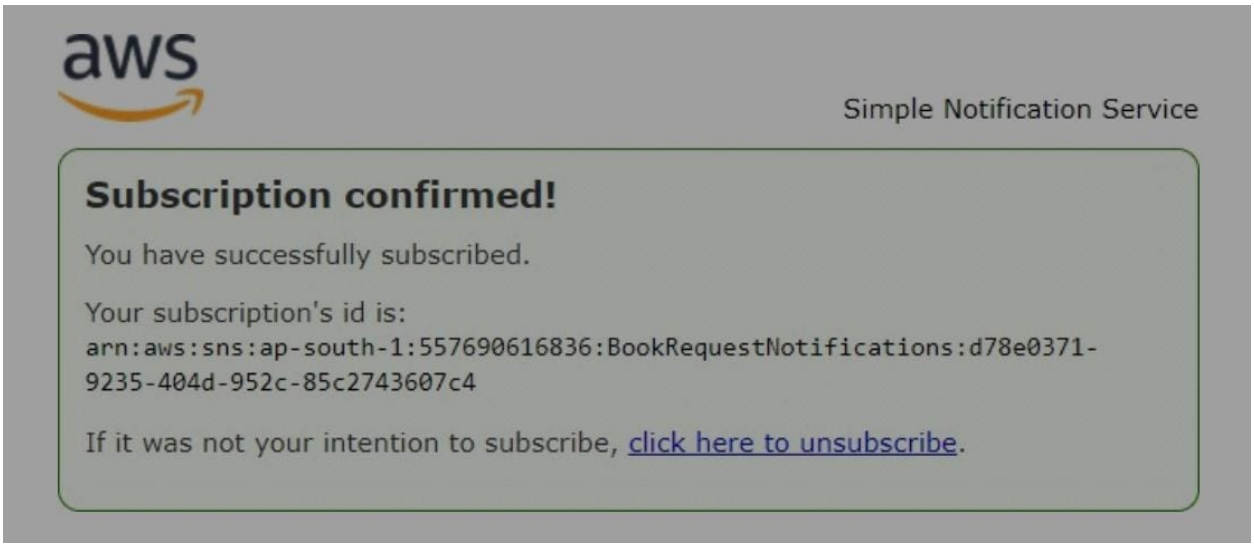


After subscription request for the mail confirmation

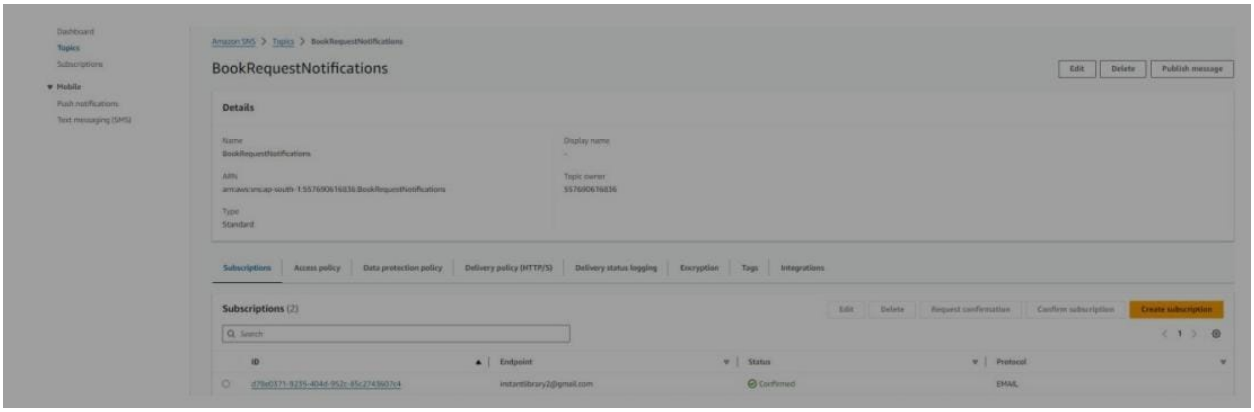


Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.





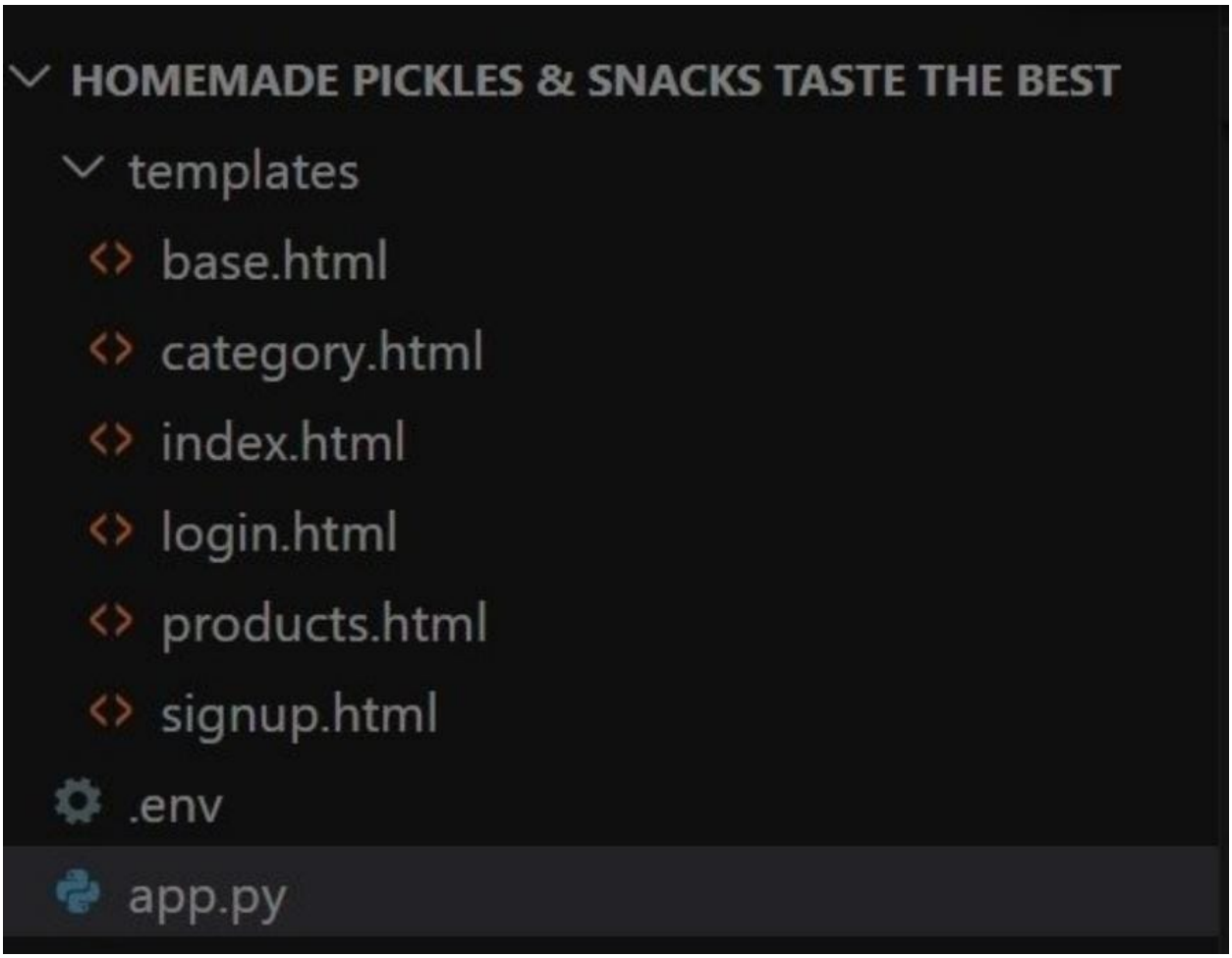
☐ Successfully done with the SNS mail subscription and setup, now store the ARN link.



Milestone 4:Backend Development and Application Setup

☐ **Activity 4.1:** Develop the backend using Flask

☐ **File Explorer Structure**



Description of the code :

- **Flask App Initialization**

```
app.py > ...
from flask import Flask, request, session, redirect, url_for, render_template, flash, jsonify
import boto3
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import uuid
import os
import logging
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from dotenv import load_dotenv
```

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

- **Dynamodb Setup:**

```
# AWS Resources
# ------------------------------------
dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
sns = boto3.client('sns', region_name=AWS_REGION)

products_table = dynamodb.Table(PRODUCTS_TABLE)
users_table = dynamodb.Table(USERS_TABLE)
orders_table = dynamodb.Table(ORDERS_TABLE)

# ------------------------------------
```

- **SNS Connection**

```
# SNS Topic ARN (create the SNS topic in AWS and provide the ARN here)
sns = boto3.client('sns', region_name='ap-south-1')
sns_topic_arn = 'arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications'

# Email settings (for sending emails)
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
SENDER_EMAIL = "instantlibrary2@gmail.com"
SENDER_PASSWORD = "luut dsih nyvq dgzv"  # Your app password



# Function to send email
def send_email(to_email, subject, body):
    msg = MIMEMultipart()
    msg['From'] = SENDER_EMAIL
    msg['To'] = to_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))

    try:
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        text = msg.as_string()
        server.sendmail(SENDER_EMAIL, to_email, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

- **Routes for Web Pages**
- **Home Route:**

```python
# Routes
# ------------------------------------
@app.route('/')
def home():
    try:
        response = products_table.scan(Limit=3)
        products = response.get('Items', [])
    except Exception as e:
        logger.error(f"Failed to load products: {e}")
        products = []
    return render_template('index.html', featured_products=products)
```

- **Signup Route:**

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

        try:
            existing = users_table.get_item(Key={'email': email})
            if 'Item' in existing:
                flash("Email already registered.", "danger")
                return redirect(url_for('signup'))

            users_table.put_item(Item={
                'email': email,
                'name': name,
                'password': password,
                'created_at': datetime.~~utcnow~~().isoformat()
            })

            session['email'] = email
            session['name'] = name

            subject = "Welcome to Our Pickle & Snacks Store!"
            body = f"Hello {name},\n\nThank you for signing up! Enjoy shopping your favorite pickles and snacks.\n\nTeam PickleMart"
            send_email(email, subject, body)

            flash("Account created!", "success")
            return redirect(url_for('home'))
        except Exception as e:
            logger.error(f"Signup error: {e}")
            flash("Error creating account.", "danger")

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

- **login Route (GET/POST):**

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

        try:
            result = users_table.get_item(Key={'email': email})
            user = result.get('Item')

            if user and check_password_hash(user['password'], password):
                session['email'] = email
                session['name'] = user['name']
                flash("Logged in successfully!", "success")
                return redirect(url_for('home'))
            else:
                flash("Invalid credentials.", "danger")
        except Exception as e:
            logger.error(f"Login error: {e}")
            flash("Login failed.", "danger")

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

- **logout route:**

```python
@app.route('/logout')
def logout():
    session.clear()
    flash("Logged out.", "info")
    return redirect(url_for('home'))
```

- **Products Routes:**

```python
@app.route('/products')
def products():
    category = request.args.get('category')
    try:
        if category:
            response = products_table.scan(
                FilterExpression="category = :c",
                ExpressionAttributeValues={":c": category}
            )
        else:
            response = products_table.scan()
        products = response.get('Items', [])
    except Exception as e:
        logger.error(f"Product load error: {e}")
        products = []
    return render_template('products.html', products=products)
```

Add to Cart Route:

```python
@app.route('/add-to-cart', methods=['POST'])
def add_to_cart():
    if not is_logged_in():
        return jsonify({'success': False, 'message': 'Login required'})

    email = session['email']
    product_id = request.json.get('product_id')
    quantity = int(request.json.get('quantity', 1))

    if email not in cart_db:
        cart_db[email] = {}

    cart_db[email][product_id] = cart_db[email].get(product_id, 0) + quantity
    return jsonify({'success': True, 'message': 'Added to cart'})
```

Remove from Cart :

```python
@app.route('/remove-from-cart', methods=['POST'])
def remove_from_cart():
    if not is_logged_in():
        return jsonify({'success': False, 'message': 'Login required'})

    email = session['email']
    product_id = request.json.get('product_id')

    try:
        if email in cart_db and product_id in cart_db[email]:
            del cart_db[email][product_id]
            return jsonify({'success': True, 'message': 'Item removed from cart'})
        else:
            return jsonify({'success': False, 'message': 'Item not found in cart'}
    except Exception as e:
        logger.error(f"Remove from cart error: {e}")
        return jsonify({'success': False, 'message': 'Error removing item'})
```

Update Cart:

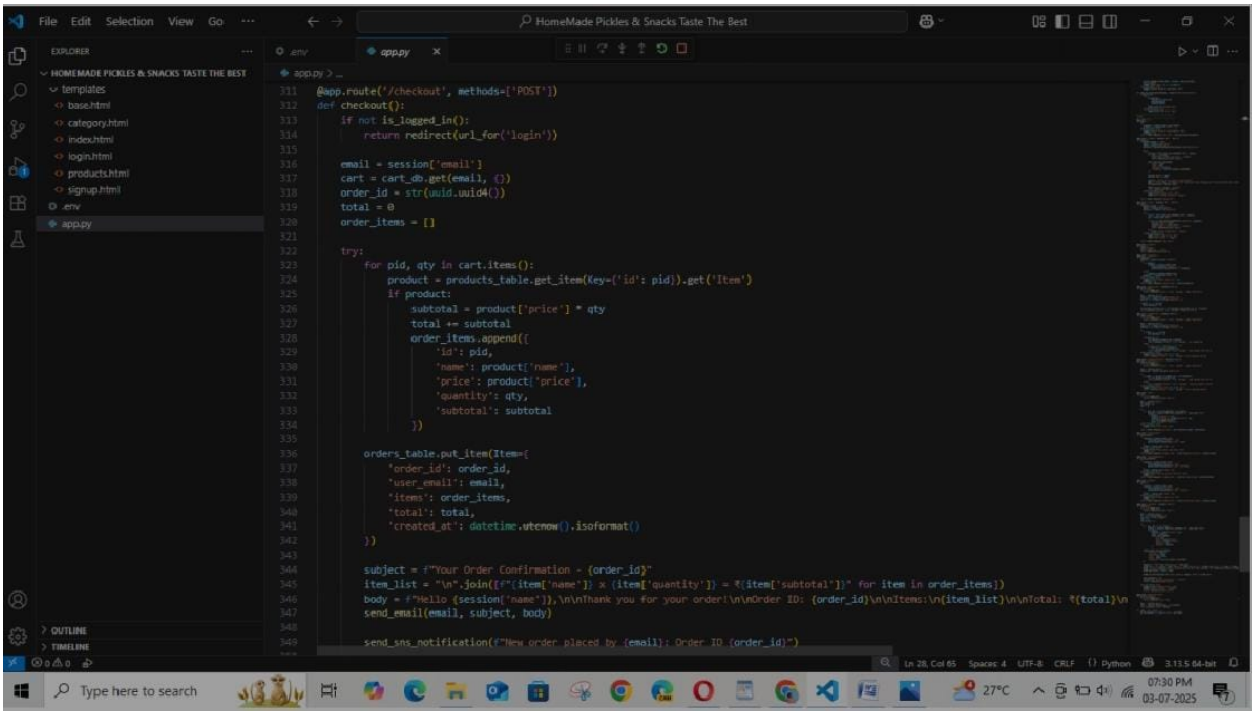
```python
@app.route('/update-cart', methods=['POST'])
def update_cart():
    if not is_logged_in():
        return jsonify({'success': False, 'message': 'Login required'})

    email = session['email']
    product_id = request.json.get('product_id')
    quantity = int(request.json.get('quantity', 1))

    try:
        if email not in cart_db:
            cart_db[email] = {}

        if quantity > 0:
            cart_db[email][product_id] = quantity
            return jsonify({'success': True, 'message': 'Cart updated'})
        else:
            # Remove item if quantity is 0 or less
            if product_id in cart_db[email]:
                del cart_db[email][product_id]
            return jsonify({'success': True, 'message': 'Item removed from cart'})
    except Exception as e:
        logger.error(f"Update cart error: {e}")
        return jsonify({'success': False, 'message': 'Error updating cart'})
```

Checkout route:



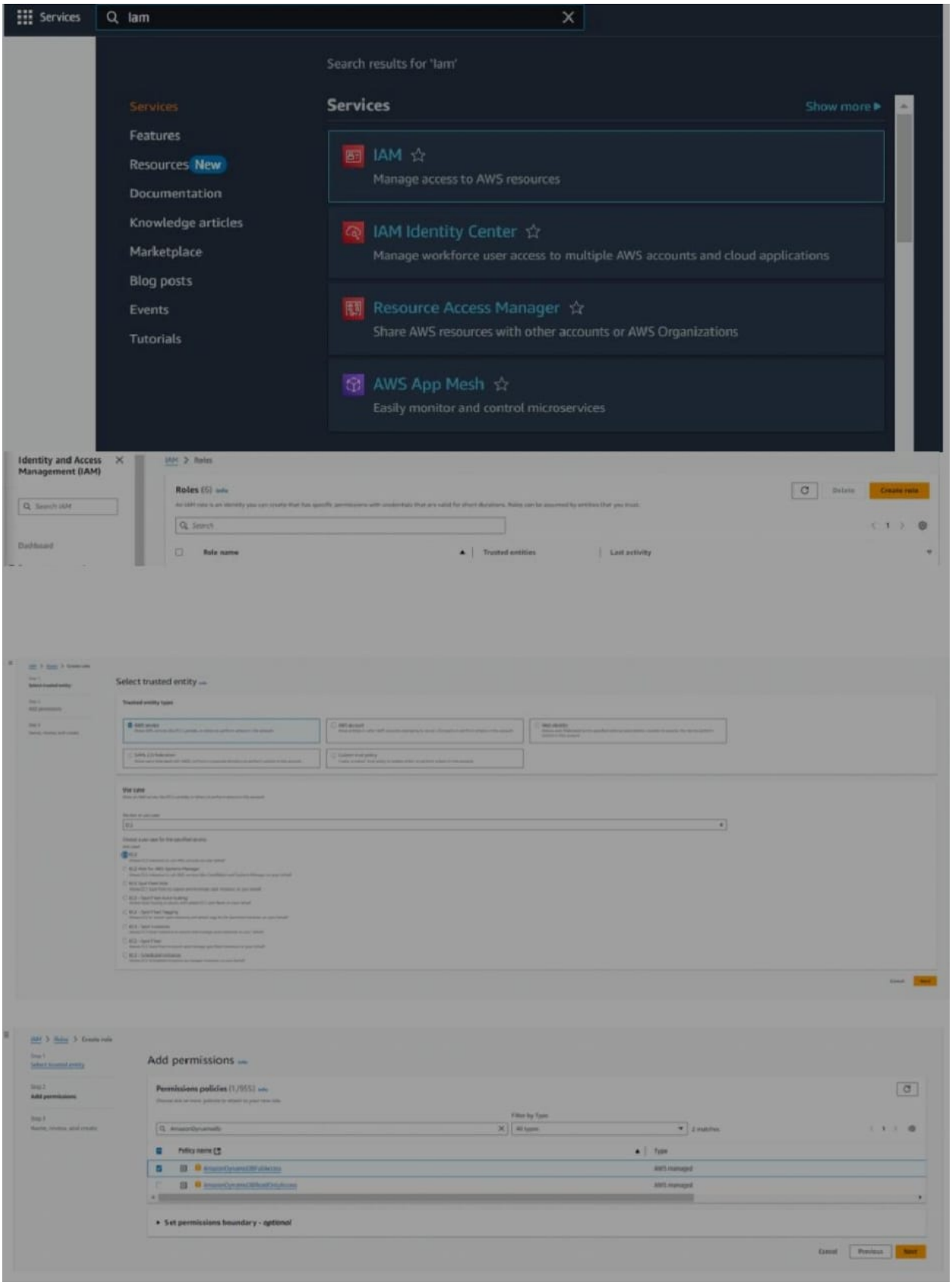
Deployment Code:

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

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.

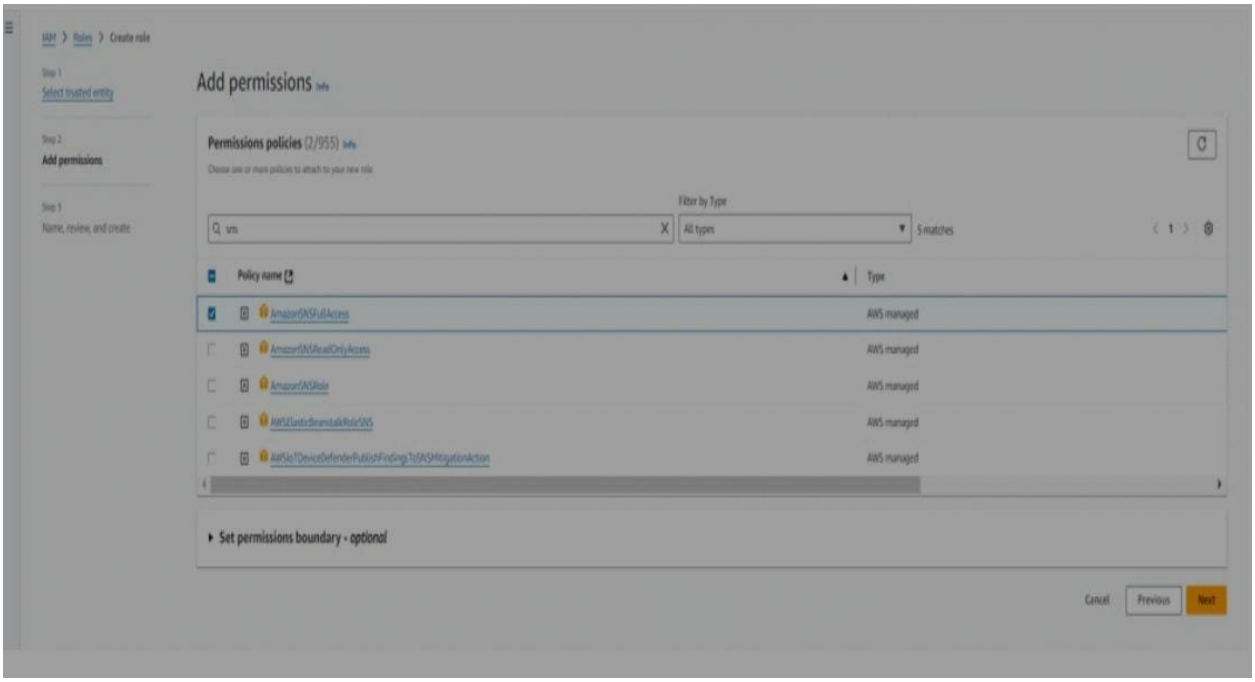

Services
Iam
Search results for 'Iam'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
Resource Access Manager
Share AWS resources with other accounts or AWS Organizations
AWS App Mesh
Easily monitor and control microservices
Identity and Access Management (IAM)
Roles
Delete
Create role
Role name
Trusted entities
Last activity
Select trusted entity
Add permissions
Permissions policies
Set permissions boundary - optional
Cancel
Previous
Next

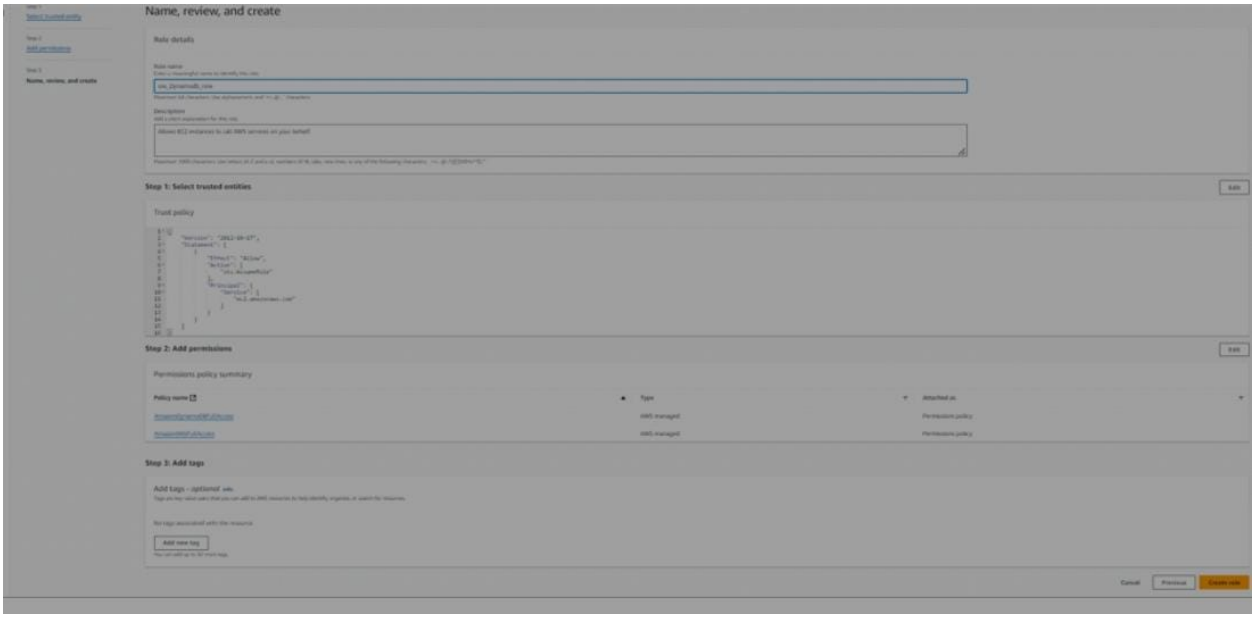
Activity 5.2: Attach Policies.

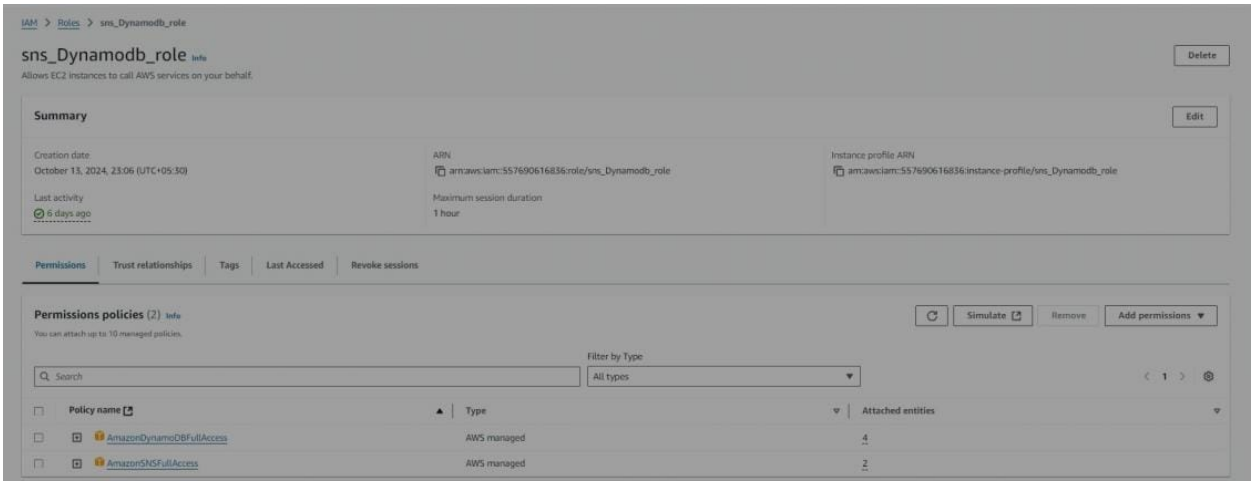
Attach the following policies to the role:

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

AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

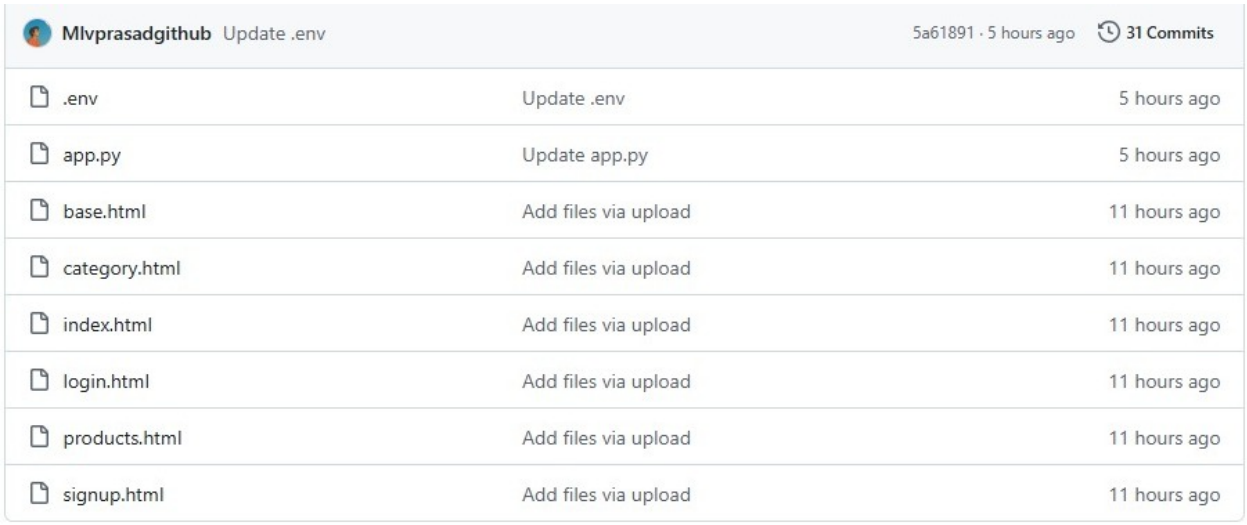


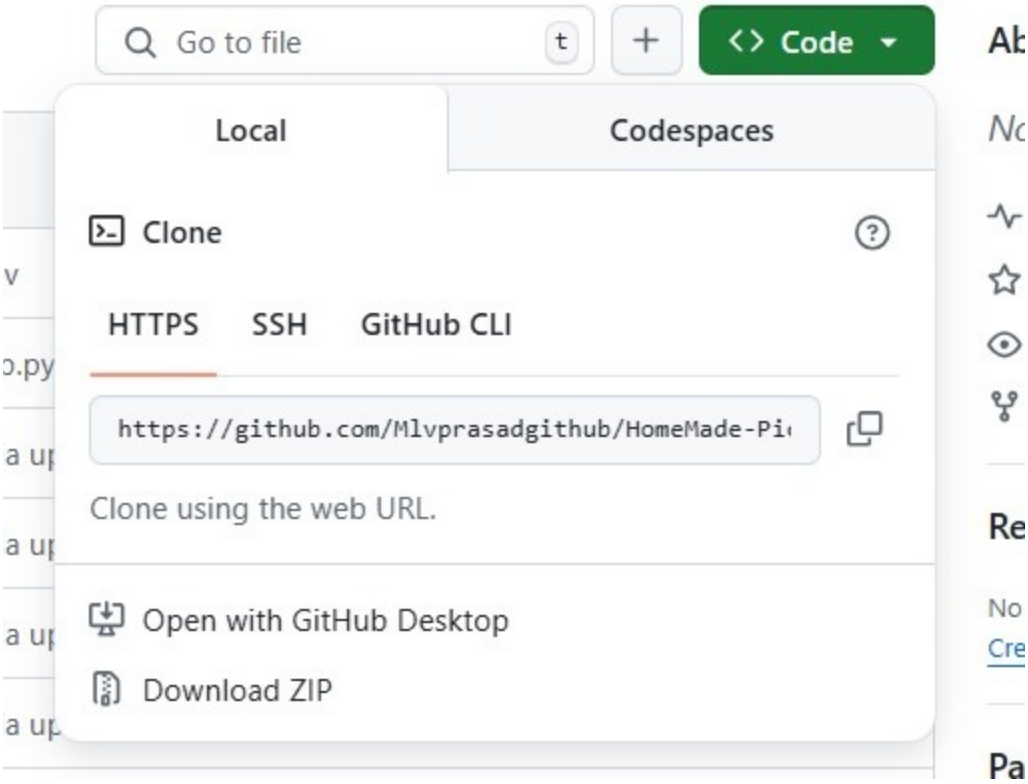




Milestone 6: EC2 Instance Setup

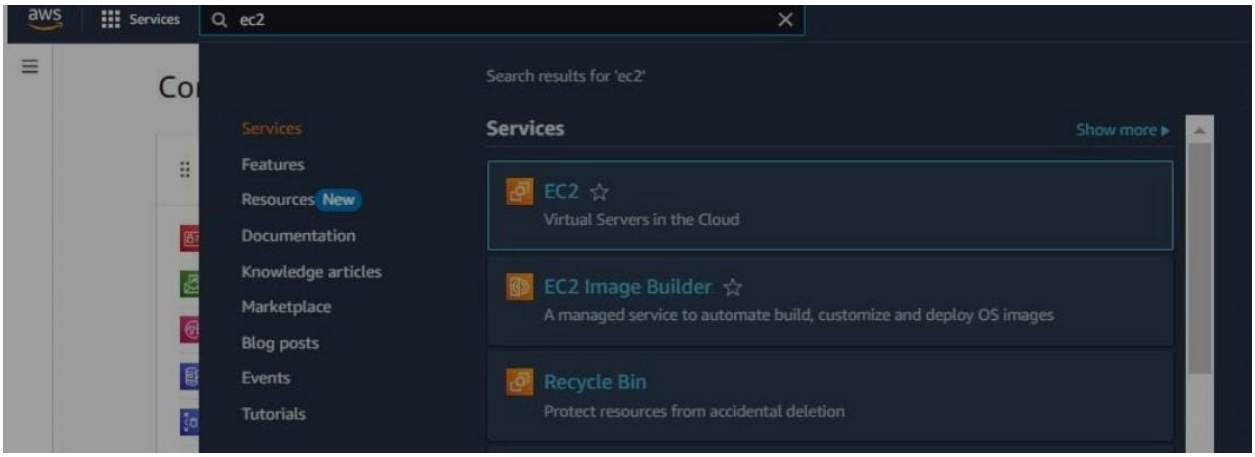
Note: Load your Flask app and Html files into GitHub repository.



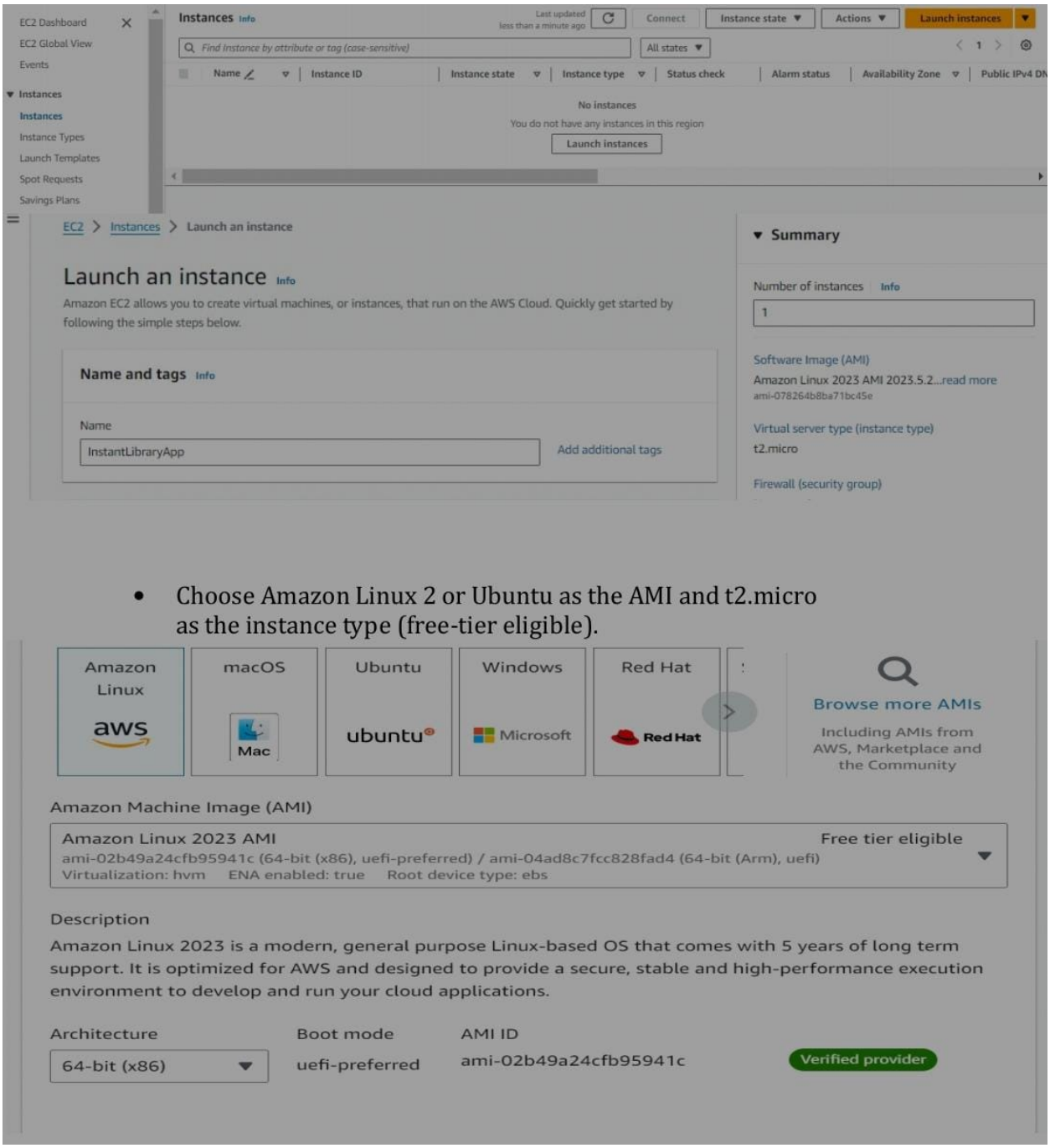


- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

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

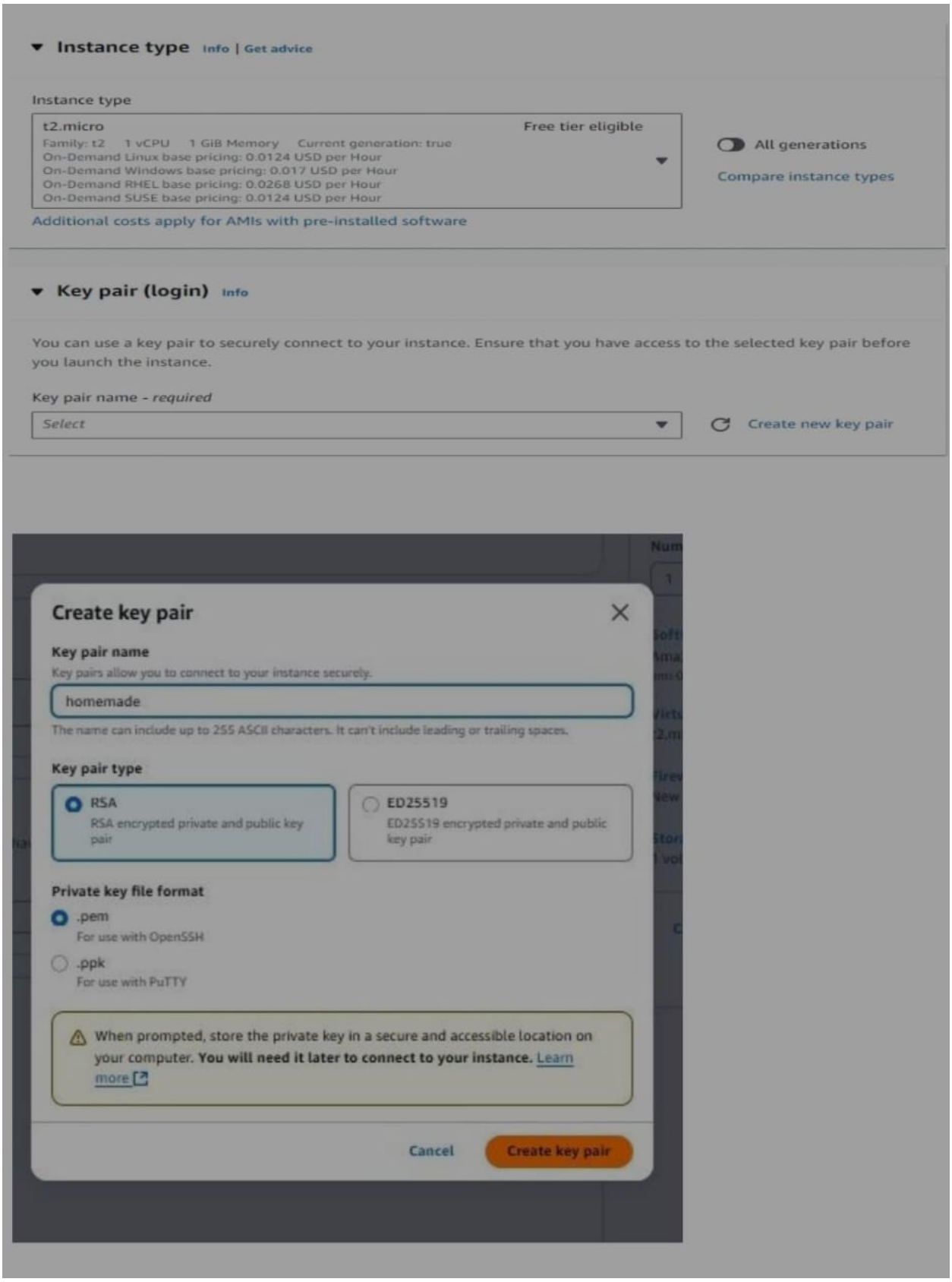


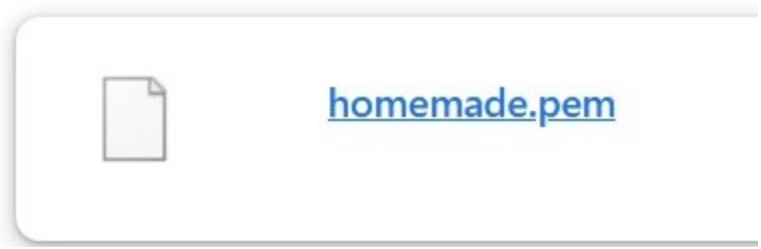
- Click on Launch instance to launch EC2 instance

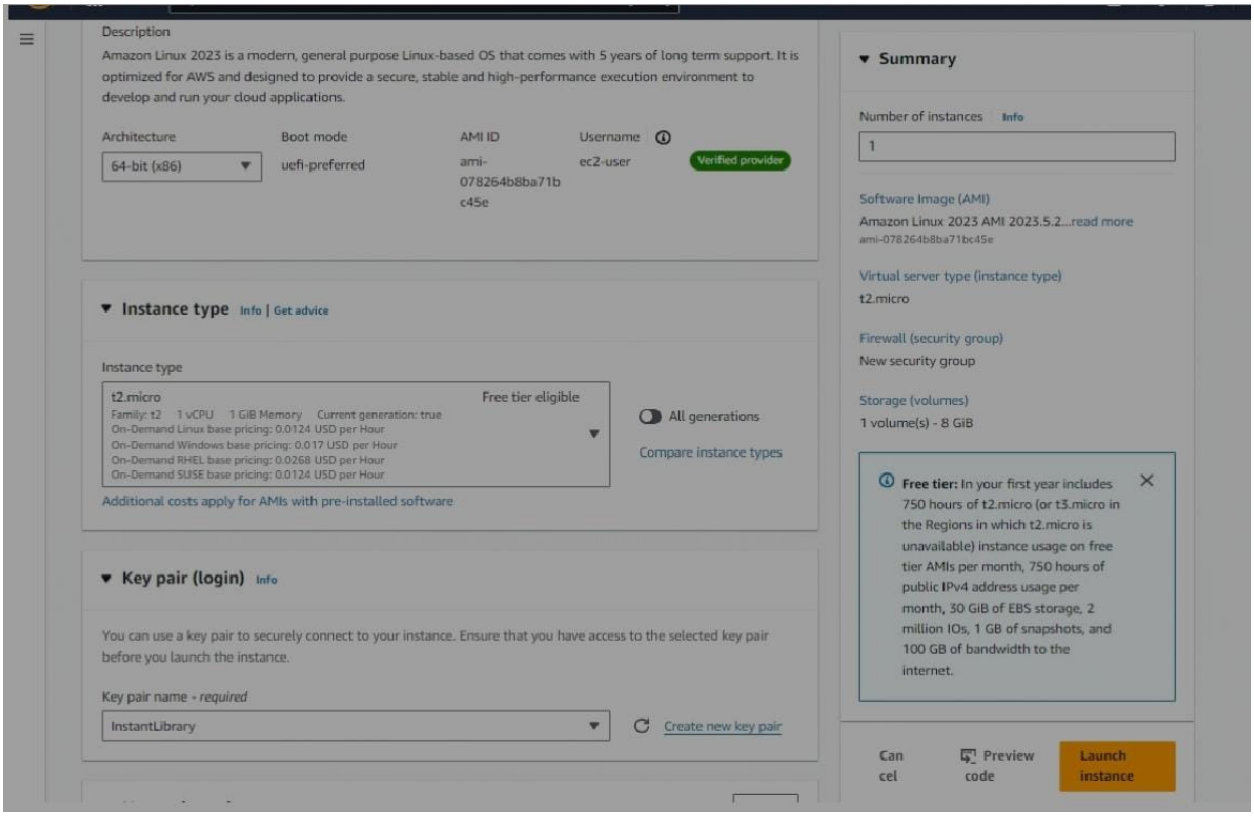


- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

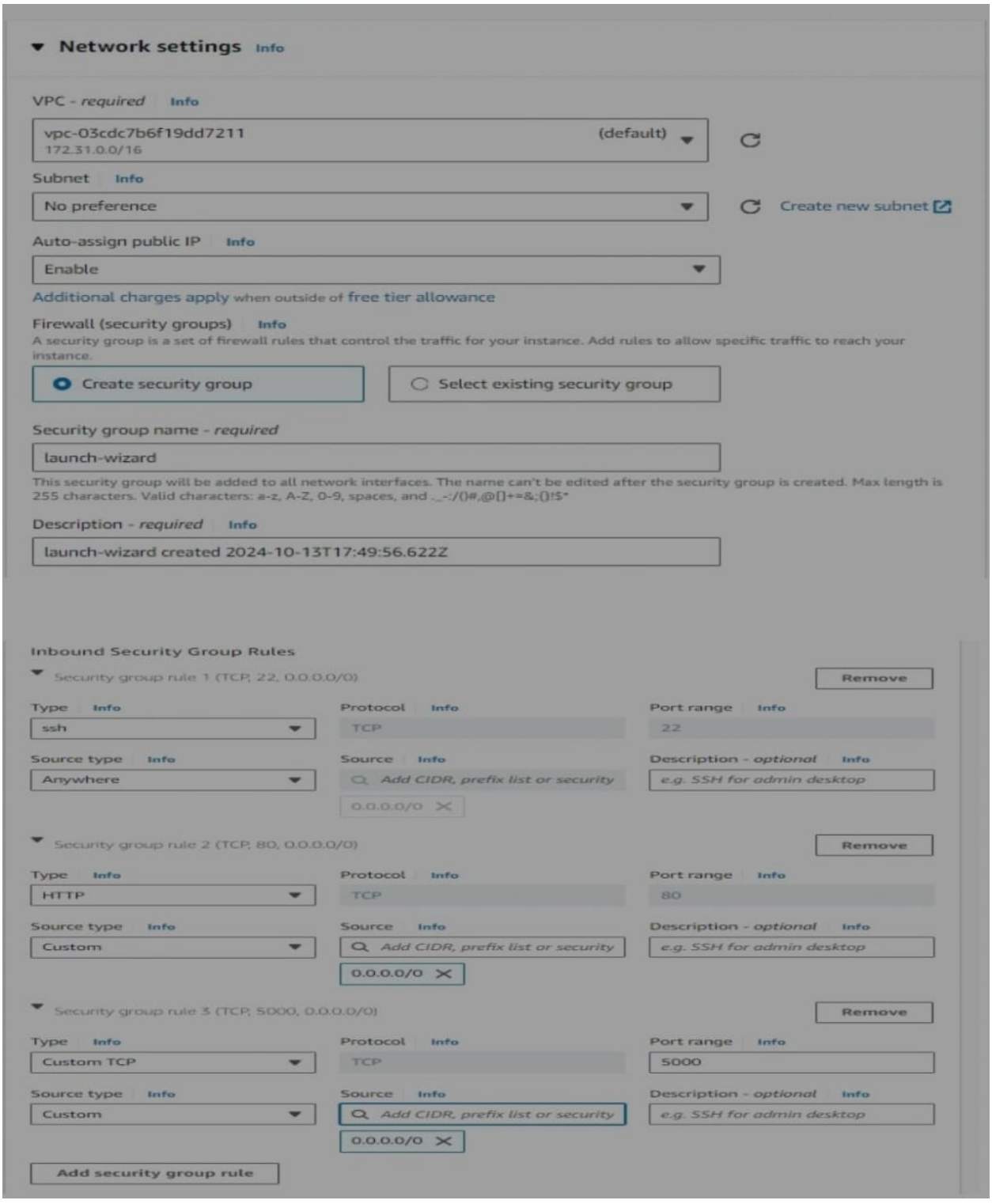
☐ Create and download the key pair for Server access.

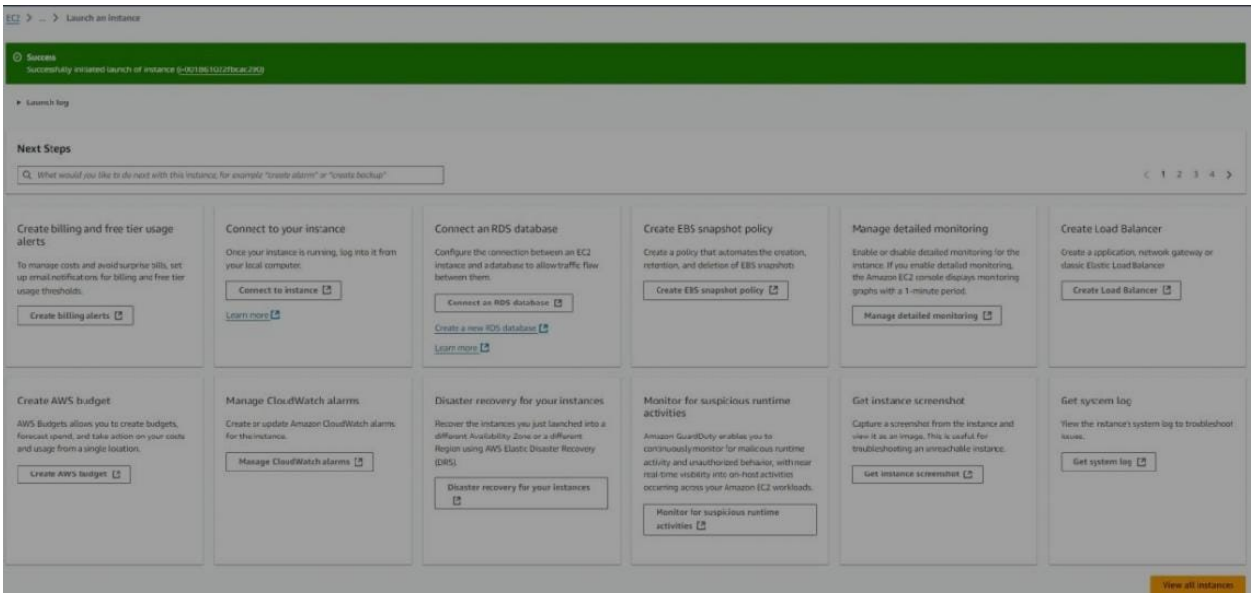

Instance type Info | Get advice
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
Select
Create new key pair
Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
homemade
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key pair
Private key file format
.pem
For use with OpenSSH
.ppk
For use with PuTTY
When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. Learn more
Cancel
Create key pair



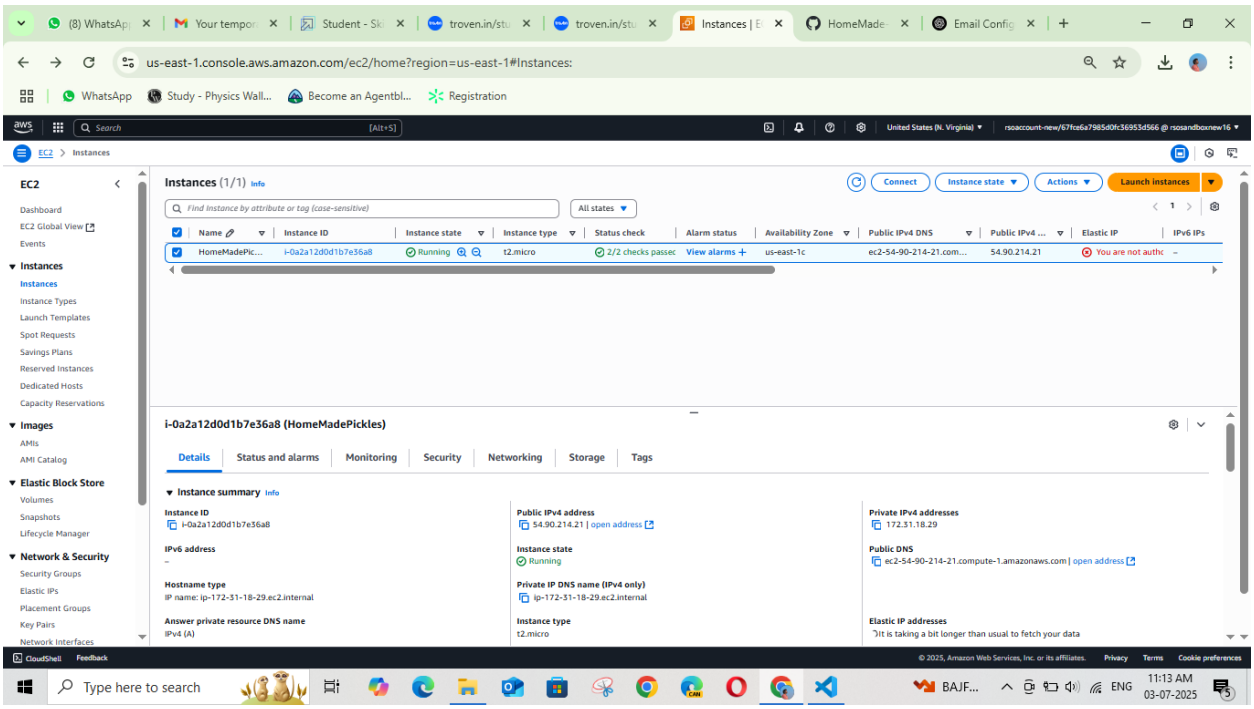


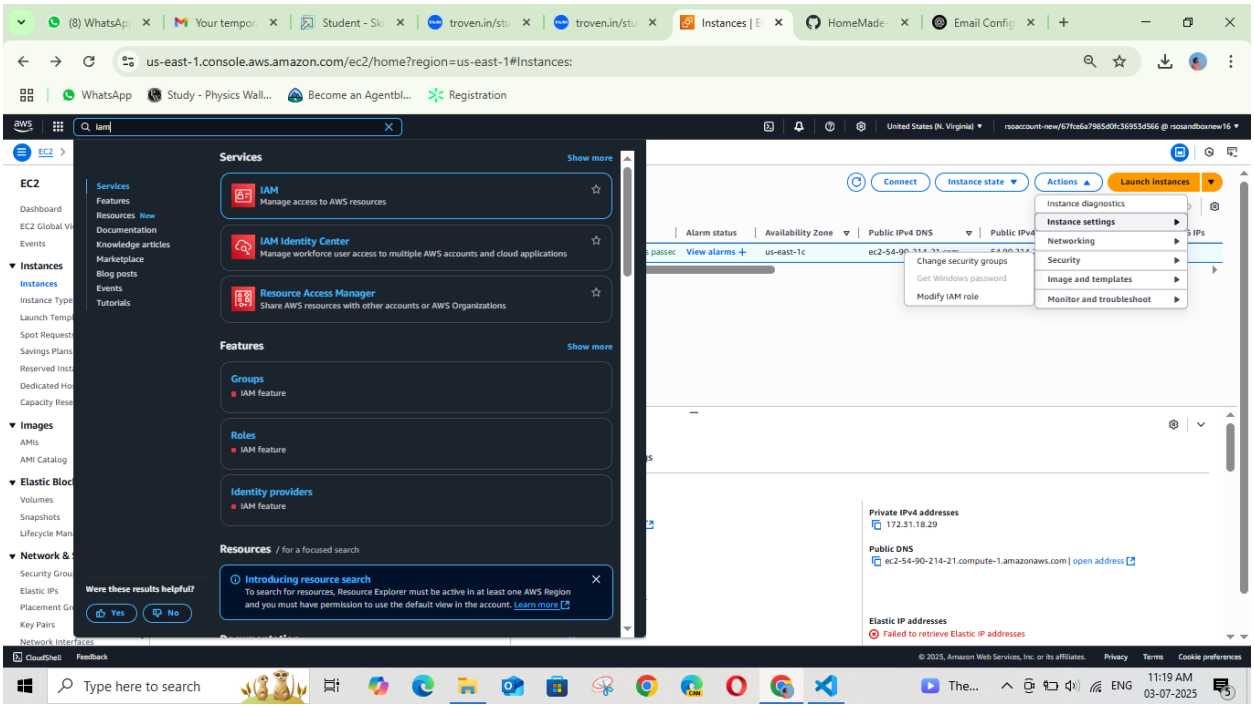
Activity 6.2:Configure security groups for HTTP, and SSH access.

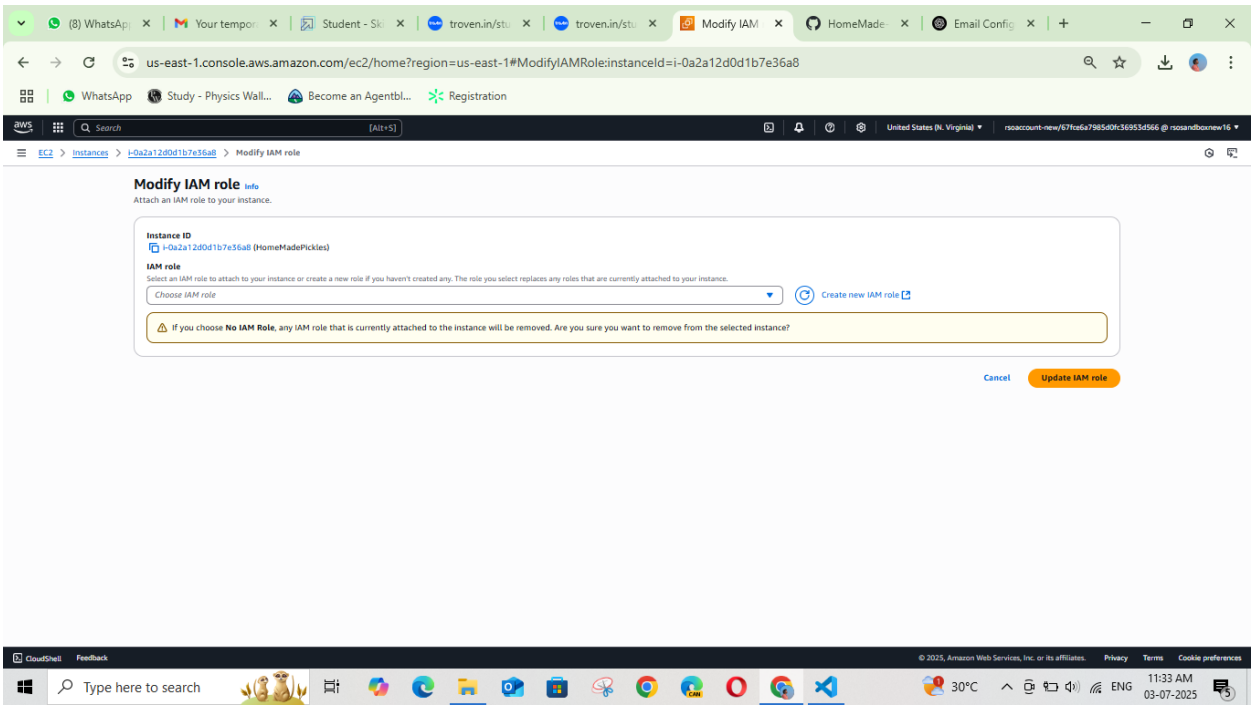

Network settings Info
VPC - required Info
vpc-03cdc7b6f19dd7211 (default)
172.31.0.0/16
Subnet Info
No preference
Create new subnet
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
Security group name - required
launch-wizard
This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&;{}!$*
Description - required Info
launch-wizard created 2024-10-13T17:49:56.622Z
Inbound Security Group Rules
Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 2 (TCP, 80, 0.0.0.0/0)
Remove
Type Info
HTTP
Protocol Info
TCP
Port range Info
80
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 3 (TCP, 5000, 0.0.0.0/0)
Remove
Type Info
Custom TCP
Protocol Info
TCP
Port range Info
5000
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Add security group rule



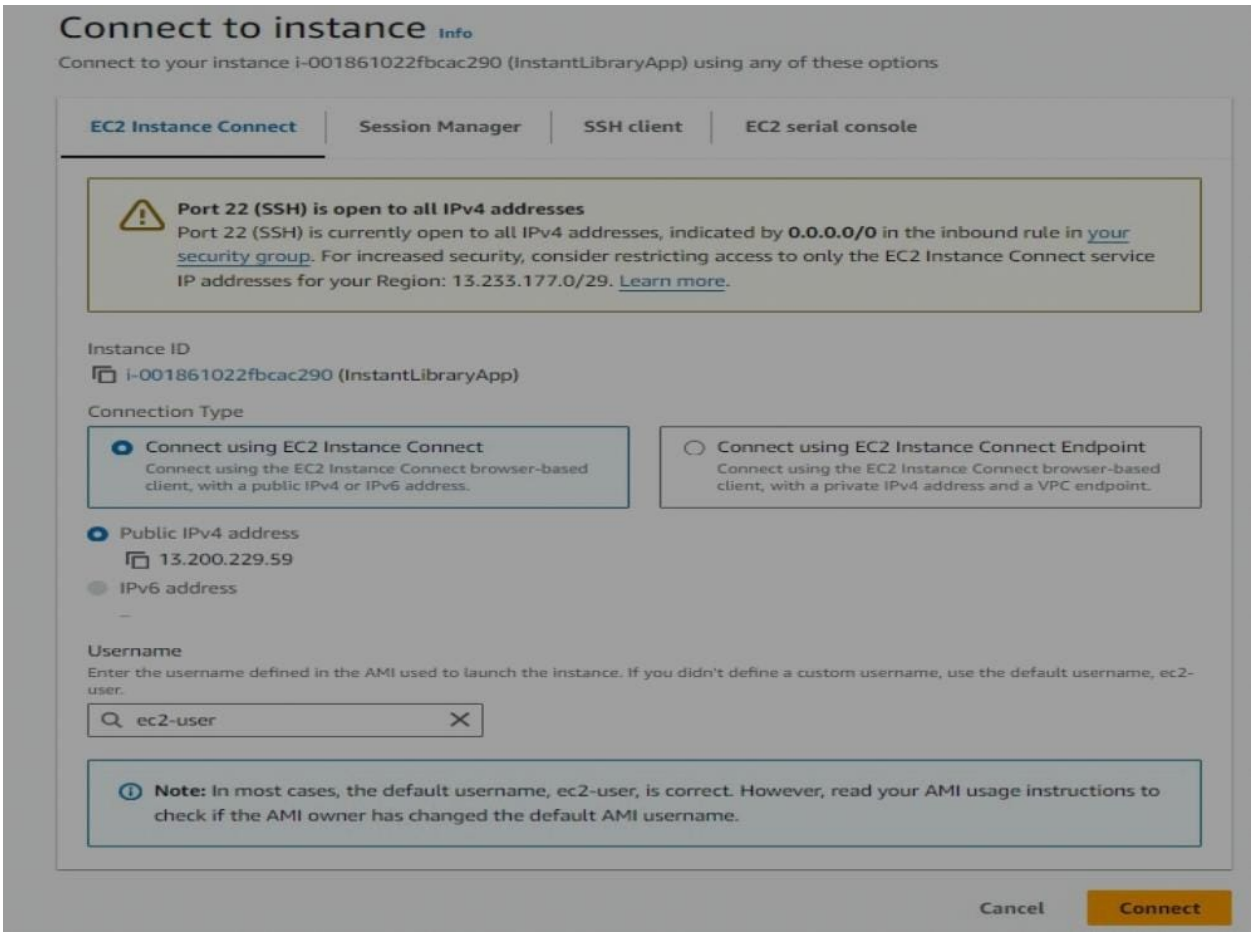
To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role 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.

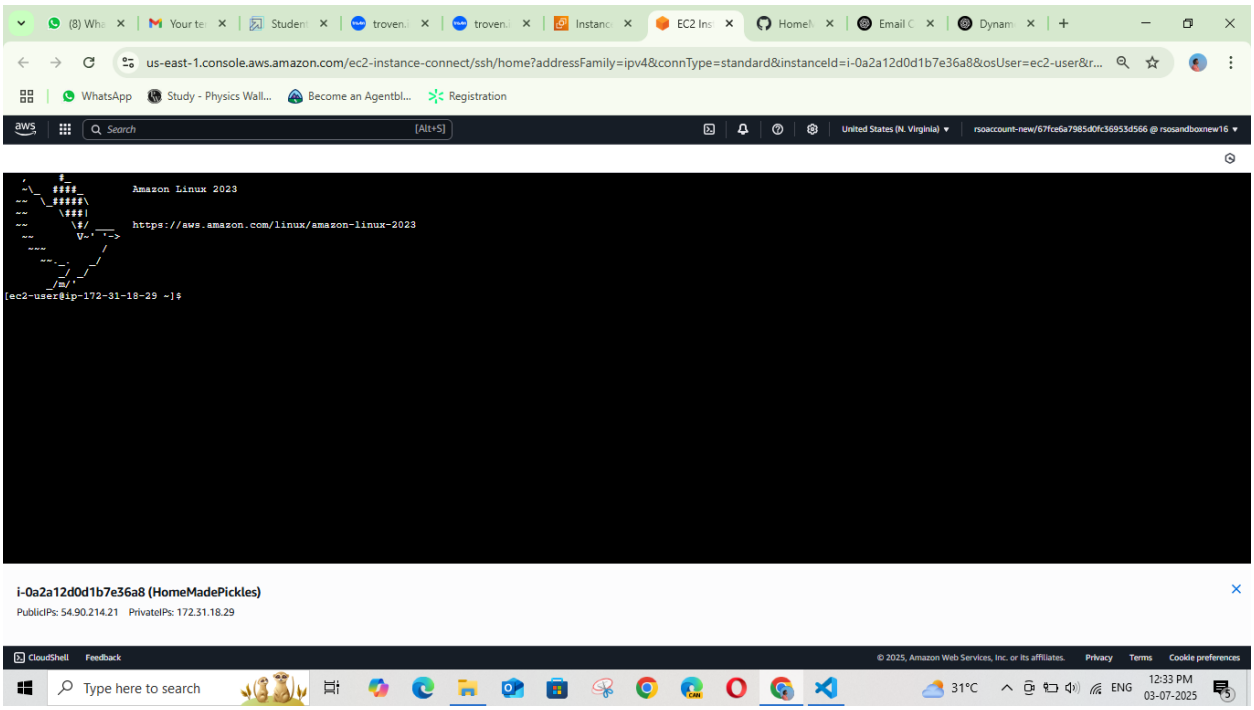

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:
Instances (1/1)
Connect
Instance state
Actions
Launch instances
HomeMadePic...
i-0a2a12d0d1b7e36a8
Running
t2.micro
2/2 checks passed
View alarms
us-east-1c
ec2-54-90-214-21.com...
54.90.214.21
i-0a2a12d0d1b7e36a8 (HomeMadePickles)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary
Instance ID
i-0a2a12d0d1b7e36a8
Public IPv4 address
54.90.214.21 | open address
Private IPv4 addresses
172.31.18.29
IPv6 address
Instance state
Running
Public DNS
ec2-54-90-214-21.compute-1.amazonaws.com | open address
Hostname type
IP name: ip-172-31-18-29.ec2.internal
Private IP DNS name (IPv4 only)
ip-172-31-18-29.ec2.internal
Answer private resource DNS name
IPv4 (A)
Instance type
t2.micro
Elastic IP addresses
It is taking a bit longer than usual to fetch your data


iam
Services
Show more
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
Resource Access Manager
Share AWS resources with other accounts or AWS Organizations
Features
Groups
IAM feature
Roles
IAM feature
Identity providers
IAM feature
Resources / for a focused search
Introducing resource search
To search for resources, Resource Explorer must be active in at least one AWS Region and you must have permission to use the default view in the account. Learn more
Were these results helpful?
Yes
No
Actions
Launch instances
Instance diagnostics
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
Private IPv4 addresses
172.31.18.29
Public DNS
ec2-54-90-214-21.compute-1.amazonaws.com | open address
Elastic IP addresses
Failed to retrieve Elastic IP addresses



Now connect the EC2 with the files


Connect to instance Info
Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Port 22 (SSH) is open to all IPv4 addresses
Port 22 (SSH) is currently open to all IPv4 addresses, indicated by 0.0.0.0/0 in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.233.177.0/29. Learn more.
Instance ID
i-001861022fbcac290 (InstantLibraryApp)
Connection Type
Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.
Public IPv4 address
13.200.229.59
IPv6 address
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.
ec2-user
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.
Cancel
Connect


us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0a2a12d0d1b7e36a8&osUser=ec2-user&r...
WhatsApp
Study - Physics Wall...
Become an Agentbl...
Registration
Search
[Alt+S]
United States (N. Virginia)
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-18-29 ~]$
i-0a2a12d0d1b7e36a8 (HomeMadePickles)
PublicIPs: 54.90.214.21 PrivateIPs: 172.31.18.29
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
31°C
ENG
12:33 PM
03-07-2025

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.

Note: change your-github-username and your-repository-name with your credentials here: 'git clone https://github.com/Mlvprasadgithub/HomeMade-Pickles-Snacks-Taste-The-Best.git '

- This will download your project to the EC2 instance.

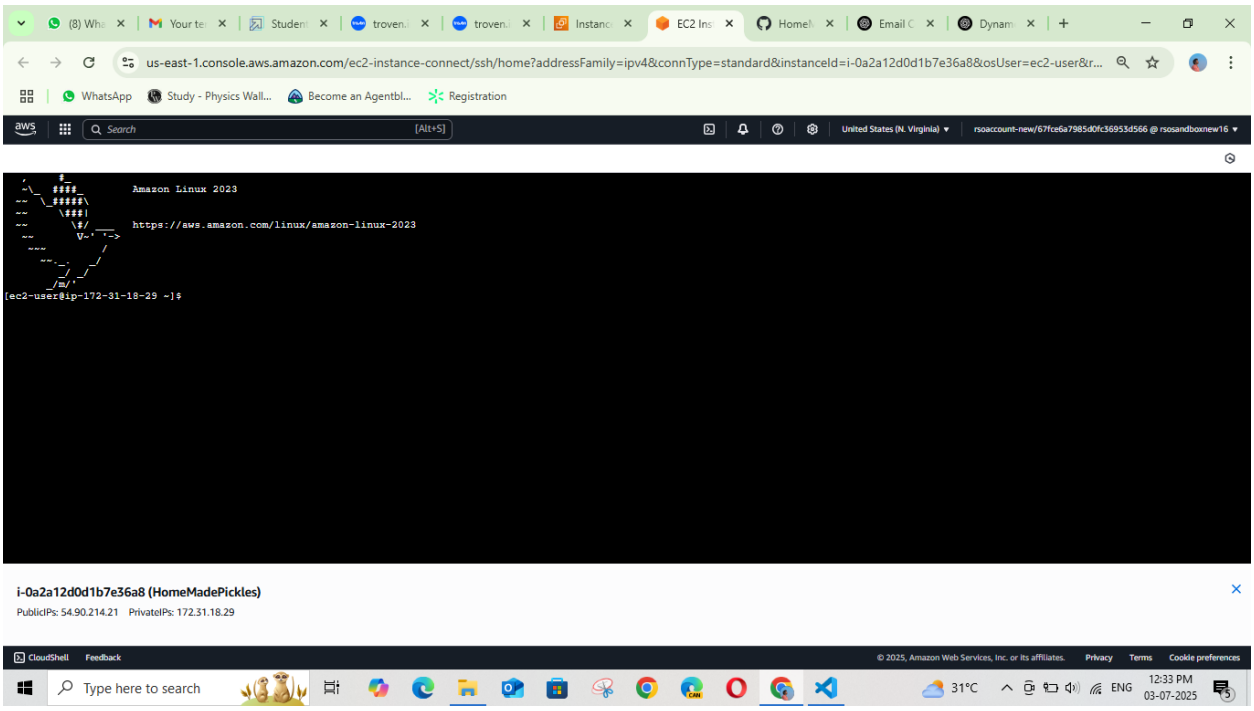
To navigate to the project directory, run the following command:

cd HomeMade Pickles & Snacks Taste the Best

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

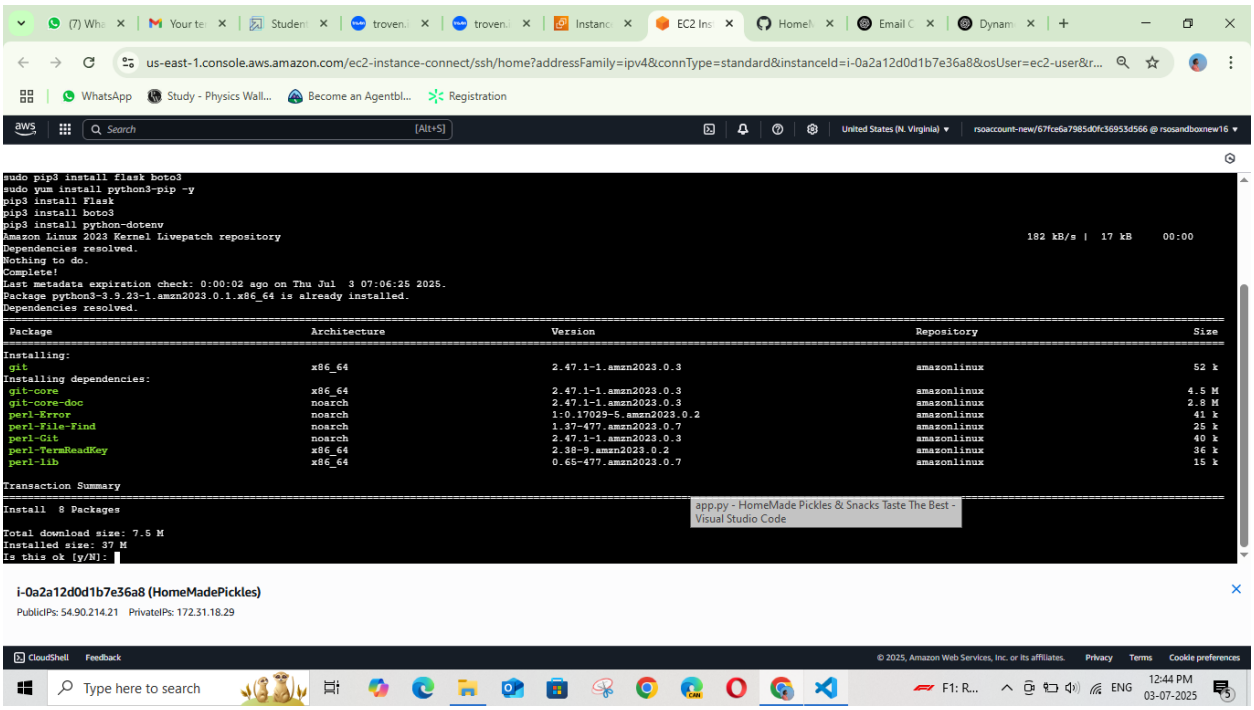
Run the Flask Application

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



Verify the Flask app is running: http://your-ec2-public-ip

☐ Run the Flask app on the EC2 instance

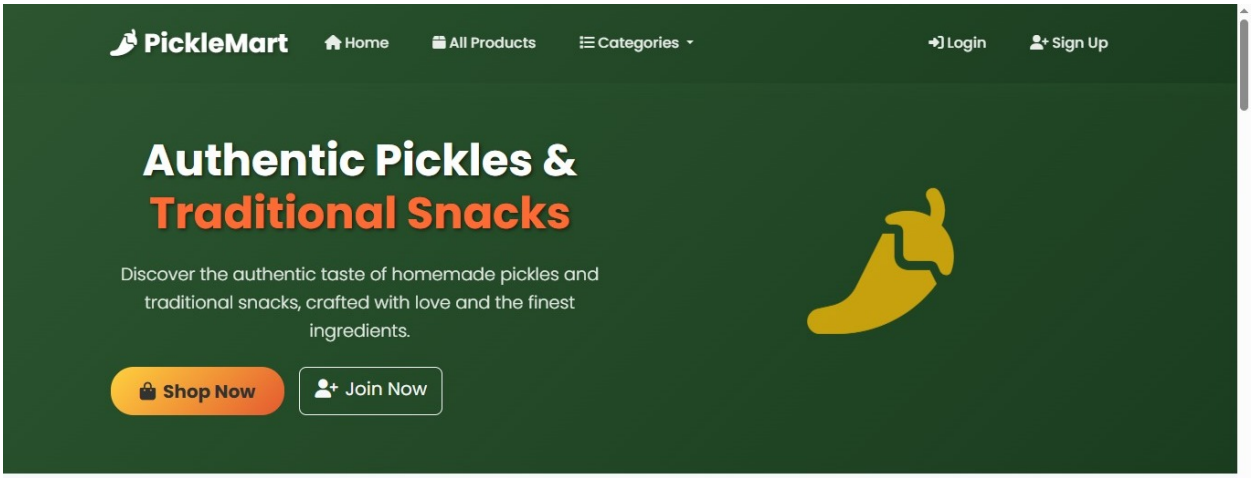


Access the website through:
PublicIPs: http://127.0.0.1:5000/

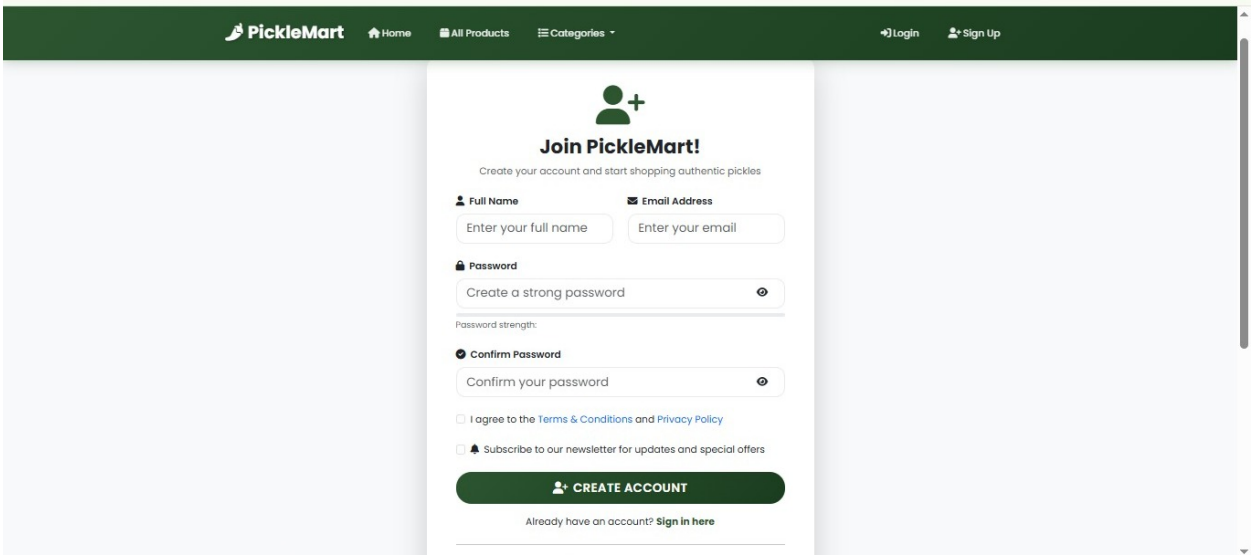
Milestone 8: Testing and Deployment

☐ Activity 8.1: Conduct functional testing to verify customer registration, login, pickles and snacks requests, and notifications.

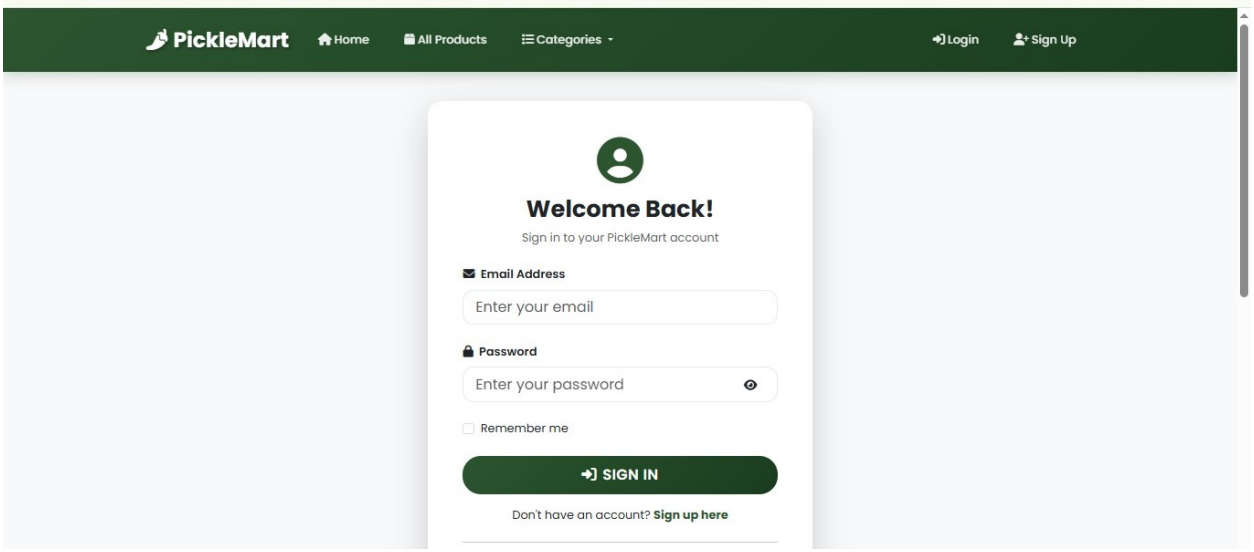
Home Page:



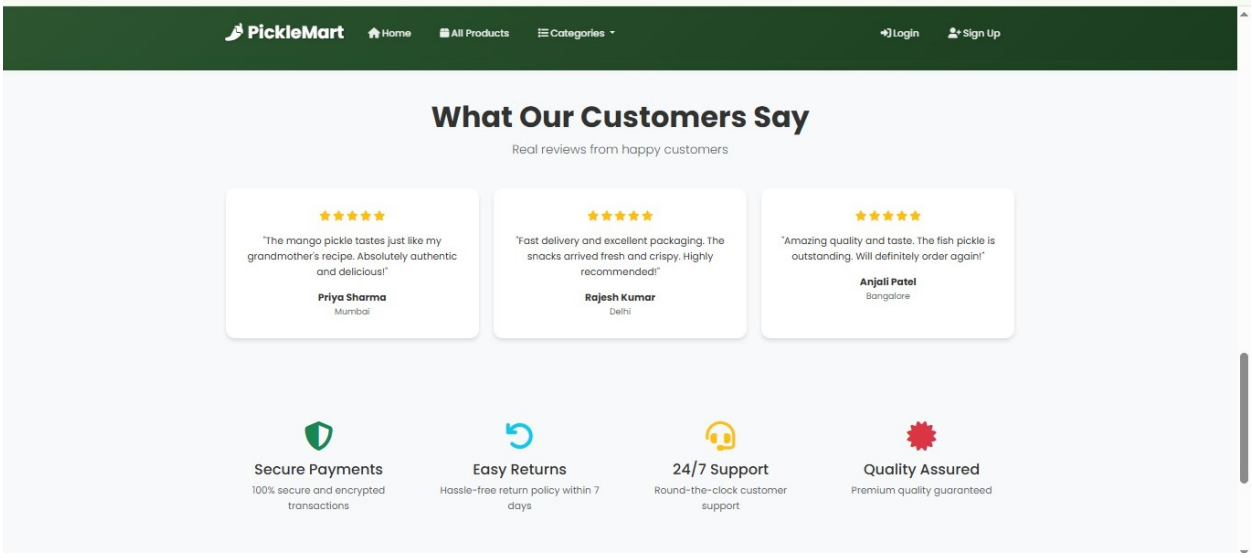
Register Page:



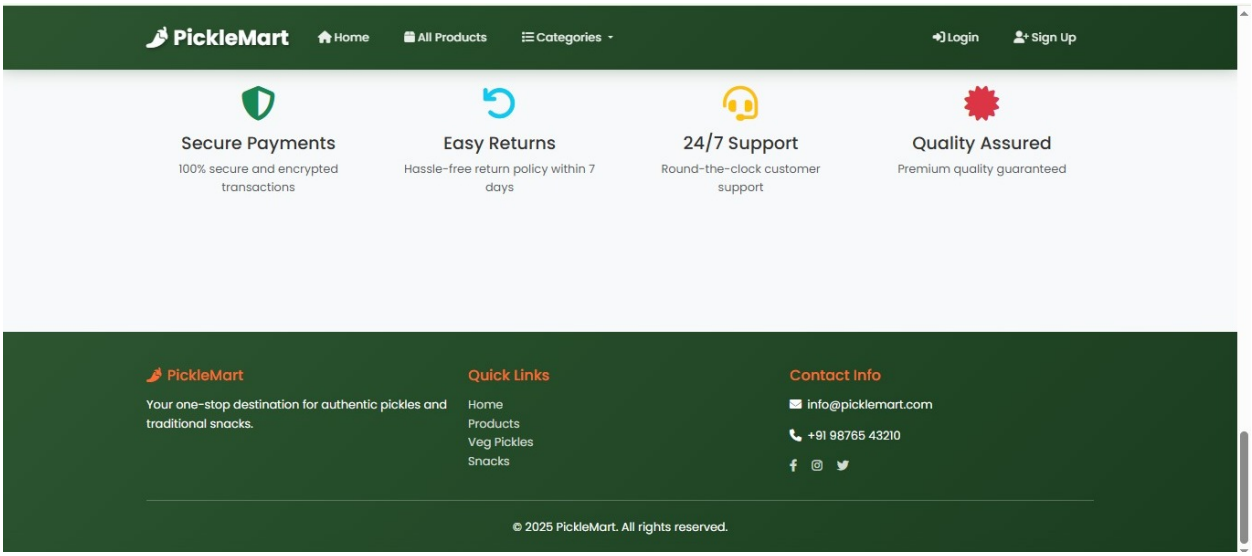
Login Page:



About Us Page:



Contact Us Page:



All Products Page:

All Products

Explore our complete collection of authentic pickles and traditional snacks

Search products... | All Categories | Sort by Name

Showing 10 products

Traditional Mango Pickle

Authentic homemade mango pickle with traditional spices and mustard oil...

4.8 (156)

~~₹150.00~~ ₹120.00 — In Stock

Add to Cart

Spicy Chicken Pickle

Tender chicken pieces marinated in aromatic spices and traditional recipe...

4.5 (89)

~~₹320.00~~ ₹250.00 — 8 left

Add to Cart

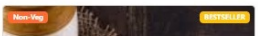
Mixed Vegetable Pickle

Colorful mix of seasonal vegetables pickled with authentic spices...

4.2 (67)

₹95.00 — In Stock

Add to Cart



Crispy Samosas (6 pcs)

Golden, crispy samosas filled with spiced potatoes and peas...

4.7 (134)

₹80.00 — In Stock

Add to Cart

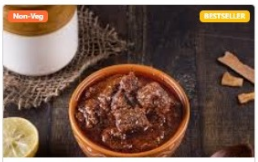
Tangy Lemon Pickle

Fresh lemons preserved in aromatic spices and salt for the perfect tang...

4.3 (78)

₹110.00 — In Stock

Add to Cart



Premium Mutton Pickle

Tender mutton pieces cooked in rich spices and preserved traditionally...

4.9 (45)

₹380.00 — 5 left

Add to Cart

Spicy Garlic Pickle

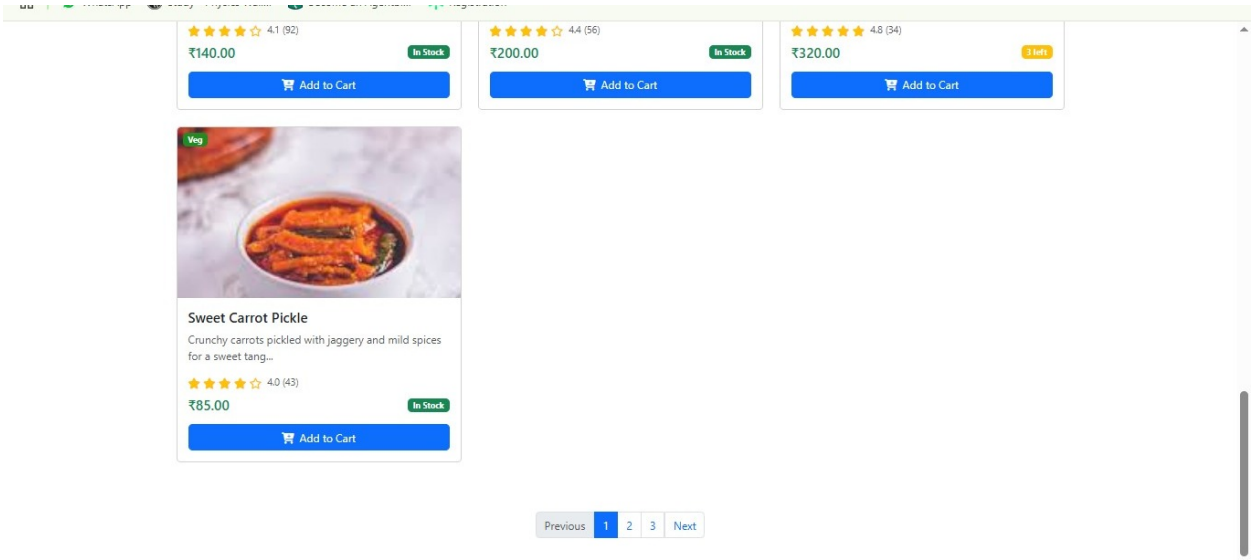
Pungent garlic cloves pickled with red chilies and traditional spices...

Coastal Fish Pickle

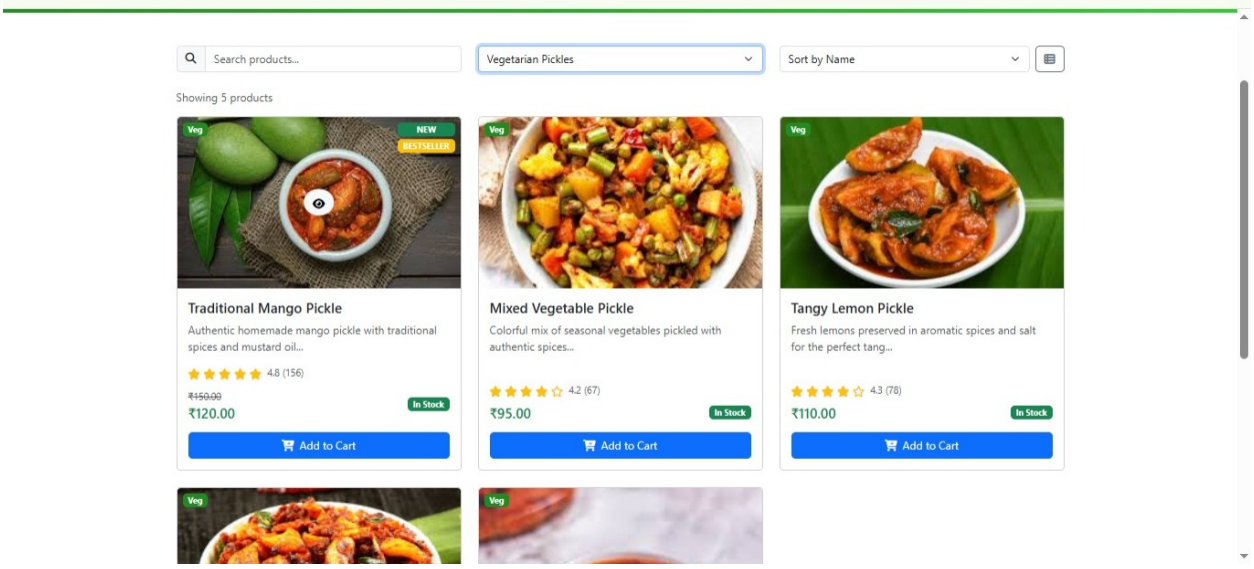
Fresh fish marinated in coastal spices and traditional recipe...

Premium Prawn Pickle

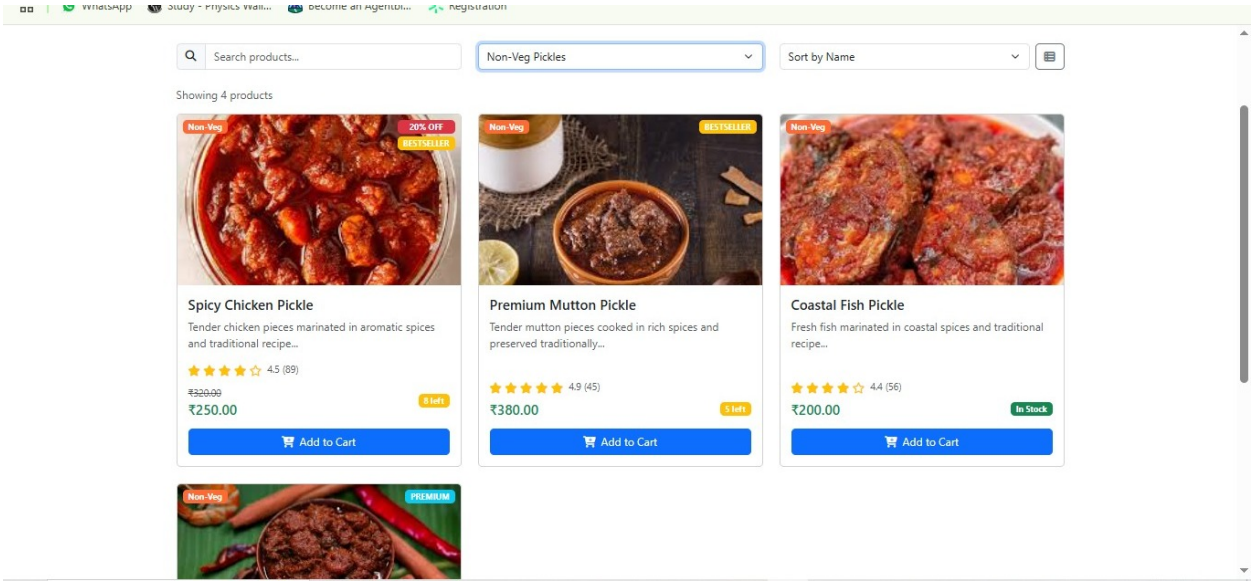
Fresh prawns cooked in rich coconut oil with traditional coastal spices...



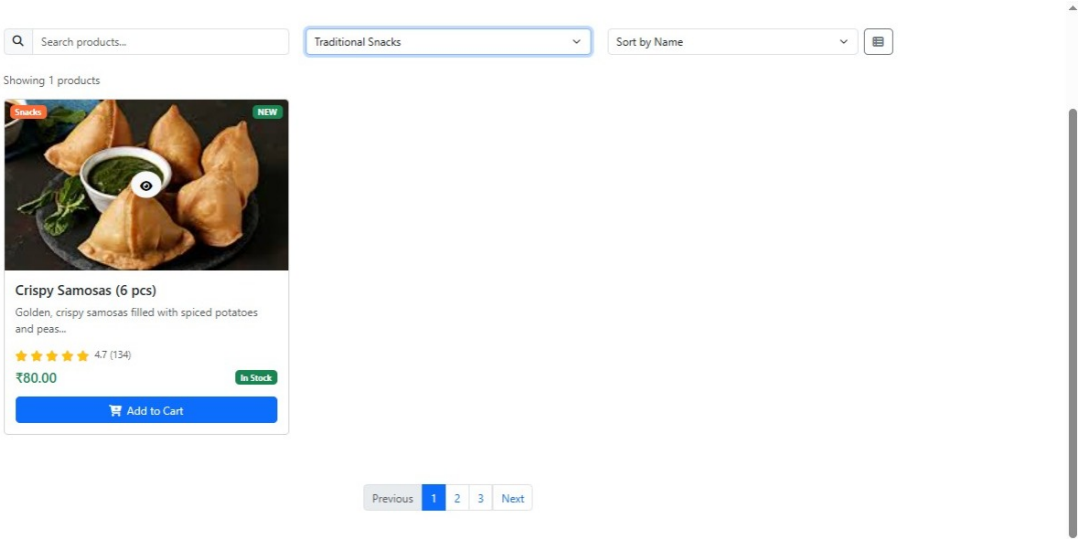
Vegetarian Pickles Page:



Non-Vegetarian Pickles Page:



Traditional Snacks Page:



Pickles Ordered Successfully! we will get back to you soon.

Exit:

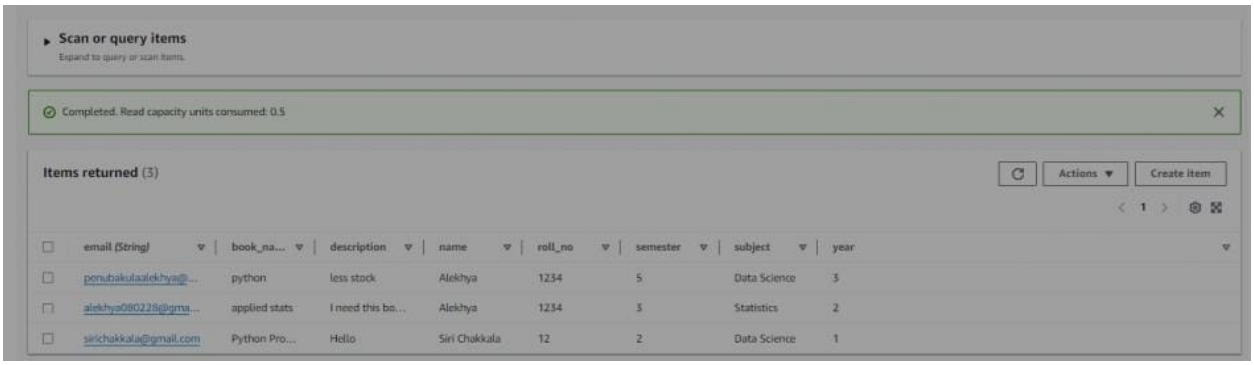
Session Ended

Please close the tab.

Dynamodb Database Updations:

Users table :

Orders table :



Products Table :

Conclusion:

The HomeMade Pickles and Snacks - Taste the Best web application successfully demonstrates the power of modern web development using Flask and AWS services to build a scalable, user-friendly e-commerce platform. By focusing on the niche of traditional, homemade pickles and snacks, the project not only highlights cultural authenticity but also bridges local products to a broader online market.
Key accomplishments of the project include:

- A fully functional **Flask-based backend** with **DynamoDB integration** for fast and scalable data storage.
- **User-friendly interfaces** for product browsing, cart management, and order placement.
- Secure **email notifications** using **AWS SNS** and **SMTP** for order confirmations and user interactions.
- Cleanly organized project structure, designed with deployment in mind for platforms like **AWS EC2**.
- Emphasis on **security**, **modularity**, and **readability** by using environment variables and best practices.

This project provides a solid foundation for future enhancements, such as adding payment gateways, user authentication, order tracking, and more advanced analytics. It also demonstrates readiness for real-world deployment and a professional understanding of full-stack development principles.

With this project, we've created more than just a web app — we've built a digital storefront that brings the flavors of tradition to the fingertips of modern consumers.