HomeMade Pickles & Snacks: Taste the Best

Description:

Home Made 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."

Scenarios:

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

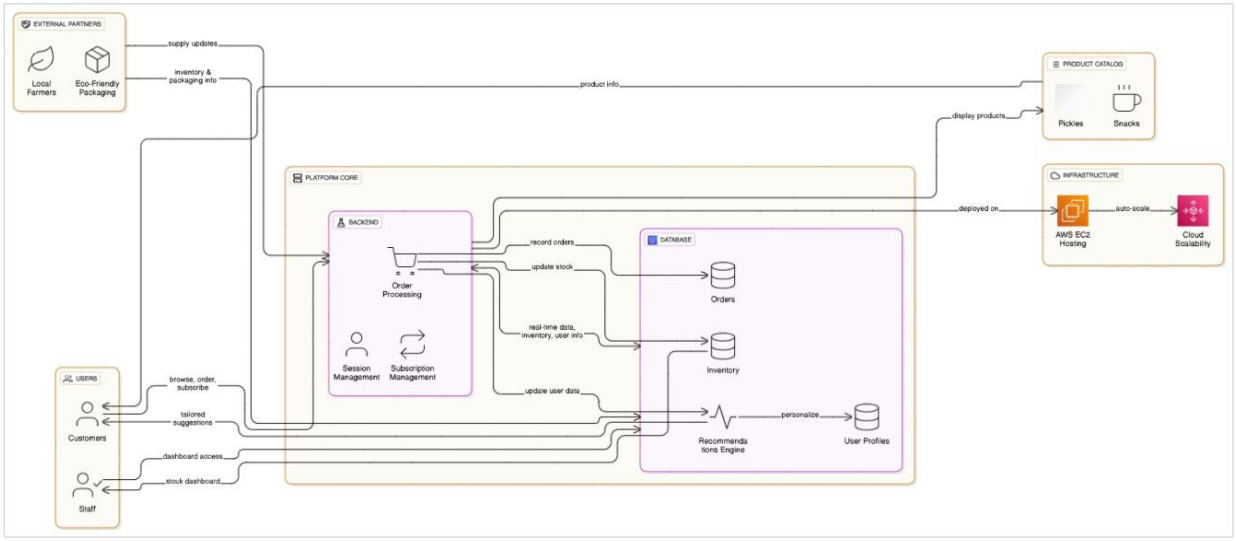
updated dashboards to monitor stock availability and fulfillment progress, ensuring timely restocking and minimizing overselling risks.

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.

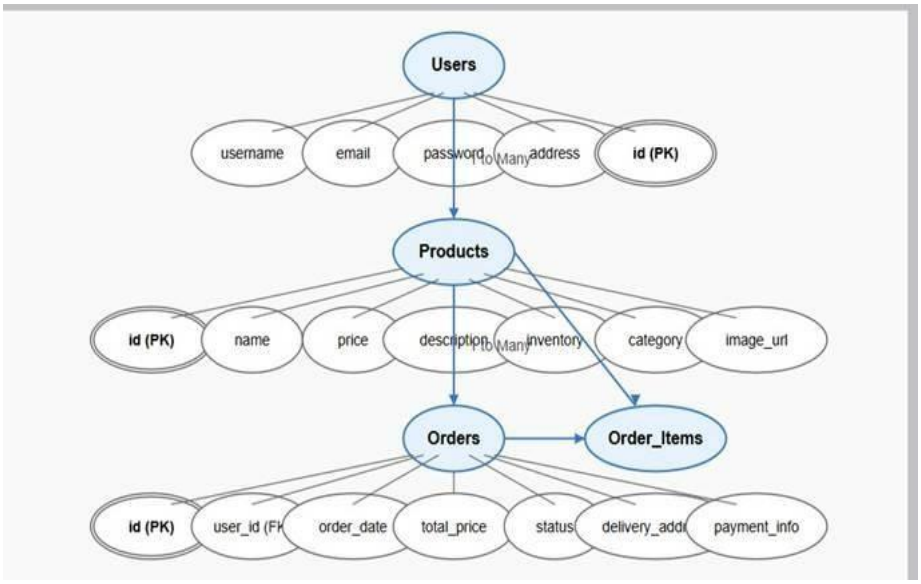
Architecture

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.



Entity Relationship (ER)Diagram

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization,data integrity, and simplified query design.



Pre-requisites

- AWS Account Setup:
  https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html
- AWS IAM (Identity and Access Management):
  https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
- AWS EC2 (Elastic Compute Cloud):
  https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
- AWS DynamoDB:
  https://docs.aws.amazon.com/amazondynamodb/Introduction.html
- Git Documentation:
  https://git-scm.com/doc
- VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)
  https://code.visualstudio.com/download

Project WorkFlow

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 User Data and order Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

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

Milestone 1 : Backend Development and Application Setup

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.

Important Instructions:

Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.

- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.

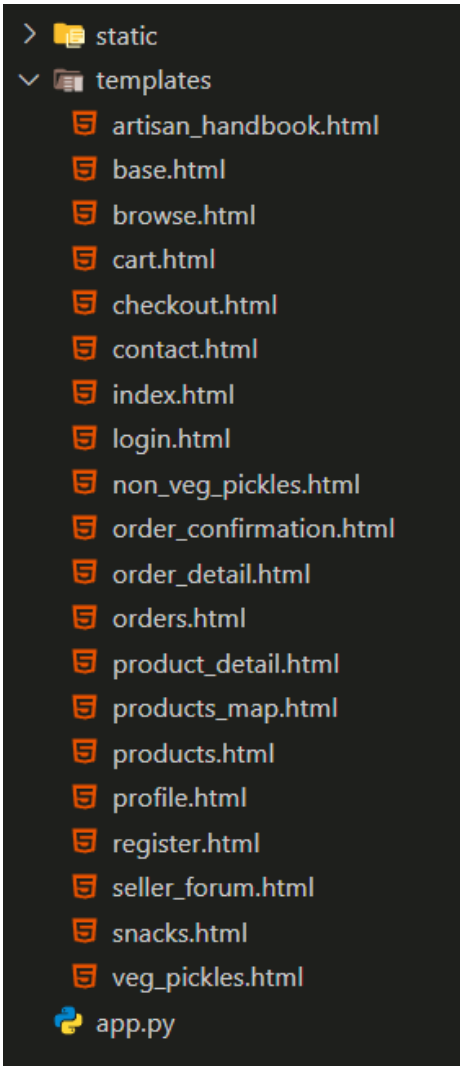
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

Post Troven Access Activation:

- Once Troven Labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
- Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.
- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

LOCAL DEPLOYMENT

- File Explorer Structure



Description of the code :

Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for
import boto3
from boto3.dynamodb.conditions import Key
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from bcrypt import hashpw, gensalt, checkpw
```

- Use boto3 to connect to DynamoDB for handling user registration, Order details database operations and also mention region_name where Dynamodb tables are created.

```
aws_session = boto3.Session(
    region_name=os.environ.get('AWS_REGION', 'ap-south-1')
)

dynamodb = aws_session.resource('dynamodb')
users_table = dynamodb.Table('PickleApp_Users')
products_table = dynamodb.Table('PickleApp_Products')
orders_table = dynamodb.Table('PickleApp_Orders')
cart_table = dynamodb.Table('PickleApp_Cart')
```

```
products = {
    'non_veg_pickles': [
        {'id': 1, 'name': 'Chicken Pickle', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 2, 'name': 'Fish Pickle', 'weights': {'250': 200, '500': 400, '1000': 800}},
        {'id': 3, 'name': 'Gongura Mutton', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 4, 'name': 'Mutton Pickle', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 5, 'name': 'Prawns Pickle', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 6, 'name': 'Chicken Pickle (Gongura)', 'weights': {'250': 350, '500': 700, '1000': 1050}},
    ],
    'veg_pickles': [
        {'id': 7, 'name': 'Traditional Mango Pickle', 'weights': {'250': 150, '500': 280, '1000': 500}},
        {'id': 8, 'name': 'Zesty Lemon Pickle', 'weights': {'250': 120, '500': 220, '1000': 400}},
        {'id': 9, 'name': 'Tomato Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 10, 'name': 'Mix Veg Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 12, 'name': 'Spicy Pandu Mirchi', 'weights': {'250': 130, '500': 240, '1000': 450}},
    ],
    'snacks': [
        {'id': 13, 'name': 'Banana Chips', 'weights': {'250': 300, '500': 600, '1000': 800}},
        {'id': 14, 'name': 'Crispy Aam-Papad', 'weights': {'250': 150, '500': 300, '1000': 600}},
        {'id': 16, 'name': 'Sweet Boondhi', 'weights': {'250': 300, '500': 600, '1000': 900}},
        {'id': 17, 'name': 'Chekkalu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 18, 'name': 'Ragi Laddu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 19, 'name': 'Dry Fruit Laddu', 'weights': {'250': 500, '500': 1000, '1000': 1500}},
        {'id': 20, 'name': 'Kara Boondi', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 21, 'name': 'Gavvalu', 'weights': {'250': 250, '500': 500, '1000': 750}},
    ]
}
```

- Routes for Web Pages
- Login Route (GET/POST):Verifies user credentials, increments login count, and redirects to the dashboard on success.

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        if not email or not password:
            flash("Email and password are required!", "danger")
            return redirect(url_for('login'))
        user, mode = get_user_by_email(email)
        if not user or not checkpw(password.encode('utf-8'), user['password'].encode('utf-8')):
            flash("Incorrect email or password! Please try again.", "danger")
            return redirect(url_for('login'))
        if mode == 'dynamodb':
            try:
                users_table.update_item(
                    Key={'email': email},
                    UpdateExpression='SET login_count = if_not_exists(login_count, :zero) + :inc',
                    ExpressionAttributeValues={':inc': 1, ':zero': 0}
                )
            except Exception as e:
                print(f"DynamoDB update error: {e}")
        elif mode == 'local':
            users = load_users()
            for u in users:
                if u['email'] == email:
                    u['login_count'] = u.get('login_count', 0) + 1
            save_users(users)
        session['user_email'] = user['email']
        session['user_name'] = user['name']
        session['phone'] = user.get('phone', '')
        flash("Login successful!", "success")
        next_page = request.form.get('next')
        if next_page and next_page.startswith('/'):
            return redirect(next_page)
        return redirect(url_for('profile'))
    return render_template('login.html')
```

- SignUp route: Collecting registration data, hashes the password, and stores user details in the database.

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirm_password']
        phone = request.form['phone']
        if not name or not email or not password or not confirm_password or not phone:
            flash("All fields are mandatory! Please fill out the entire form.", "danger")
            return redirect(url_for('register'))
        if password != confirm_password:
            flash("Passwords do not match! Please try again.", "danger")
            return redirect(url_for('register'))
        user, mode = get_user_by_email(email)
        if user:
            flash("User already exists! Please log in.", "info")
            return redirect(url_for('login'))
        hashed_password = hashpw(password.encode('utf-8'), gensalt()).decode('utf-8')
        user_obj = {
            'email': email,
            'name': name,
            'password': hashed_password,
            'phone': phone,
            'login_count': 0,
            'street': '',
            'city': '',
            'pincode': '',
            'country': 'India',
            'registration_date': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
        save_user(user_obj, 'dynamodb' if mode == 'dynamodb' else 'local')
        try:
            sns.publish(
                TopicArn=sns_topic_arn,
                Message=f'New user registered: {name} ({email})',
                Subject='New User Registration'
            )
        except Exception as e:
            print(f"SNS error: {e}")
        try:
            send_email(email, "Welcome to PickleApp!", f"Hi {name},\n\nThank you for registering at PickleApp!")
        except Exception as e:
            print(f"Email error: {e}")
        session['user_email'] = email
        session['user_name'] = name
        session['phone'] = phone
        flash("Registration successful!", "success")
        return redirect(url_for('profile'))
    return render_template('register.html')
```

- Logout route:The user can Logout so that the user can get back to the Login Page

```python
@app.route('/logout')
def logout():
    session.clear()
    flash("Logged out successfully!", "success")
    return redirect(url_for('login'))
```

- Home Route: Home page contains the routing for different categories which are Veg_pickles,Non_Veg_pickles,Snacks.

```python
@app.route('/veg-pickles')
def veg_pickles():
    products = get_products_by_category('veg_pickles')
    return render_template('veg_pickles.html', products=products, category='veg_pickles')

@app.route('/non-veg-pickles')
def non_veg_pickles():
    products = get_products_by_category('non_veg_pickles')
    return render_template('non_veg_pickles.html', products=products, category='non_veg_pickles')

@app.route('/snacks')
def snacks():
    products = get_products_by_category('snacks')
    return render_template('snacks.html', products=products, category='snacks')
```

- Check out Route:
-

```python
@app.route('/checkout')
def checkout():
    if not session.get('user_email'):
        flash("Please log in to proceed with checkout.", "danger")
        return redirect(url_for('login', next='/checkout'))
    return render_template('checkout.html')

@app.route('/order-confirmation')
def order_confirmation():
    if not session.get('user_email'):
        flash("Please log in to view order confirmation.", "danger")
        return redirect(url_for('login', next='/order-confirmation'))
    return render_template('order_confirmation.html')
```

Description: These are the image routes are used to fall back in the local host for the static images in the static folder .

Image Routes:

```python
@app.route('/api/product/<int:product_id>')
def api_product(product_id):
    time.sleep(1)
    img_map = {
        'Chicken Pickle': '/static/img/chicken-pickle.jpg',
        'Fish Pickle': '/static/img/fish-pickle.jpg',
        'Gongura Mutton': '/static/img/gongura-mutton-pickle.webp',
        'Mutton Pickle': '/static/img/mutton-pickle.jpg',
        'Prawns Pickle': '/static/img/prawn-pickle.webp',
        'Chicken Pickle (Gongura)': '/static/img/gongura-chicken-pickle.jpg',
        'Traditional Mango Pickle': '/static/img/mango-pickle.jpg',
        'Zesty Lemon Pickle': '/static/img/zesty-lemon-pickle.png',
        'Tomato Pickle': '/static/img/tomato-pickle.webp',
        'Mix Veg Pickle': '/static/img/mixed-veg-pickle.jpg',
        'Spicy Pandu Mirchi': '/static/img/red-chilli-pickle.jpg',
        'Banana Chips': '/static/img/banana-chips.jpg',
        'Crispy Aam-Papad': '/static/img/crispy-aam-papad.jpg',
        'Sweet Boondhi': '/static/img/sweet-boondhi.jpg',
        'Chekkalu': '/static/img/chekkalu.jpg',
        'Ragi Laddu': '/static/img/ragi-ladoo.jpg',
        'Dry Fruit Laddu': '/static/img/dry-fruit.jpg',
        'Kara Boondi': '/static/img/kaaraa-boondhi.jpg',
        'Gavvalu': '/static/img/gavvalu.jpg',
    }
    desc_map = {
        'non_veg_pickles': 'Authentic homemade non-veg pickles, rich in flavor and tradition.',
        'veg_pickles': 'Classic vegetarian pickles made with fresh ingredients.',
        'snacks': 'Crispy, crunchy, and delicious snacks for every occasion.',
    }
```

Milestone 2 : AWS Account Setup

Important Notice: Use Troven Labs for AWS Access

Students are strictly advised not to create their own AWS accounts, as doing so may incur charges. Instead, we have set up a dedicated section called “Labs” on the Troven platform, which provides temporary and cost-free access to AWS services.

Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

All steps involving AWS (such as deploying to EC2, connecting to DynamoDB, or using SNS) must be carried out within the Troven Labs platform, as we've configured temporary credentials for each student.

Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.

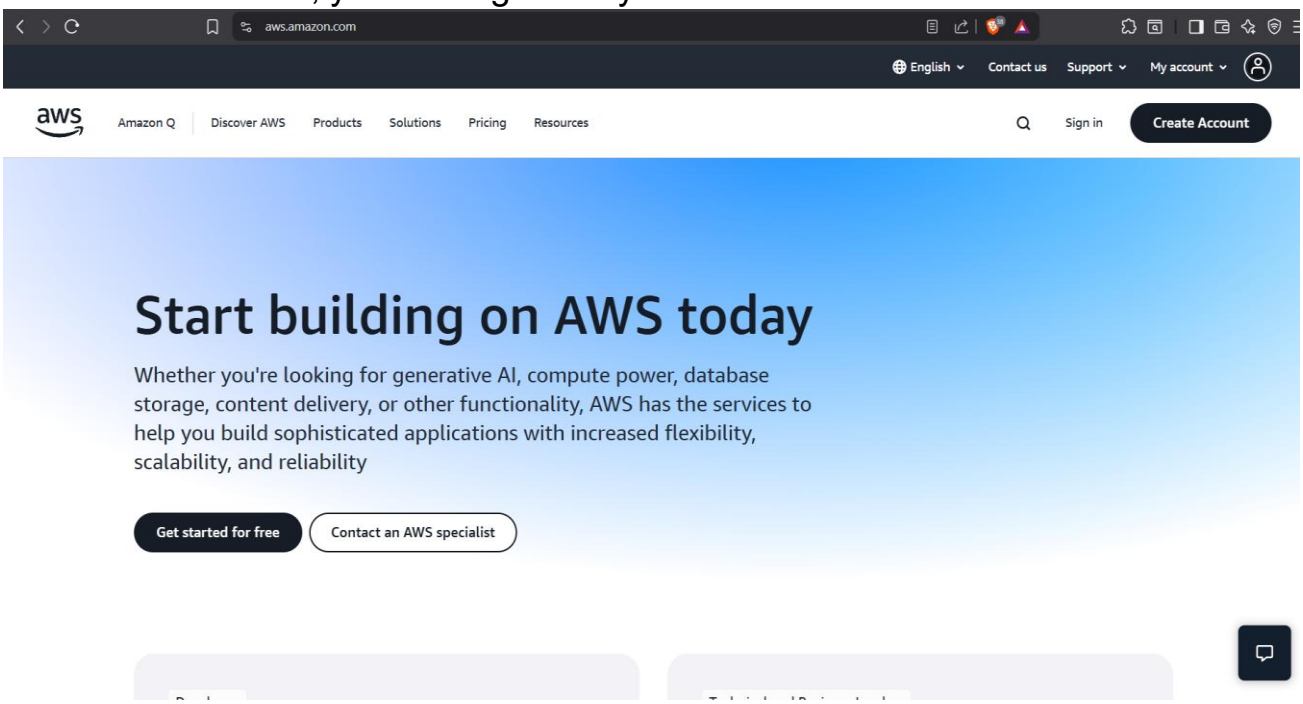
Please refer the below link -

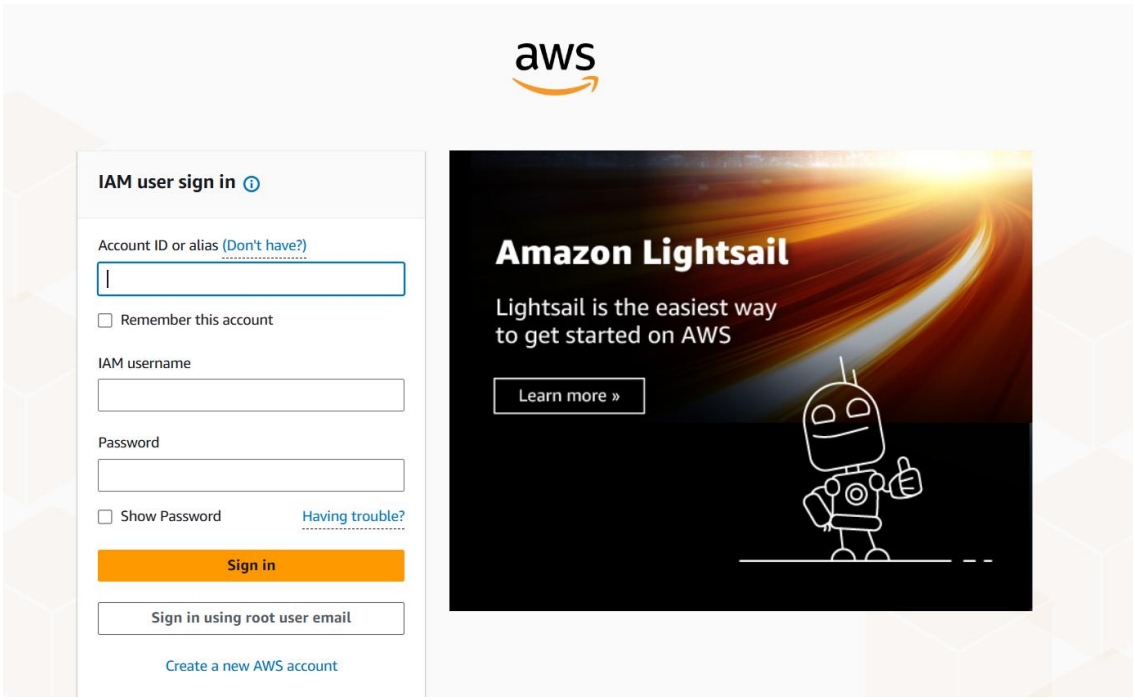
https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd- 28qgI/view?usp=sharing

AWS Account Setup and Login

This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.

- Go to the AWS website (https://aws.amazon.com/).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.





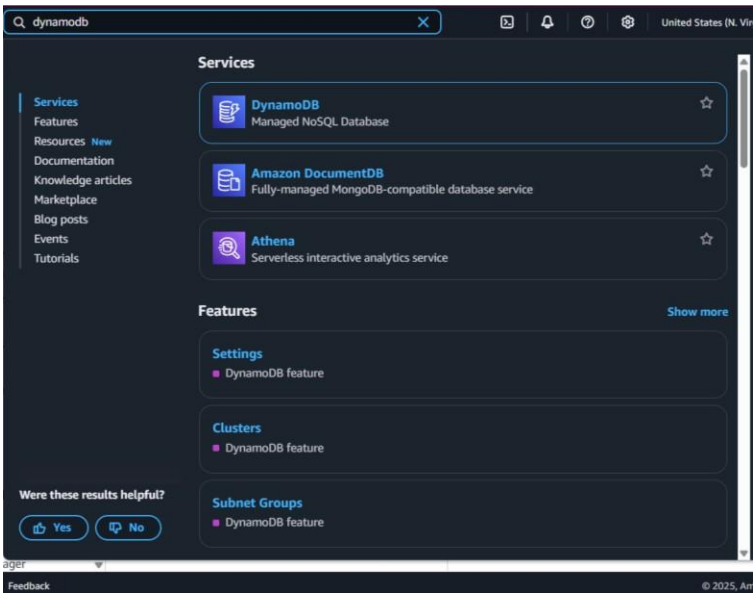
- Log in to the AWS Management Console
- After setting up your account, log in to the AWS Management Console.

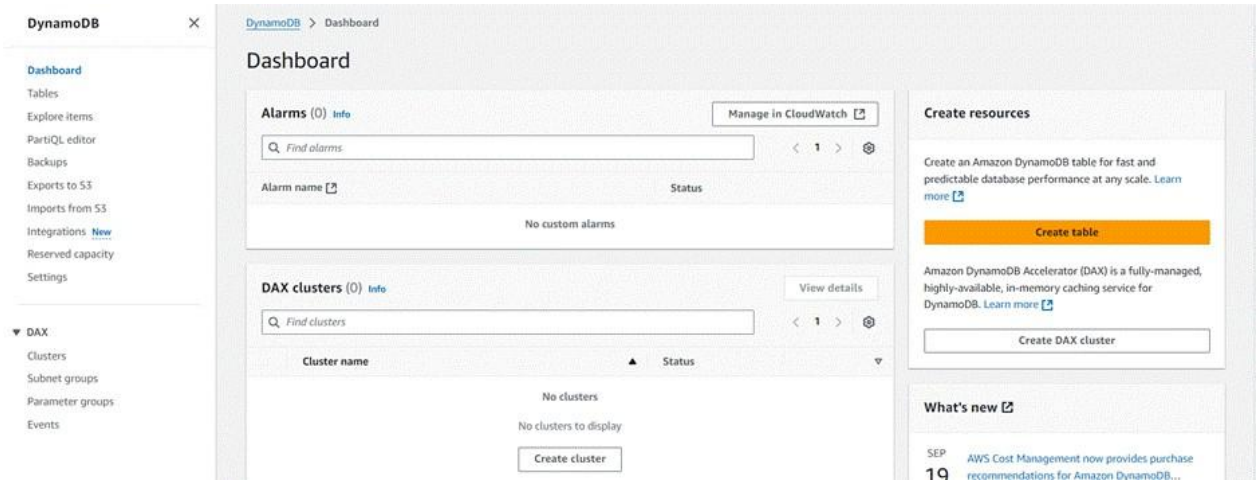
Milestone 3 : DynamoDB Database Creation and Setup

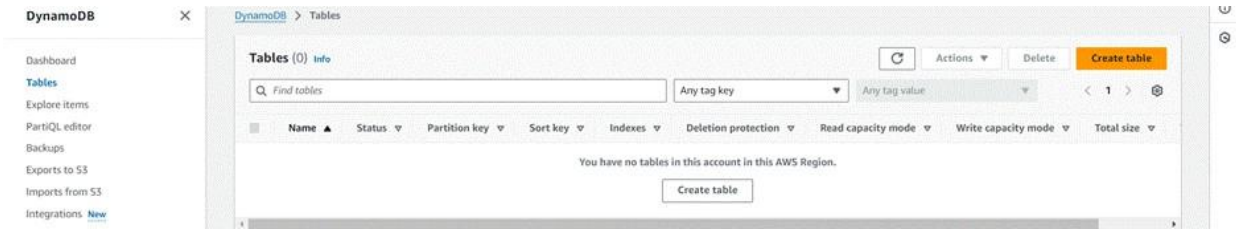
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high- performance data storage for seamless backend operations.

Navigate to the DynamoDB

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

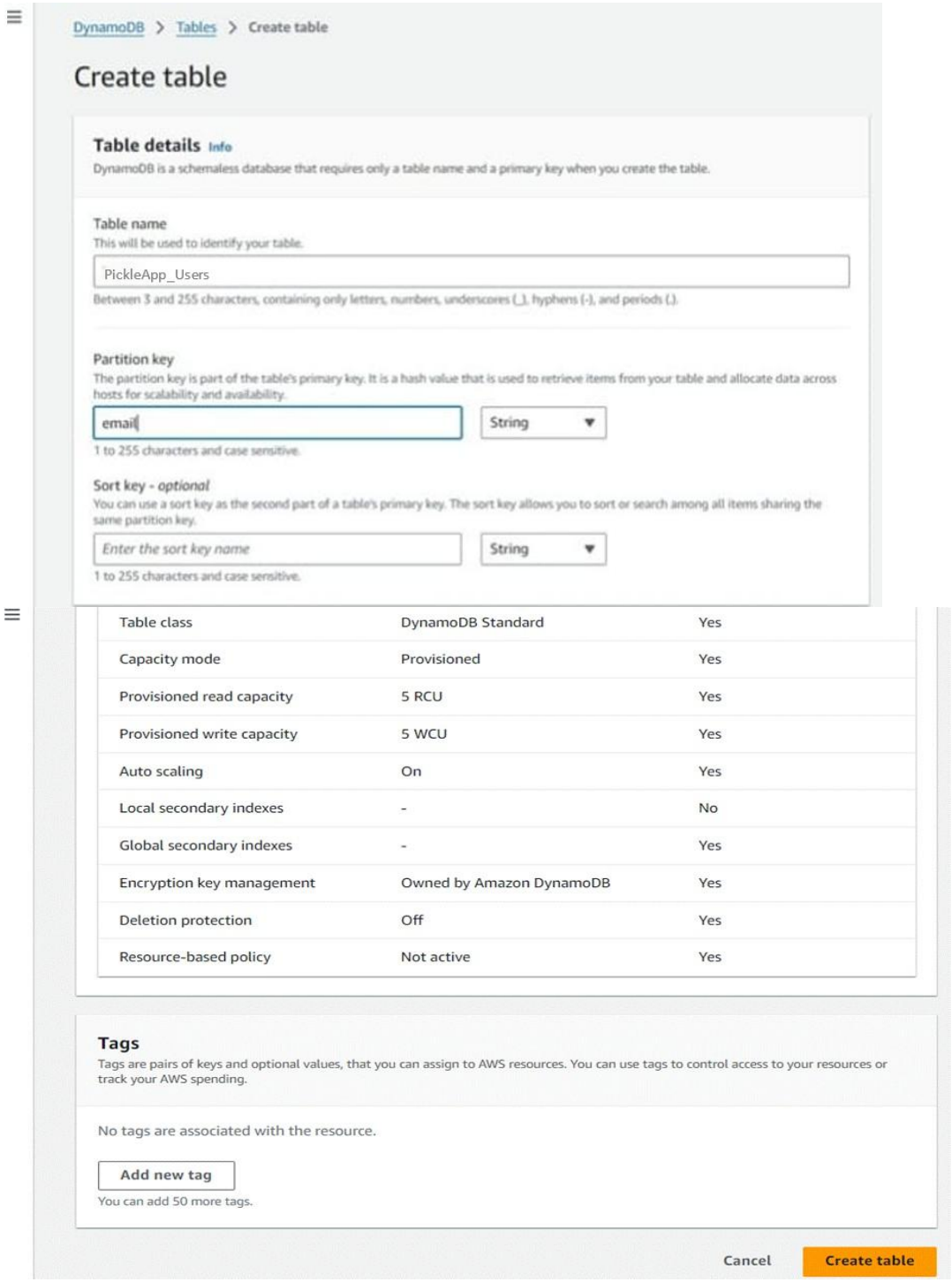






Create a DynamoDB table for storing data

- Create PickleApp_Users table with partition key “Username” with type String and click on create tables.



- Follow the same steps to create an PickleApp_Cart table

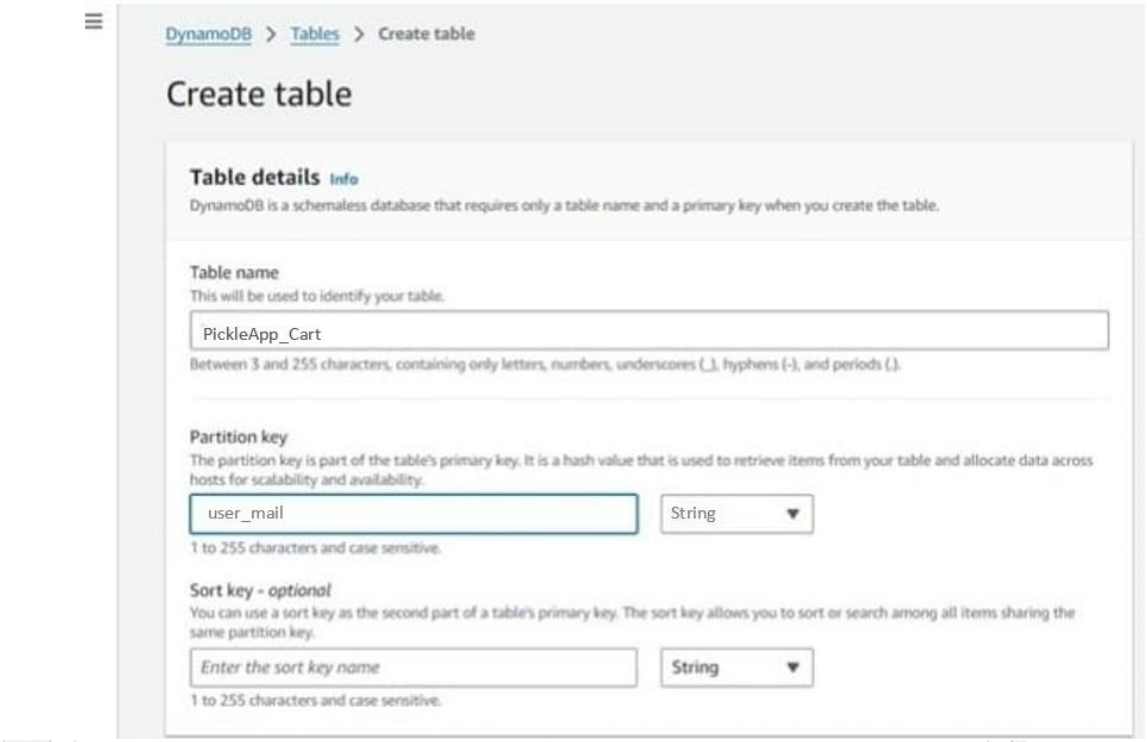


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

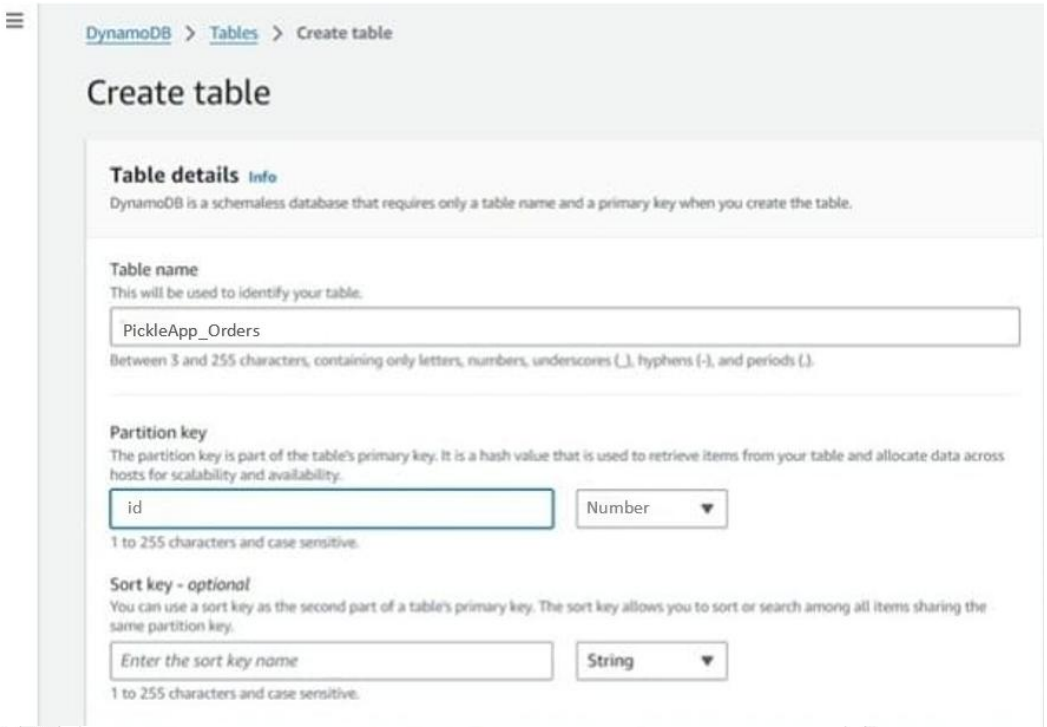
No tags are associated with the resource.

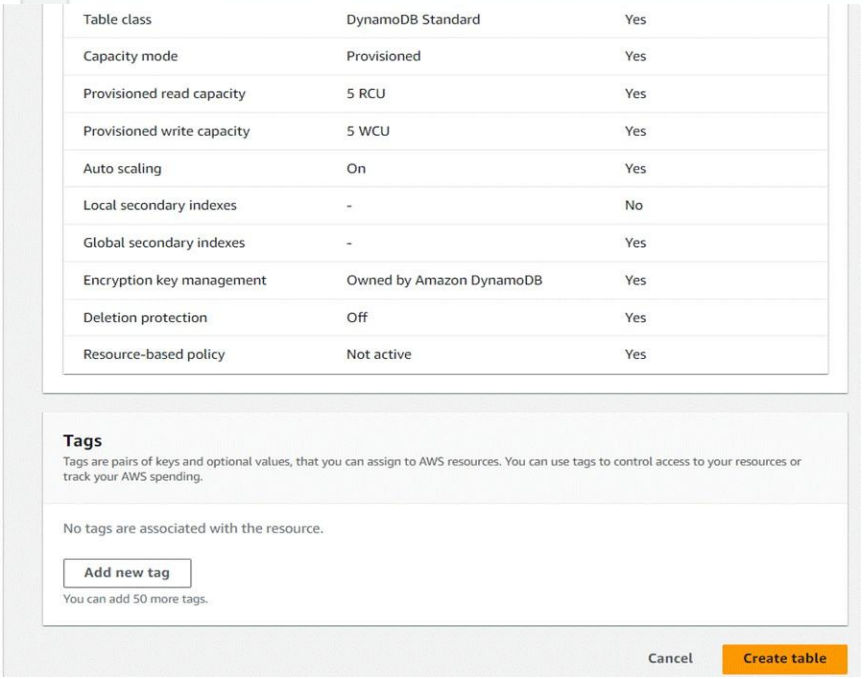
Add new tag

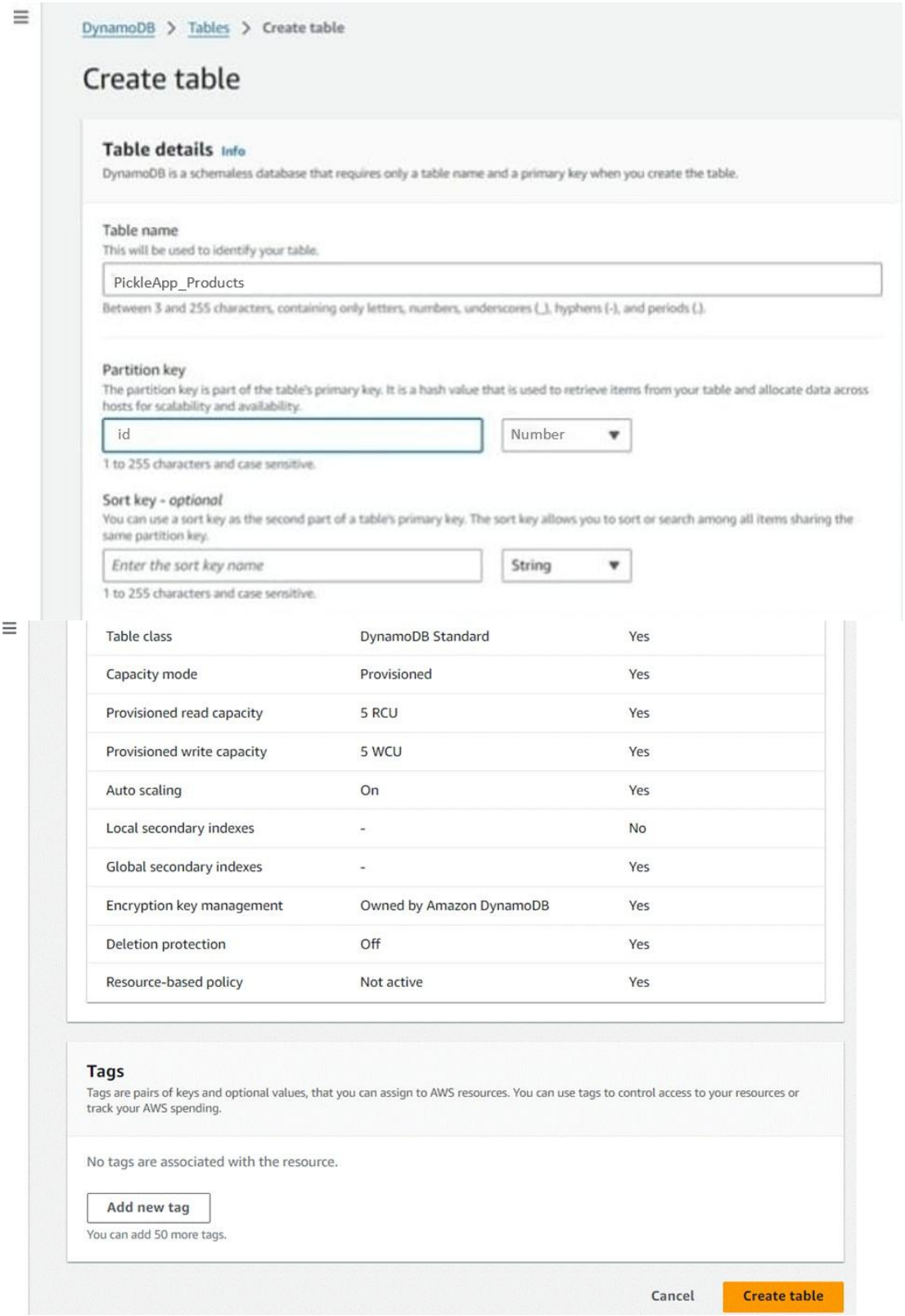
You can add 50 more tags.

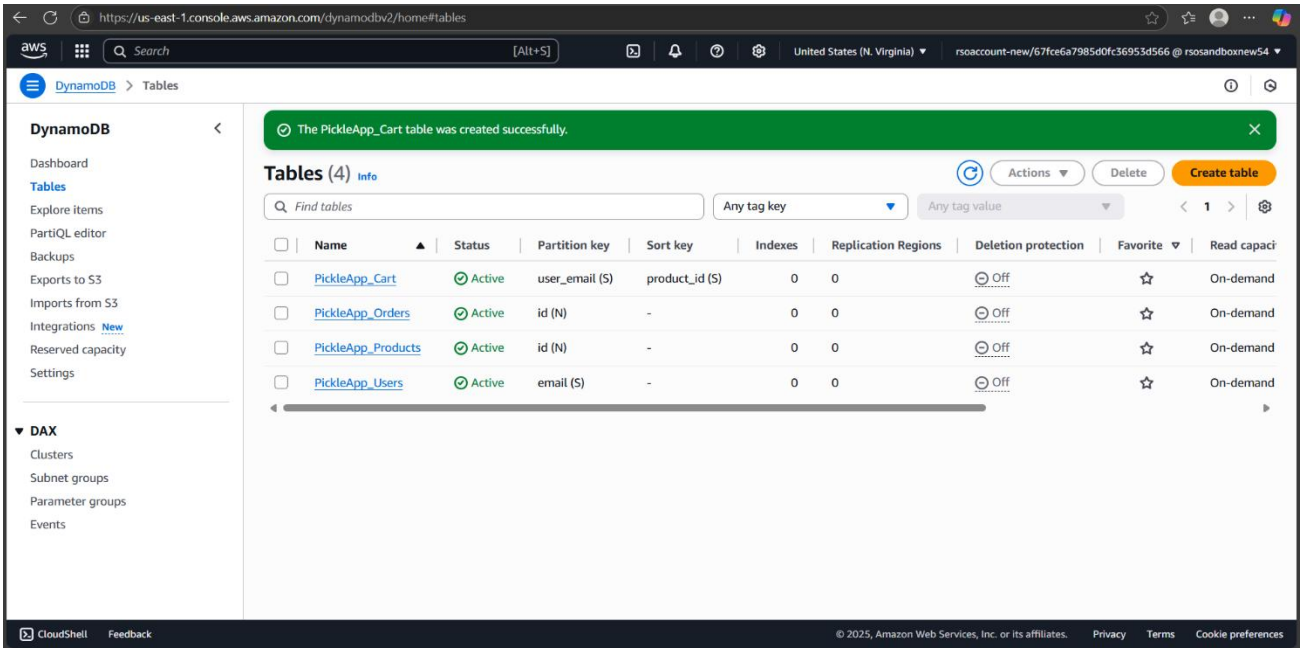
Cancel Create table

- Now for the PickleApp_Orders and for the PickleApp_Products.



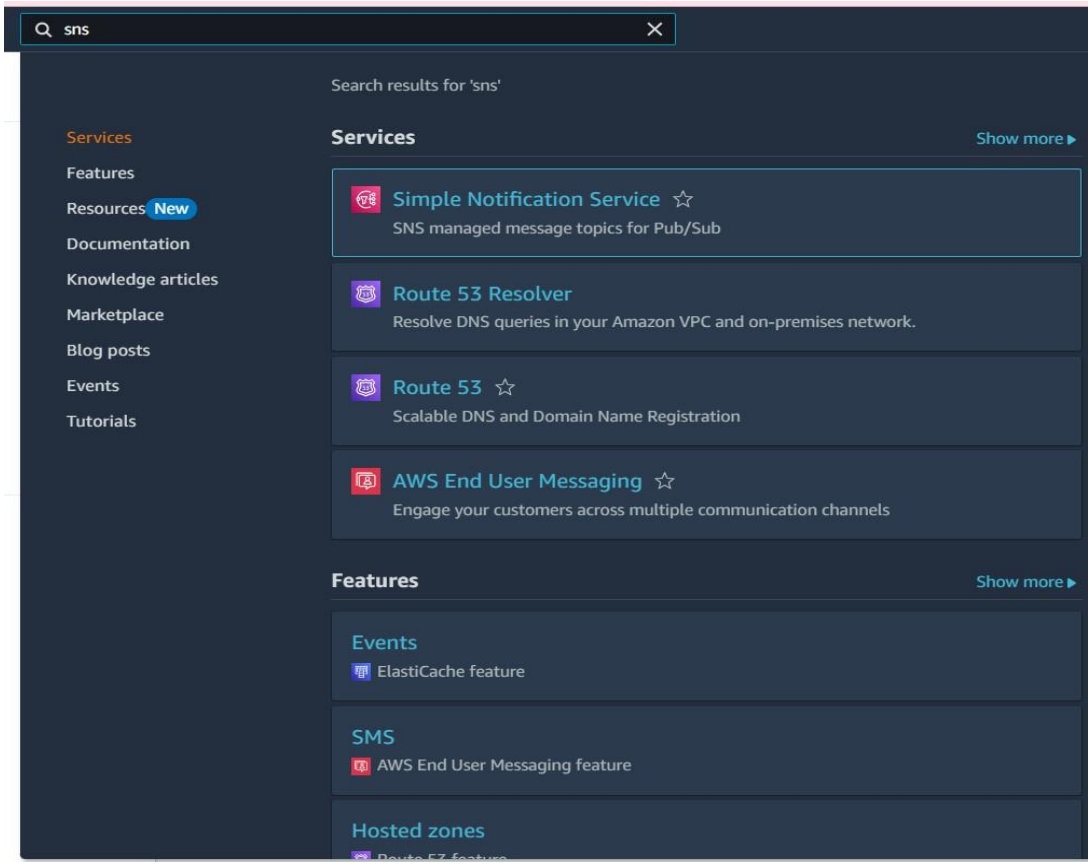


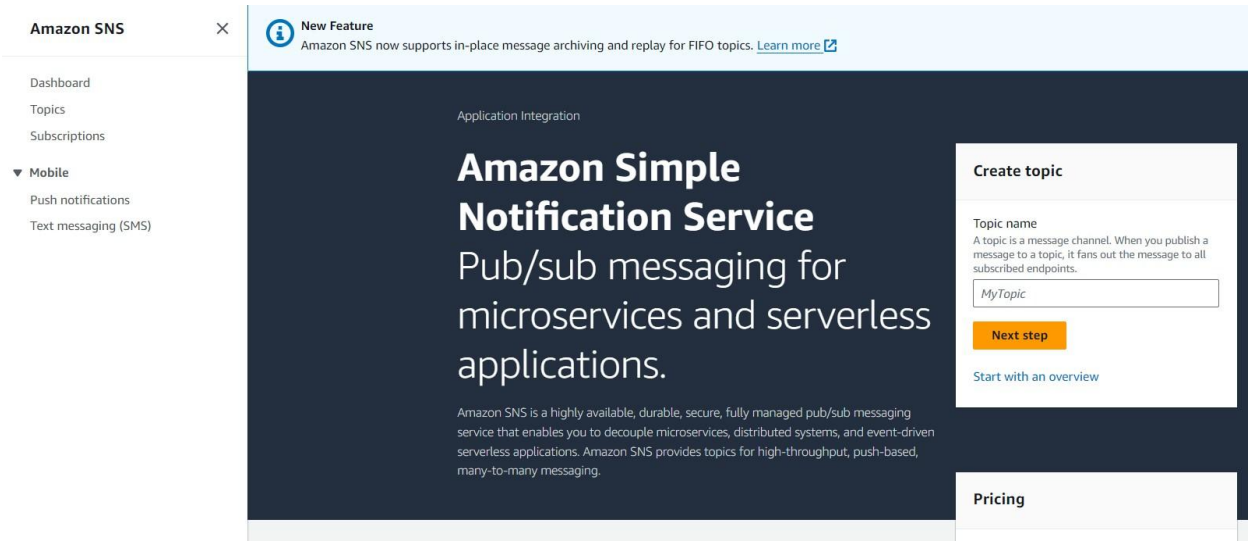

DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
PickleApp_Products
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
id
Number
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.
Table class DynamoDB Standard Yes
Capacity mode Provisioned Yes
Provisioned read capacity 5 RCU Yes
Provisioned write capacity 5 WCU Yes
Auto scaling On Yes
Local secondary indexes - No
Global secondary indexes - Yes
Encryption key management Owned by Amazon DynamoDB Yes
Deletion protection Off Yes
Resource-based policy Not active Yes
Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.
No tags are associated with the resource.
Add new tag
You can add 50 more tags.
Cancel
Create table



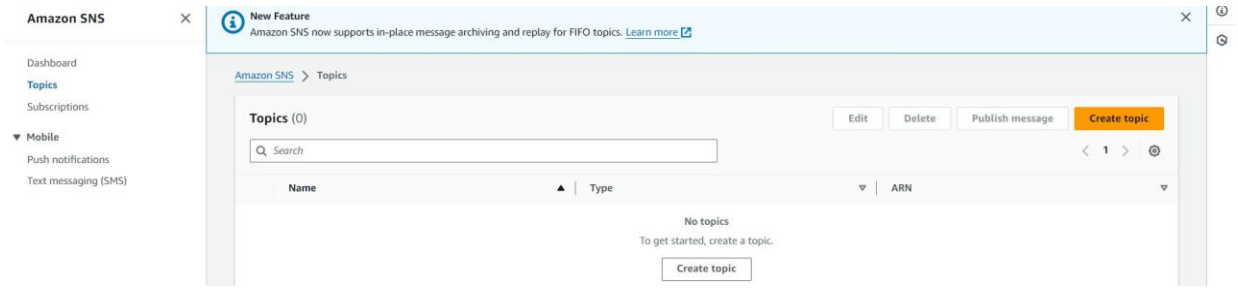
Milestone 4 : SNS Notification Setup:

- Create SNS topics for sending email notifications for ordering alerts
    - In the AWS Console, search for SNS and navigate to the SNS Dashboard.

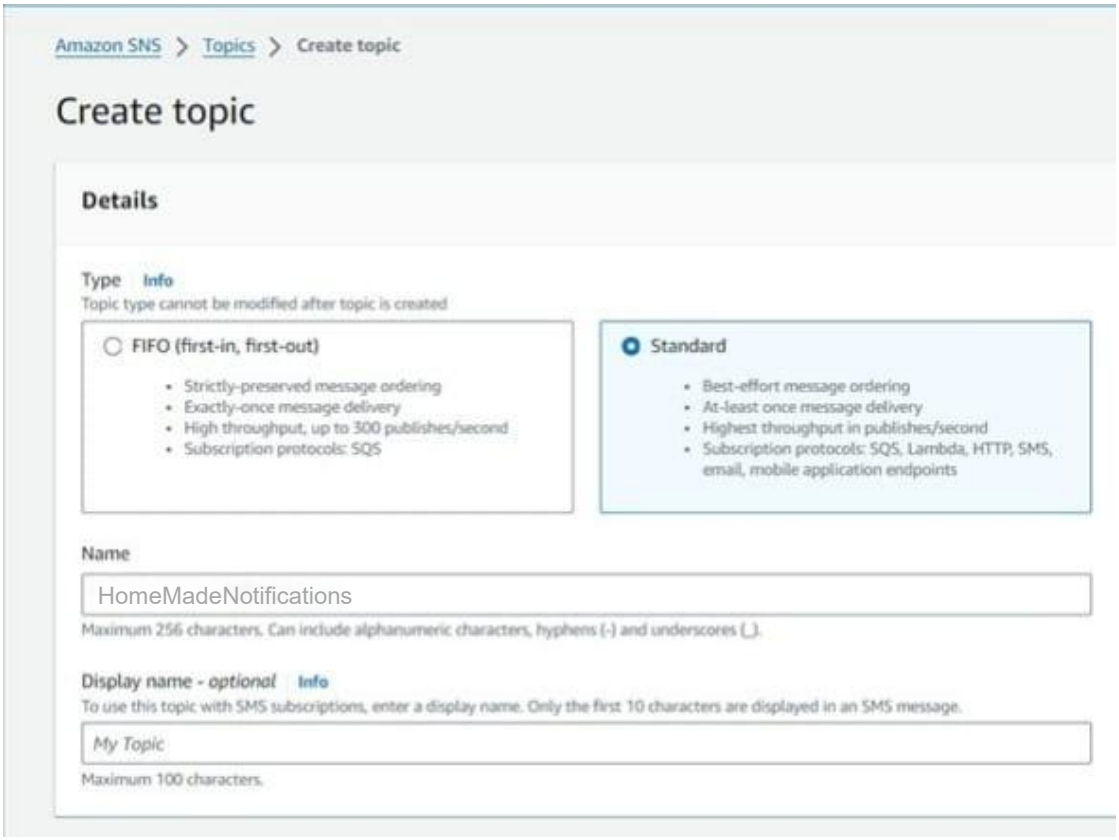


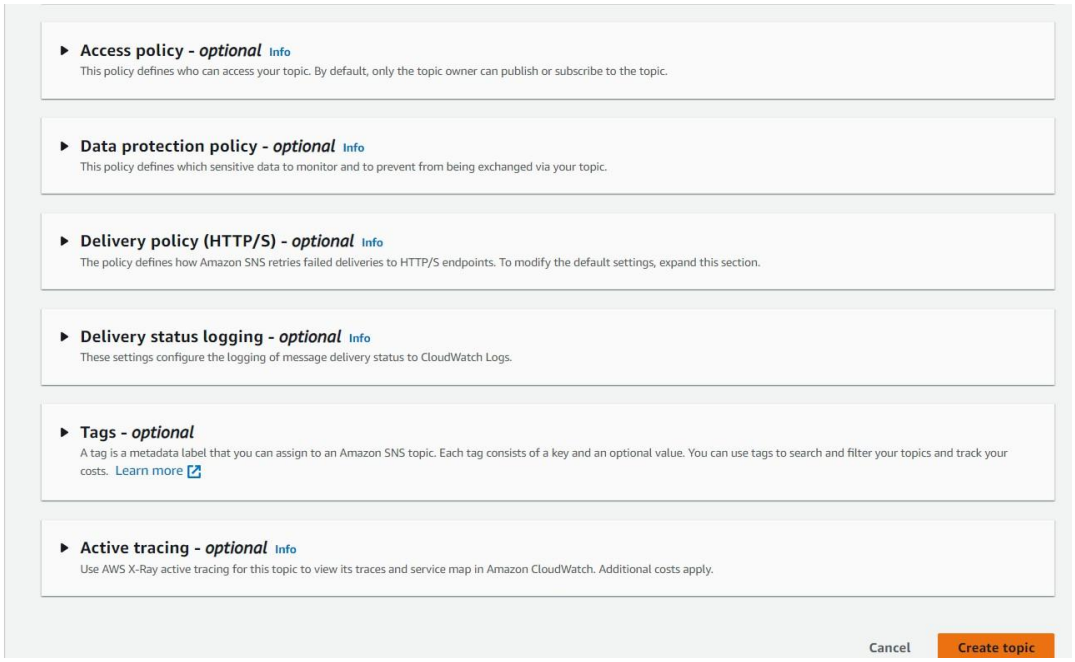


- Click on **Create Topic** and choose a name for the topic.

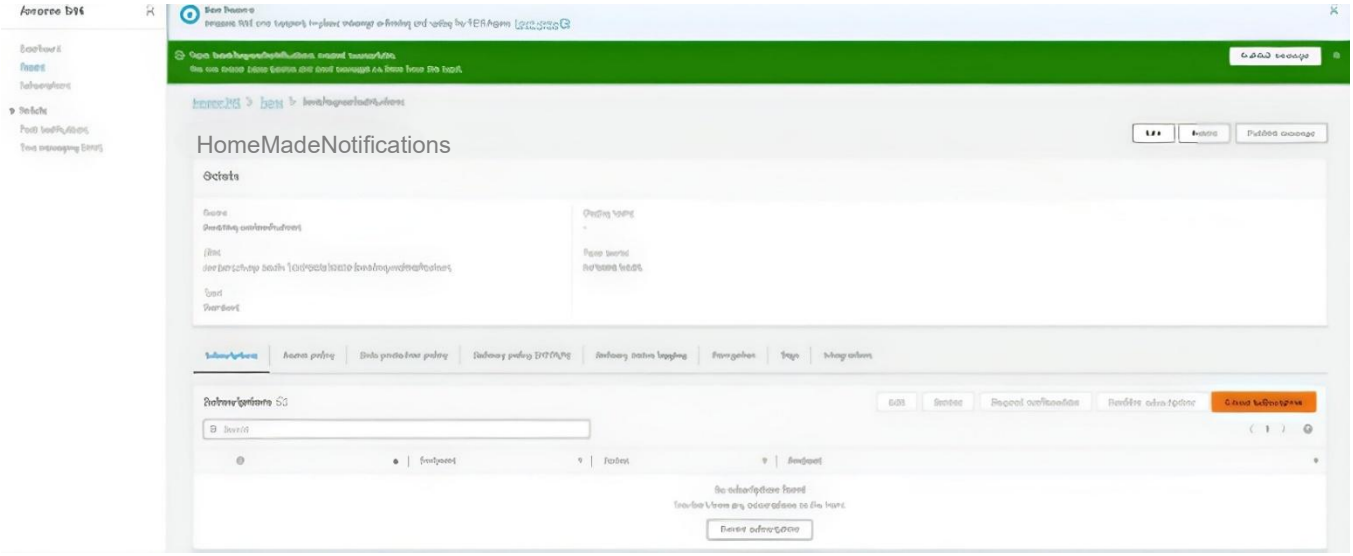


- Choose Standard type for general notification use cases and Click on Create Topic.

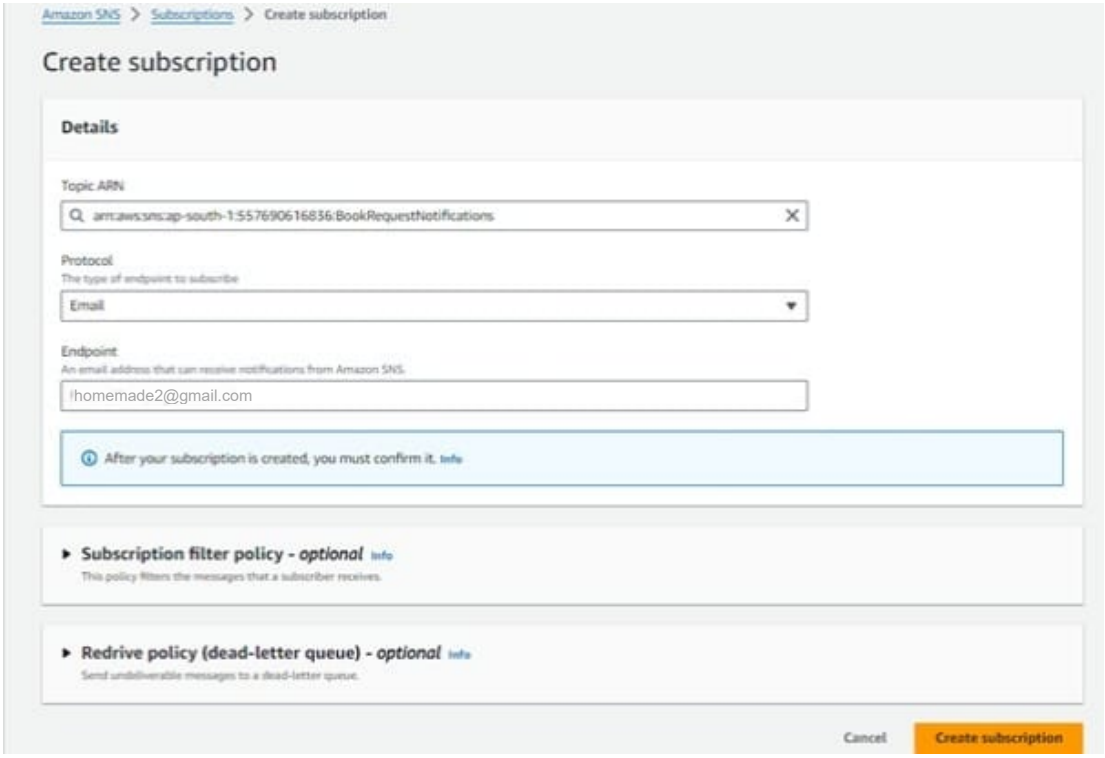

Amazon SNS > Topics > Create topic
Create topic
Details
Type Info
Topic type cannot be modified after topic is created
FIFO (first-in, first-out)
Strictly-preserved message ordering
Exactly-once message delivery
High throughput, up to 300 publishes/second
Subscription protocols: SQS
Standard
Best-effort message ordering
At-least once message delivery
Highest throughput in publishes/second
Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints
Name
HomeMadeNotifications
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).
Display name - optional Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.

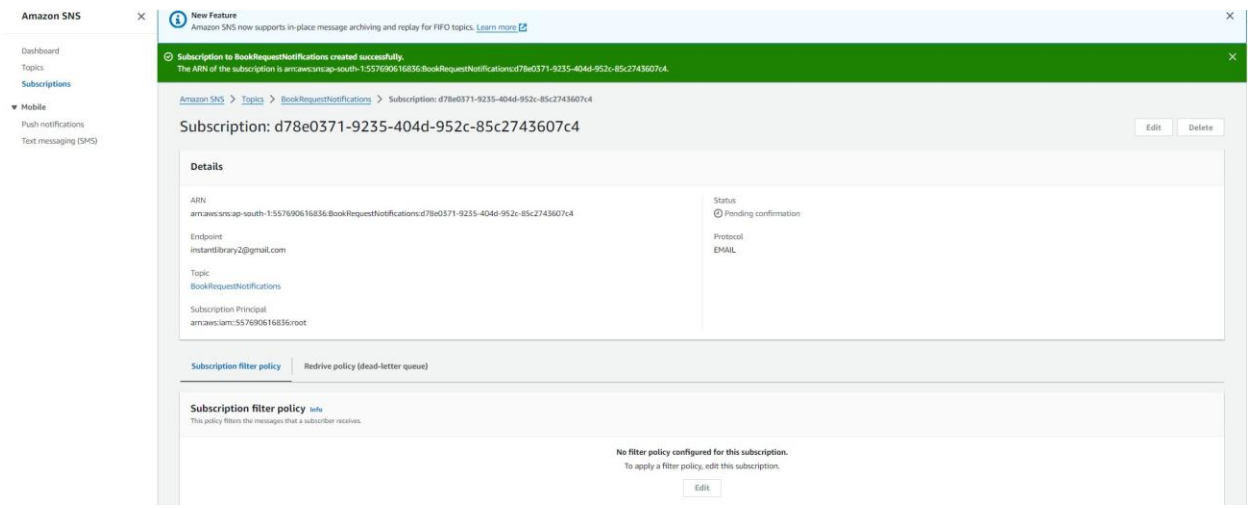

Access policy - optional Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
Data protection policy - optional Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
Delivery policy (HTTP/S) - optional Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
Delivery status logging - optional Info
These settings configure the logging of message delivery status to CloudWatch Logs.
Tags - optional
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more
Active tracing - optional Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.
Cancel
Create topic

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

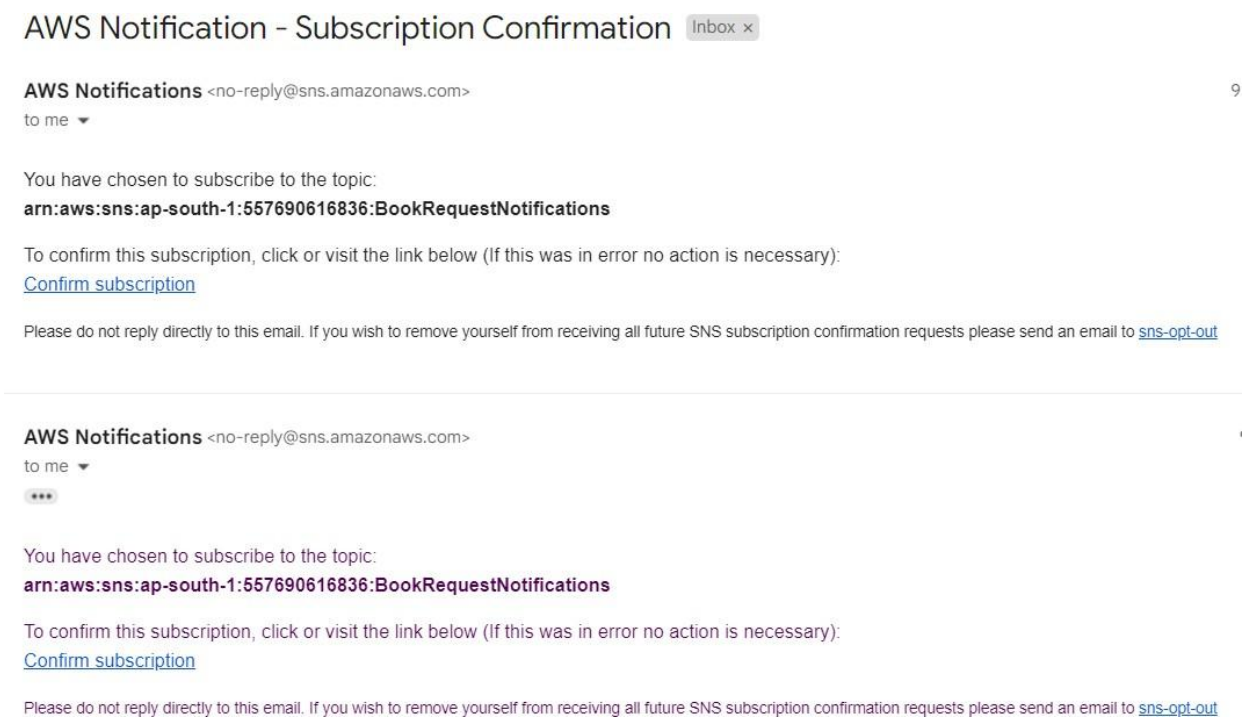


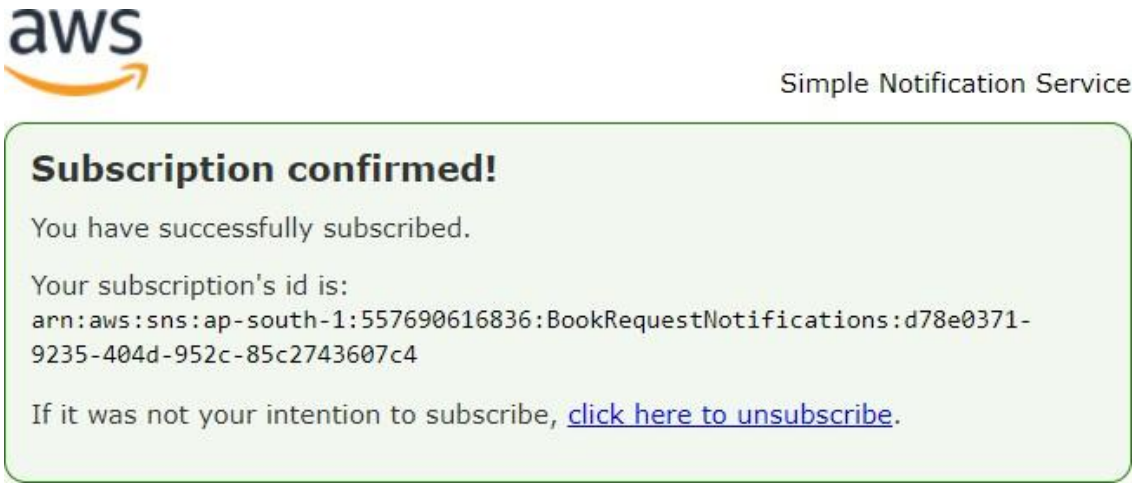
- Subscribe users (or admin staff) to this topic via Email. When a order request is made, notifications will be sent to the subscribed emails**.**





- 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.





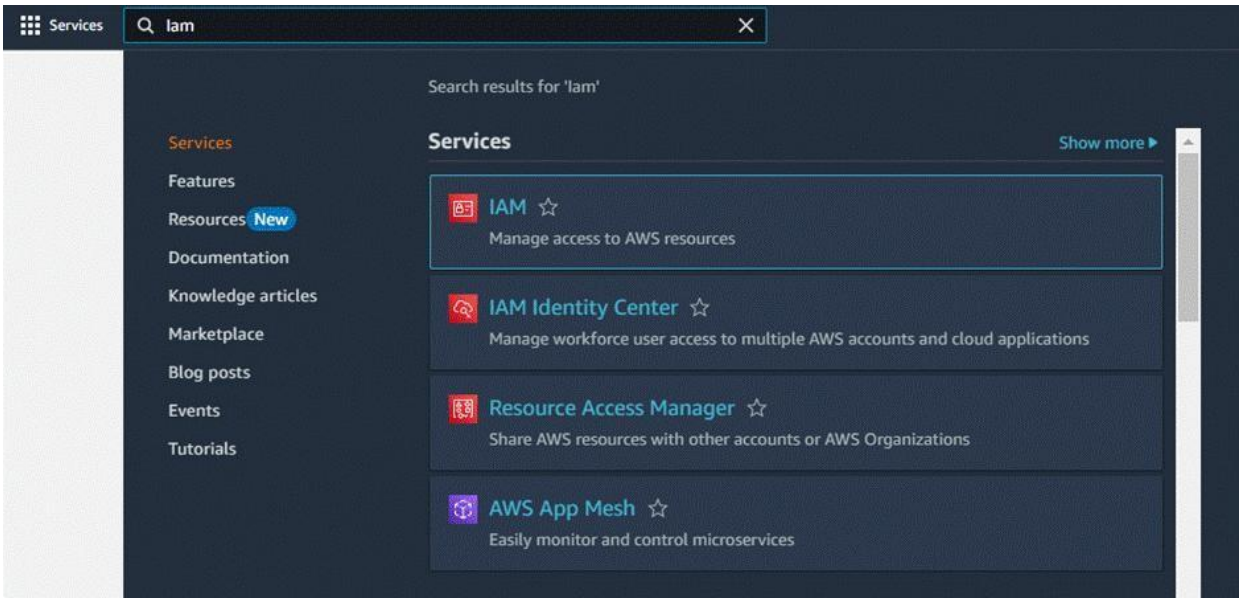
- Successfully done with the SNS mail subscription and setup, now store the ARN link.

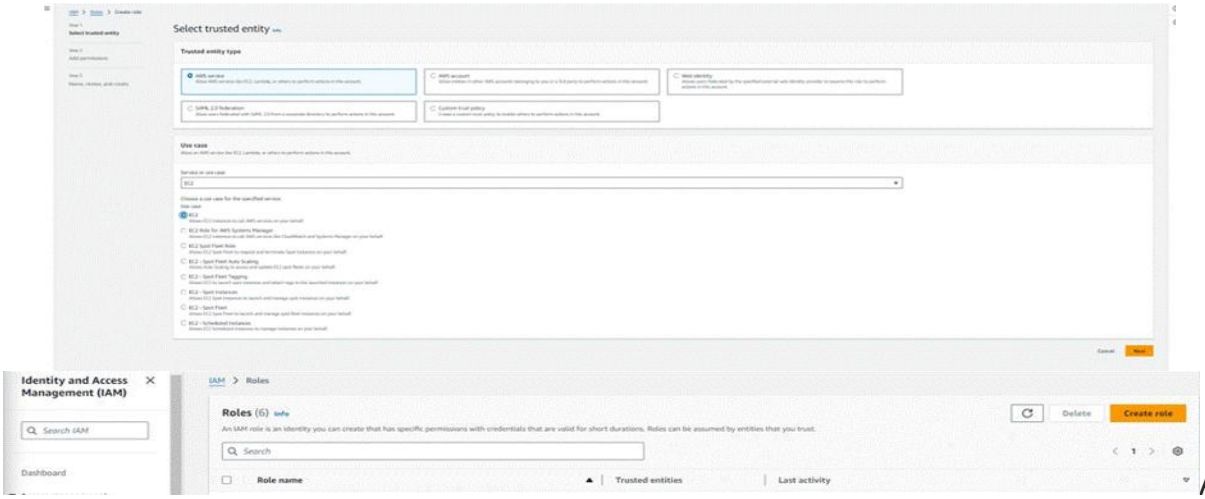
Milestone 5 : IAM Role Setup

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

Create IAM Role.

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

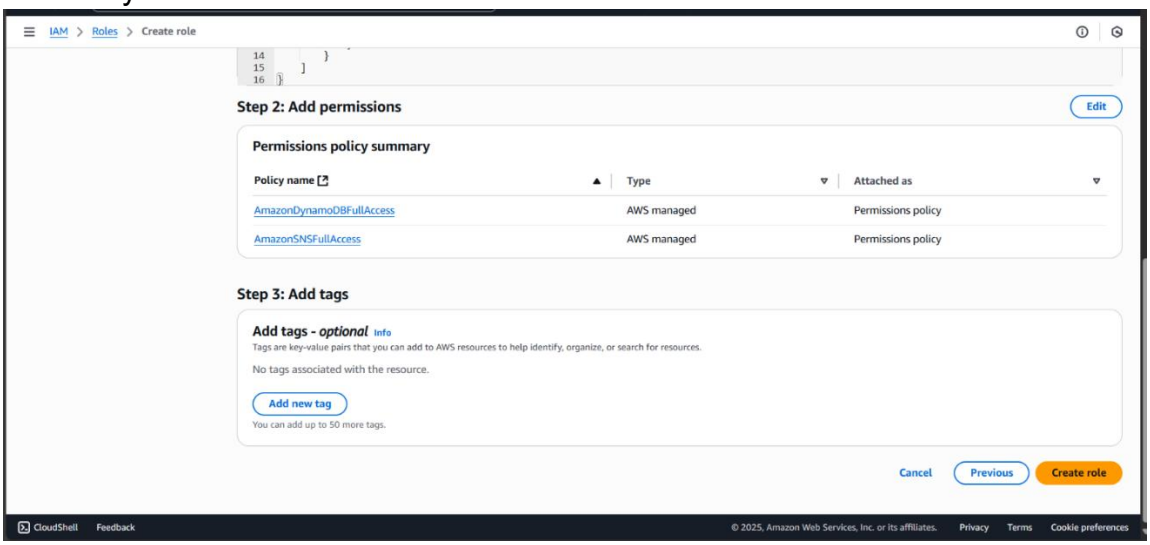


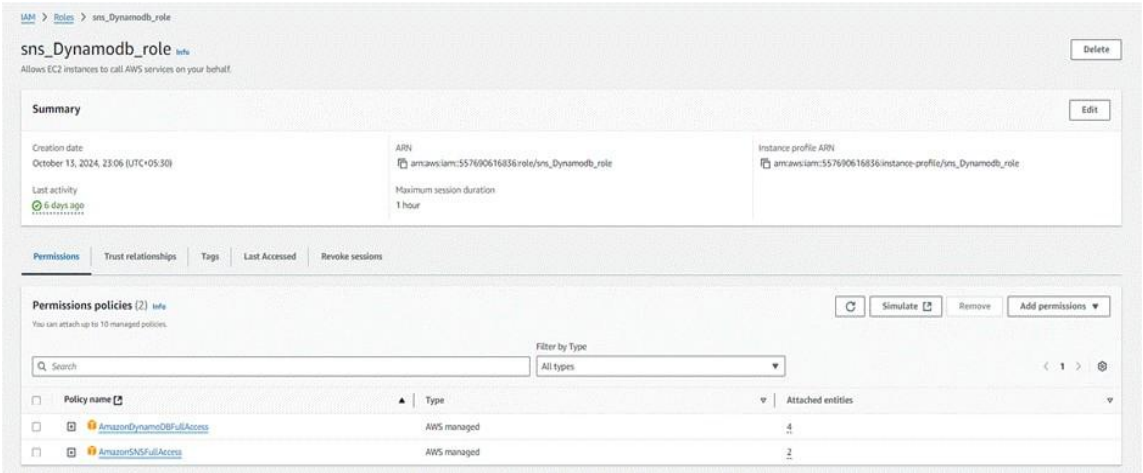


Attach Policies

Attach the following policies to the role:

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

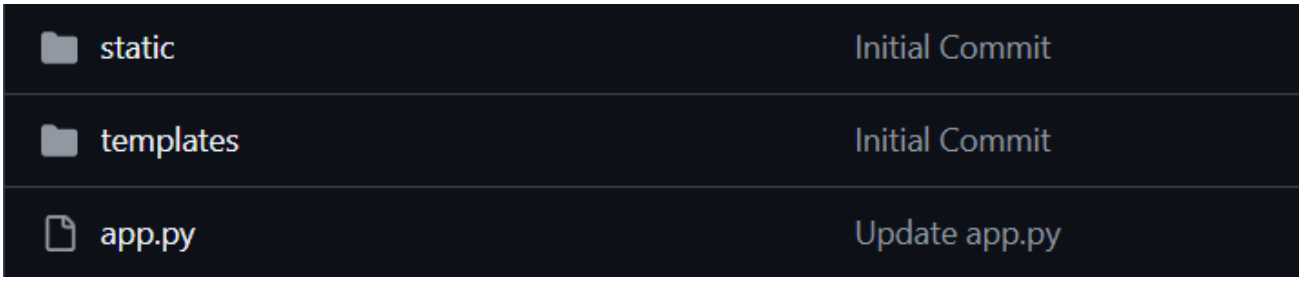


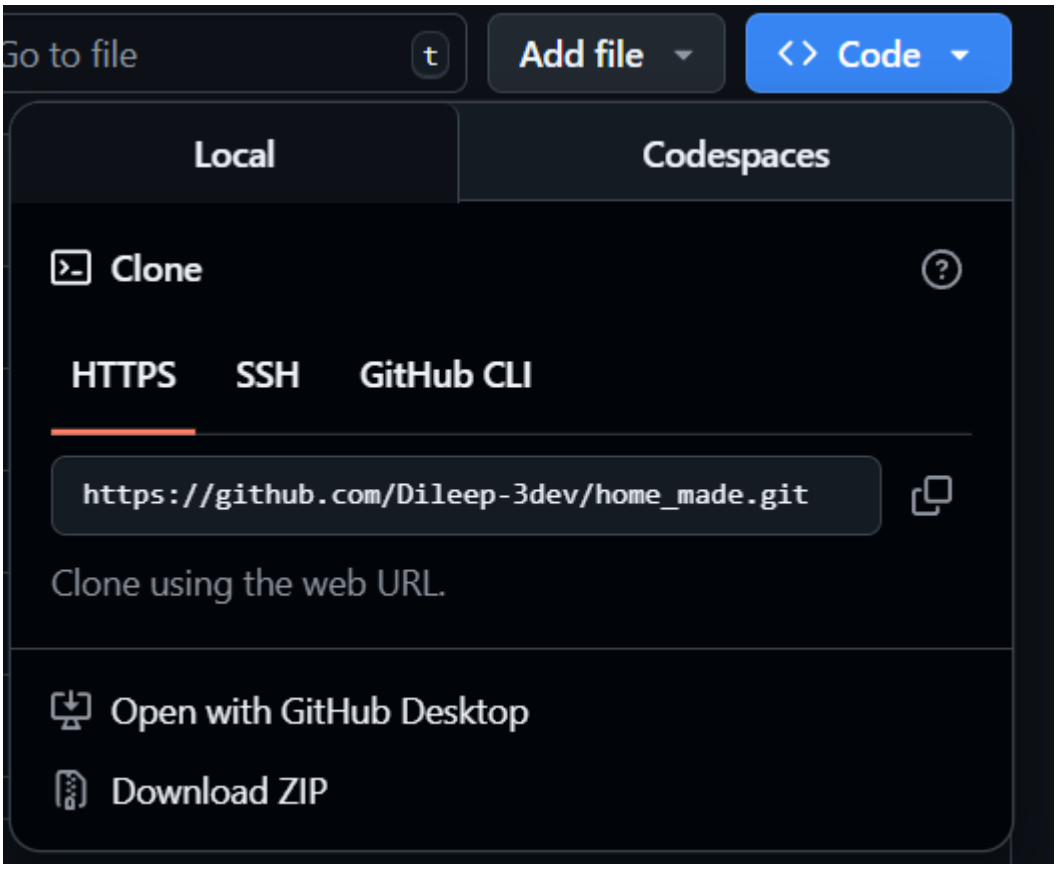


Milestone 6 : EC2 Instance Setup

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access, and configure your application or services to be publicly accessible.

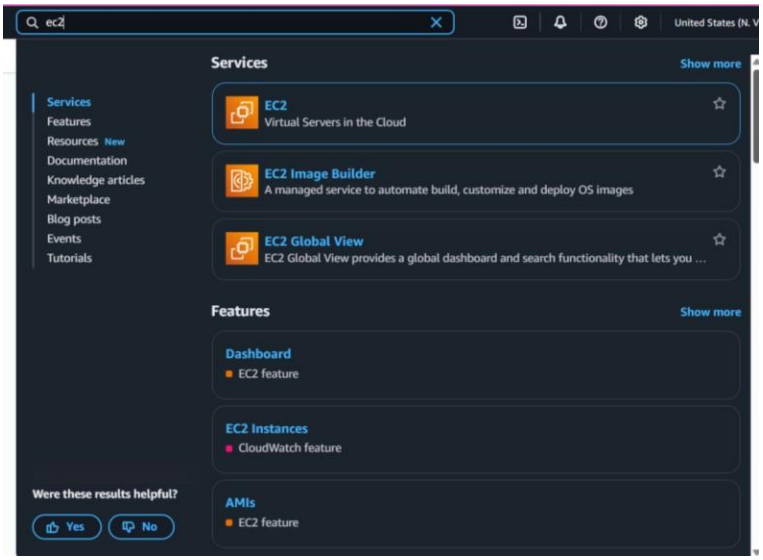
- Note: Load your Flask app and Html files into GitHub repository.



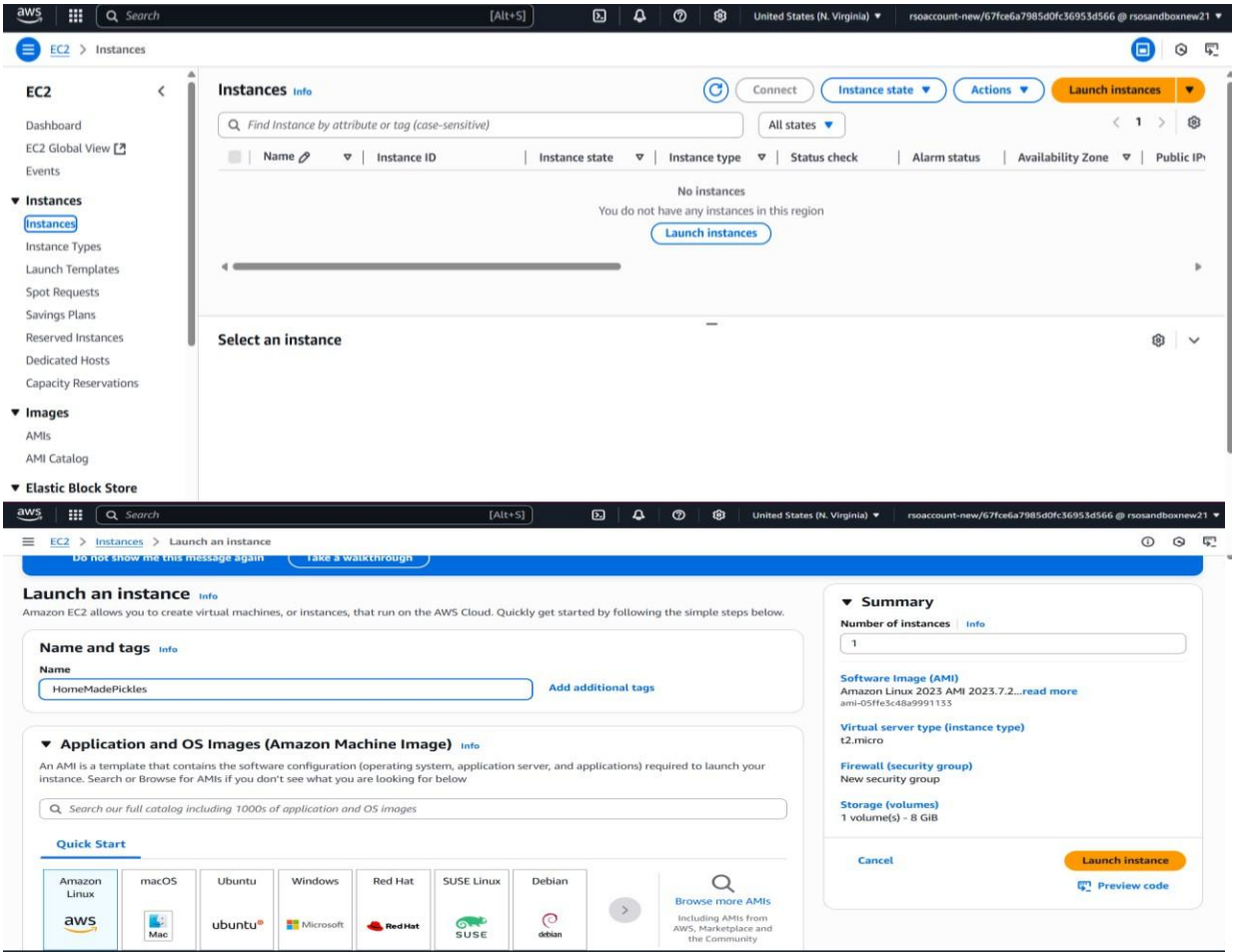


Launch an EC2 instance to host the Flask

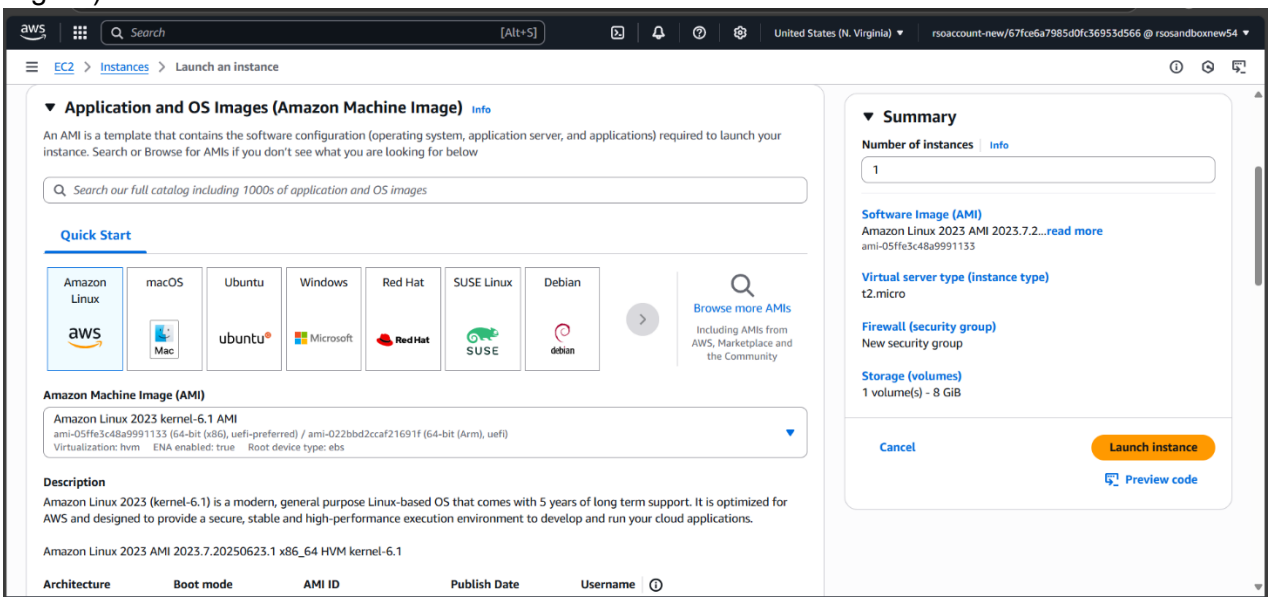
- Launch EC2 Instance
- In the AWS Console, navigate to EC2 and launch a new instance.



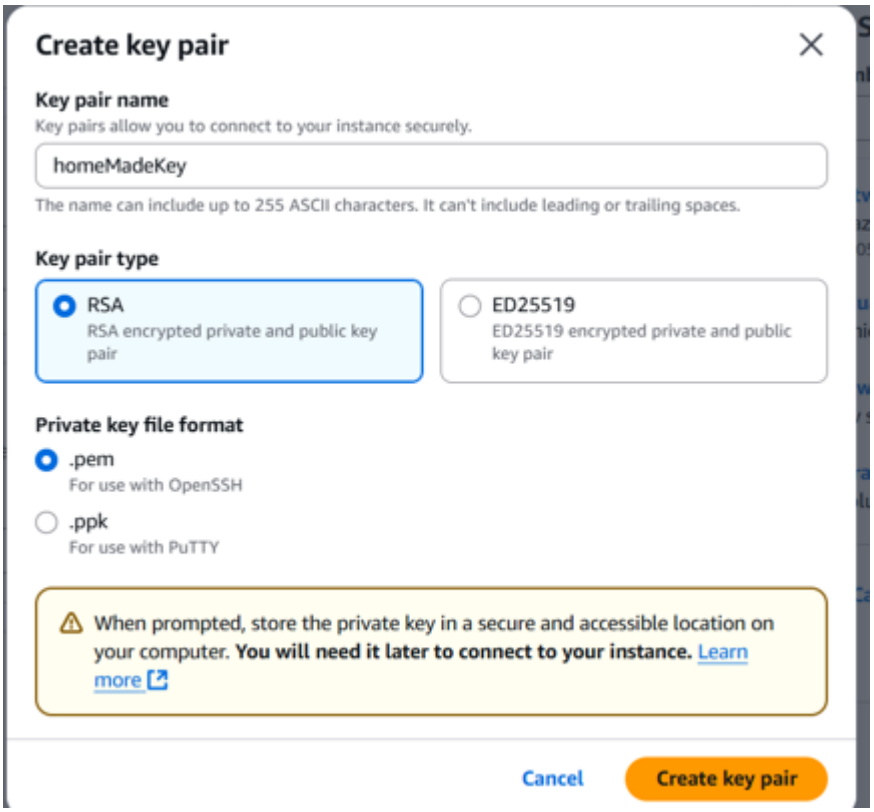
- Click on Launch instance to launch EC2 instance

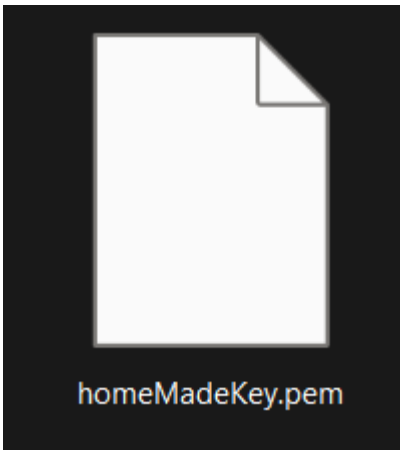


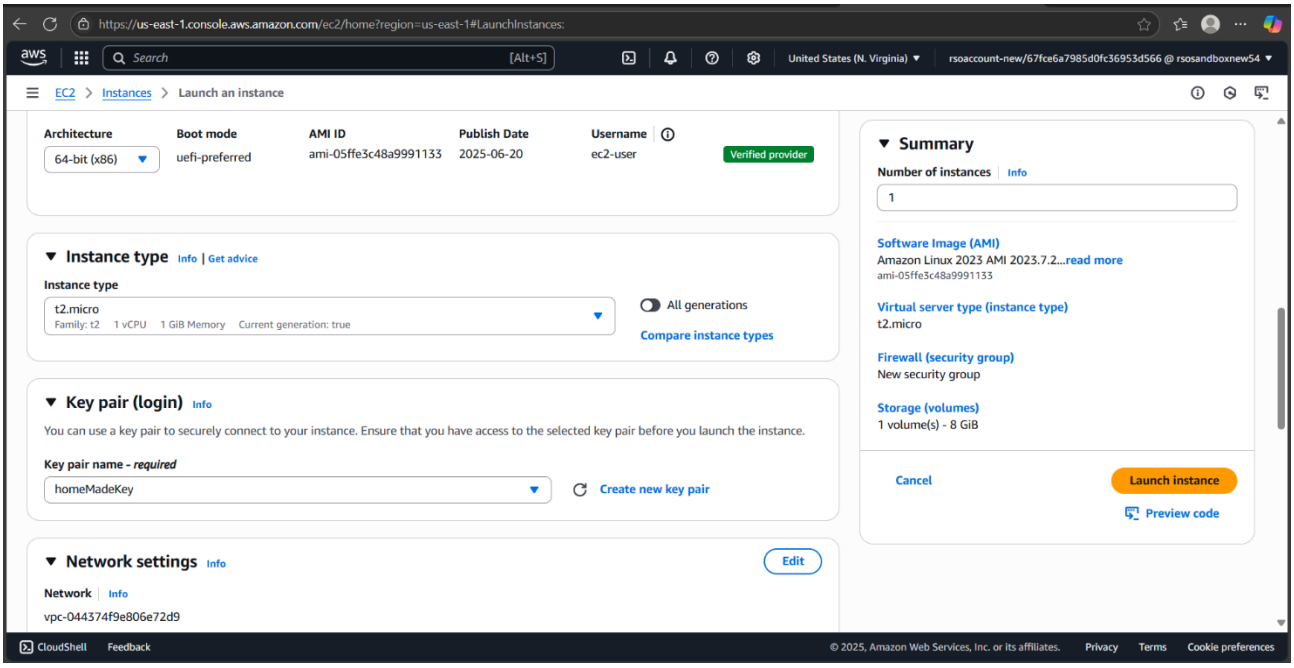
? Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



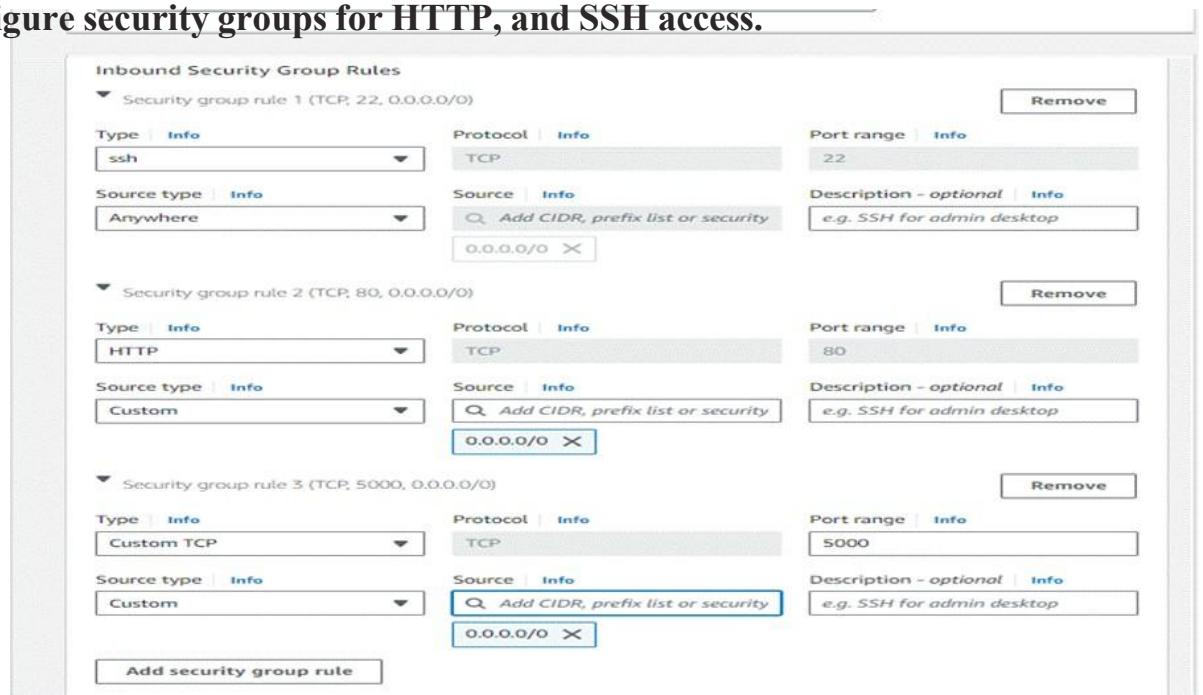
- Create and download the key pair for Server access.
-

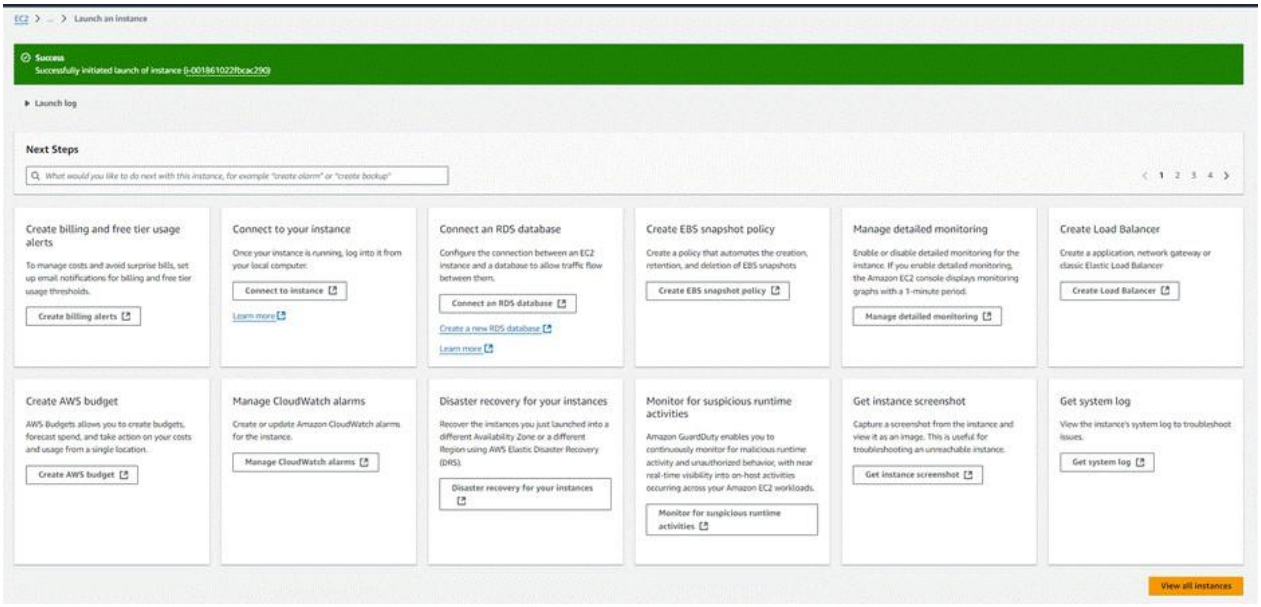




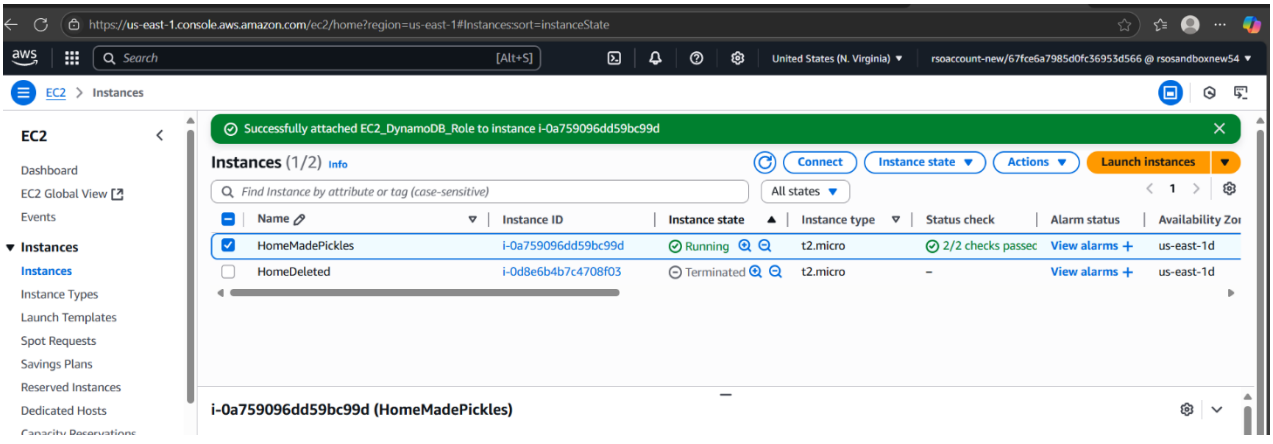


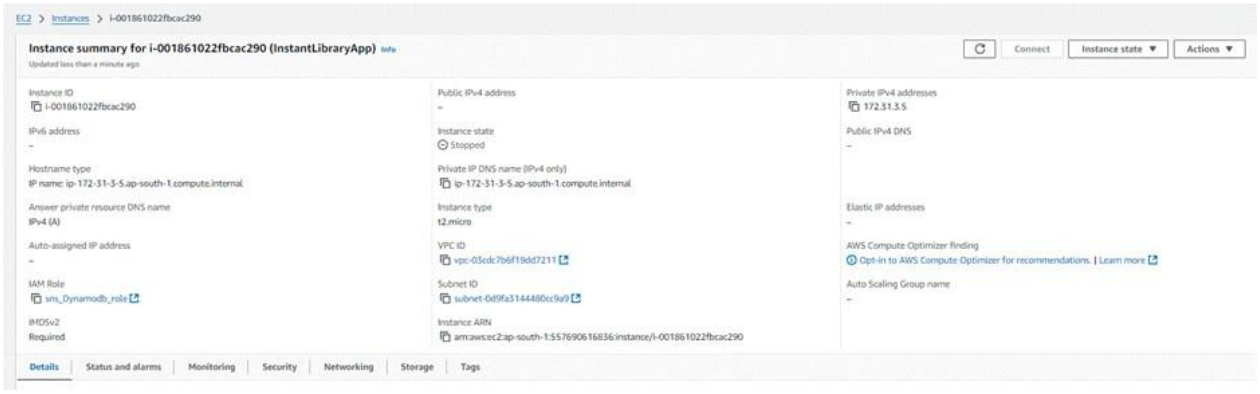
Configure security groups for HTTP, and SSH access.

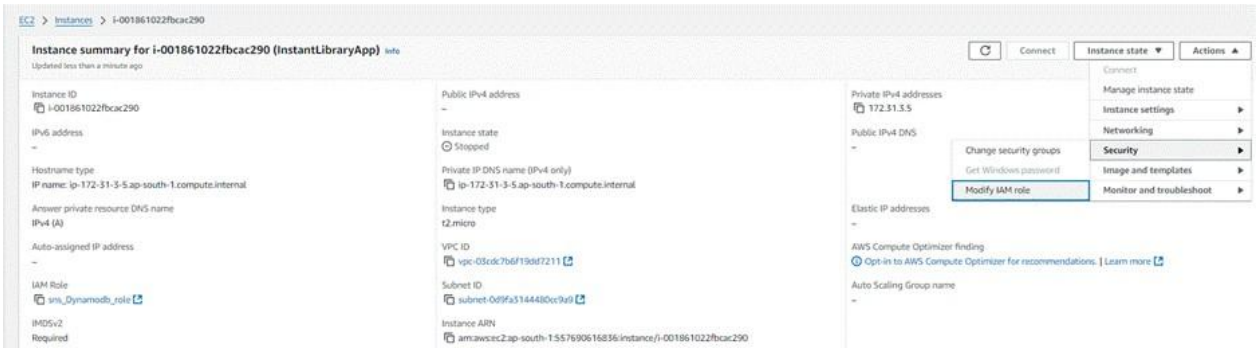


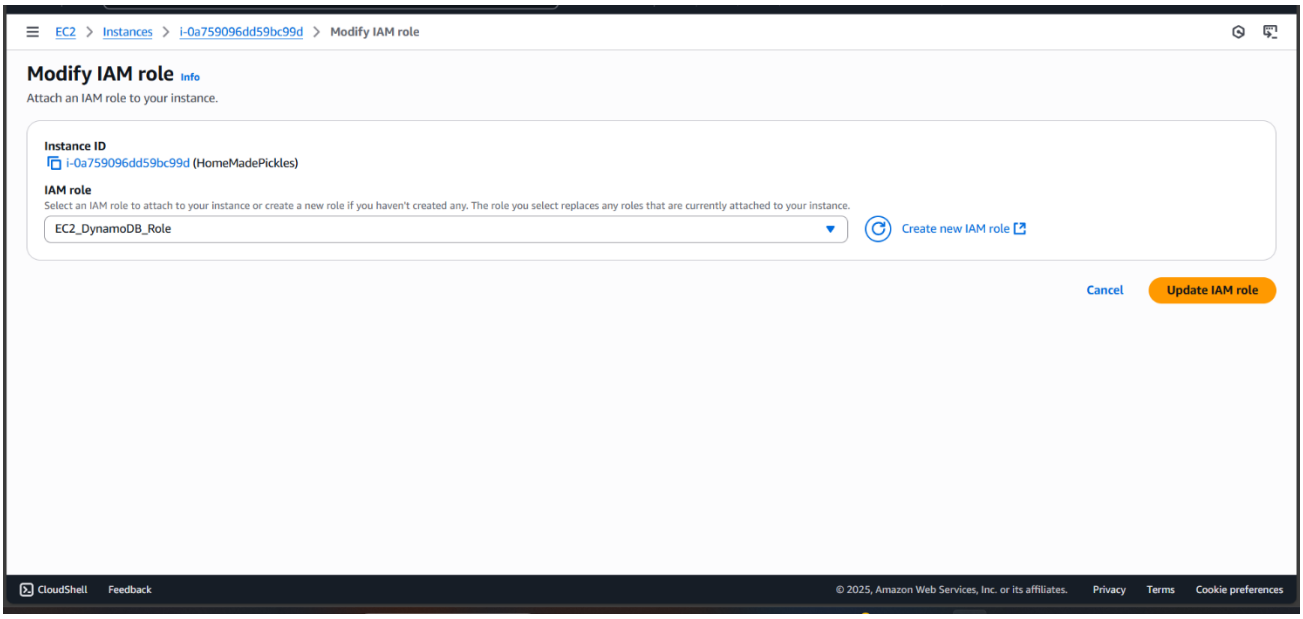


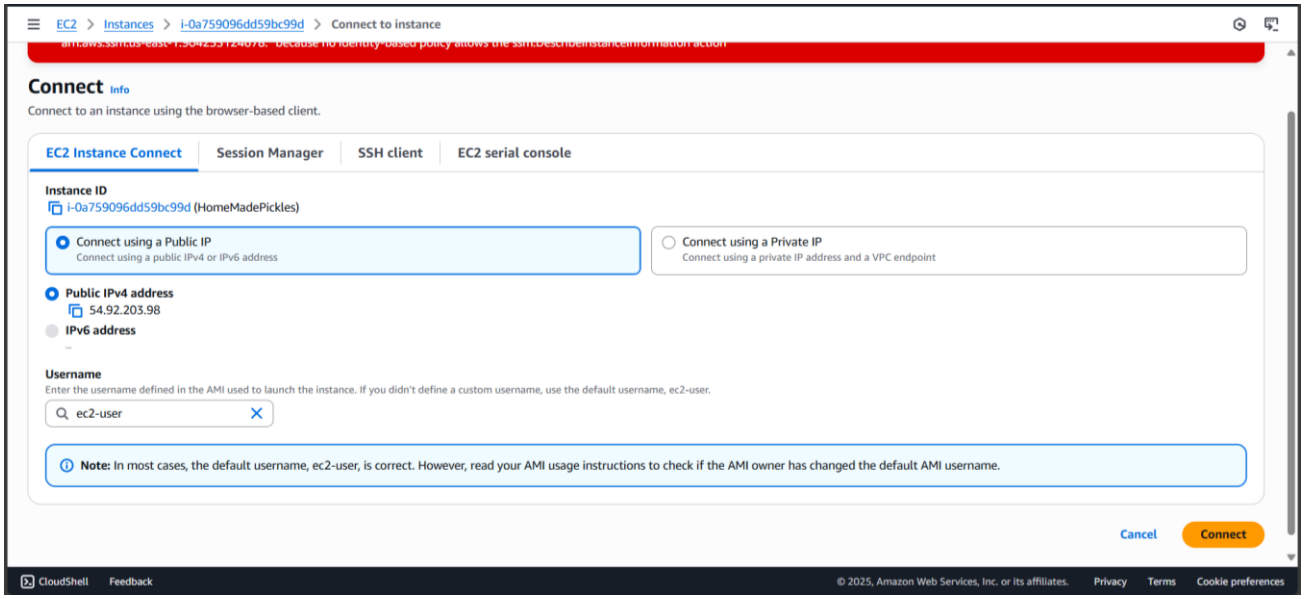
- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected,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

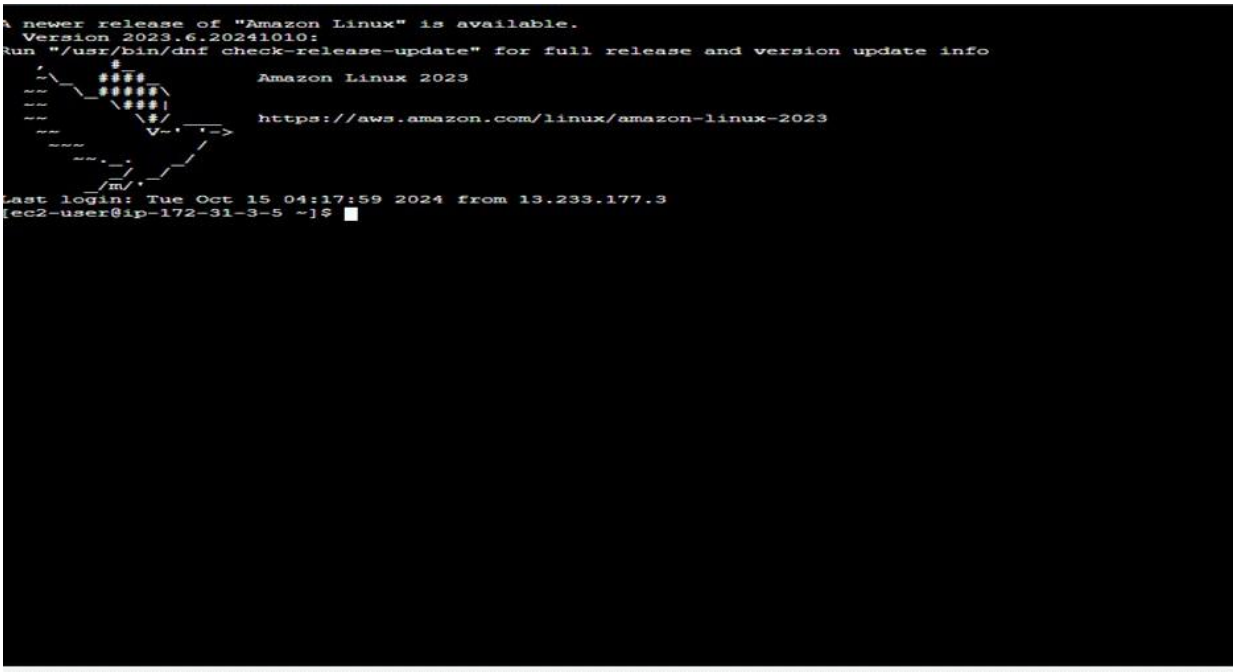


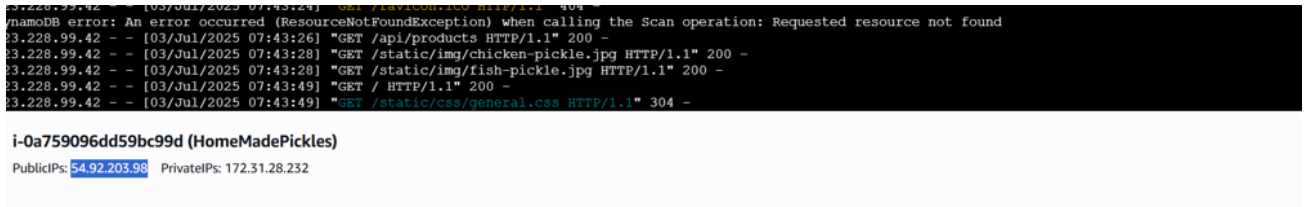

EC2 > Instances > i-001861022fbcac290
Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-001861022fbcac290
IPv6 address
-
Hostname type
IP name: ip-172-31-3-5.ap-south-1.compute.internal
Answer private resource DNS name
IPv4 (A)
Auto-assigned IP address
-
IAM Role
sns_Dynamodb_role
IMDSv2
Required
Public IPv4 address
-
Instance state
Stopped
Private IP DNS name (IPv4 only)
ip-172-31-3-5.ap-south-1.compute.internal
Instance type
t2.micro
VPC ID
vpc-03cdc7b6f19dd7211
Subnet ID
subnet-0d9fa3144480cc9a9
Instance ARN
arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290
Private IPv4 addresses
172.31.3.5
Public IPv4 DNS
-
Elastic IP addresses
-
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
-
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags


EC2 > Instances > i-001861022fbcac290
Instance summary for i-001861022fbcac290 (InstantLibraryApp) Info
Updated less than a minute ago
Connect
Instance state
Actions
Connect
Manage instance state
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
Instance ID
i-001861022fbcac290
IPv6 address
-
Hostname type
IP name: ip-172-31-3-5.ap-south-1.compute.internal
Answer private resource DNS name
IPv4 (A)
Auto-assigned IP address
-
IAM Role
sns_Dynamodb_role
IMDSv2
Required
Public IPv4 address
-
Instance state
Stopped
Private IP DNS name (IPv4 only)
ip-172-31-3-5.ap-south-1.compute.internal
Instance type
t2.micro
VPC ID
vpc-03cdc7b6f19dd7211
Subnet ID
subnet-0d9fa3144480cc9a9
Instance ARN
arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290
Private IPv4 addresses
172.31.3.5
Public IPv4 DNS
-
Elastic IP addresses
-
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
-


EC2 > Instances > i-0a759096dd59bc99d > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-0a759096dd59bc99d (HomeMadePickles)
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.
EC2_DynamoDB_Role
Create new IAM role
Cancel
Update IAM role
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


EC2 > Instances > i-0a759096dd59bc99d > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0a759096dd59bc99d (HomeMadePickles)
Connect using a Public IP
Connect using a public IPv4 or IPv6 address
Connect using a Private IP
Connect using a private IP address and a VPC endpoint
Public IPv4 address
54.92.203.98
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
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$


DynamoDB error: An error occurred (ResourceNotFoundException) when calling the Scan operation: Requested resource not found
23.228.99.42 - - [03/Jul/2025 07:43:26] "GET /api/products HTTP/1.1" 200 -
23.228.99.42 - - [03/Jul/2025 07:43:28] "GET /static/img/chicken-pickle.jpg HTTP/1.1" 200 -
23.228.99.42 - - [03/Jul/2025 07:43:28] "GET /static/img/fish-pickle.jpg HTTP/1.1" 200 -
23.228.99.42 - - [03/Jul/2025 07:43:49] "GET / HTTP/1.1" 200 -
23.228.99.42 - - [03/Jul/2025 07:43:49] "GET /static/css/general.css HTTP/1.1" 304 -
i-0a759096dd59bc99d (HomeMadePickles)
PublicIPs: 54.92.203.98 PrivateIPs: 172.31.28.232

Milestone 7 : Deployment on EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

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

Clone Your Flask Project from GitHub

Run: 'git clone https://github.com/Dileep-3dev/home_made.git'

- This will download your project to the EC2 instance.

  Here : 'git clone https://github.com/Dileep-3dev/home_made.git'

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

```
cd home_made
```

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
```

- This will download your project to the EC2 instance.

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

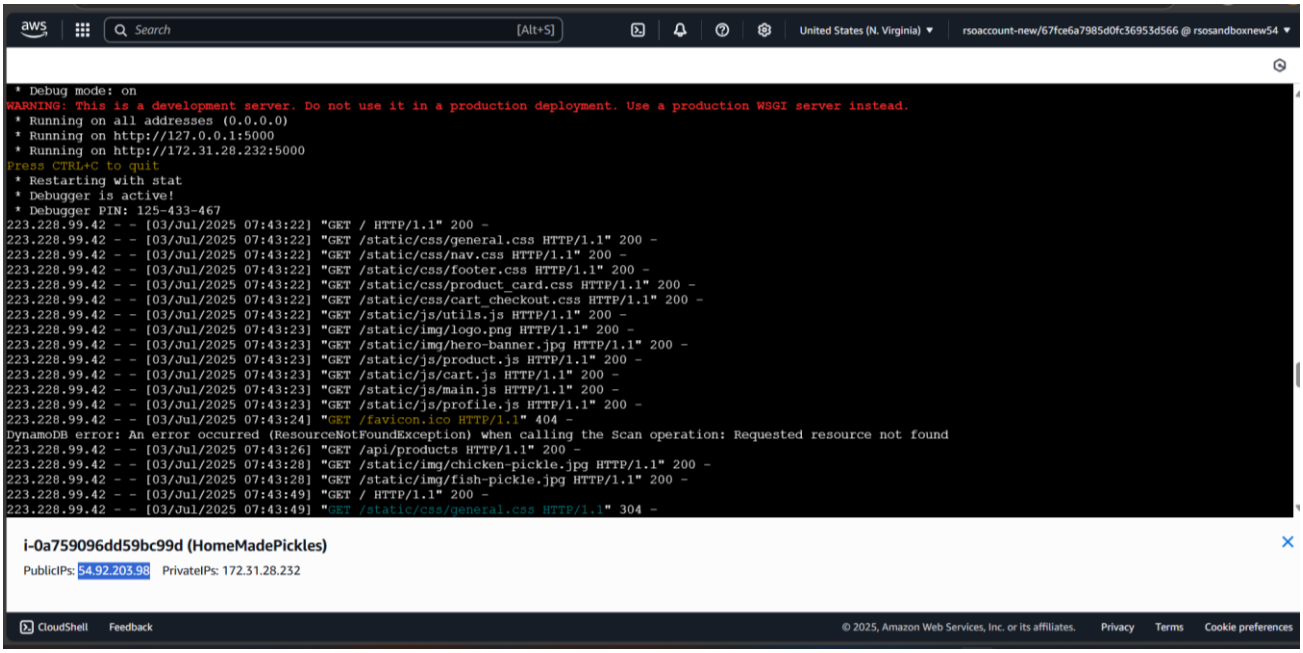
- cd Homemadepicklesandsnacks
- cd “Home Made Pickles1”

Create a Virtual Environment:

- python3 -m venv
- source venv/bin/activate
- sudo yum install python3 git
  sudo pip3 install flask boto3

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

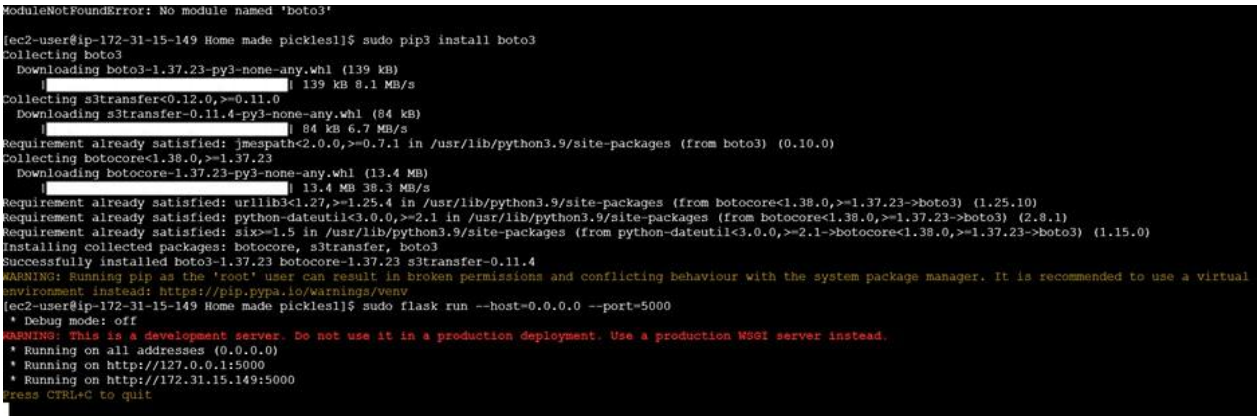
- Run the Flask Application
- sudo flask run --host=0.0.0.0 --port=5000



Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



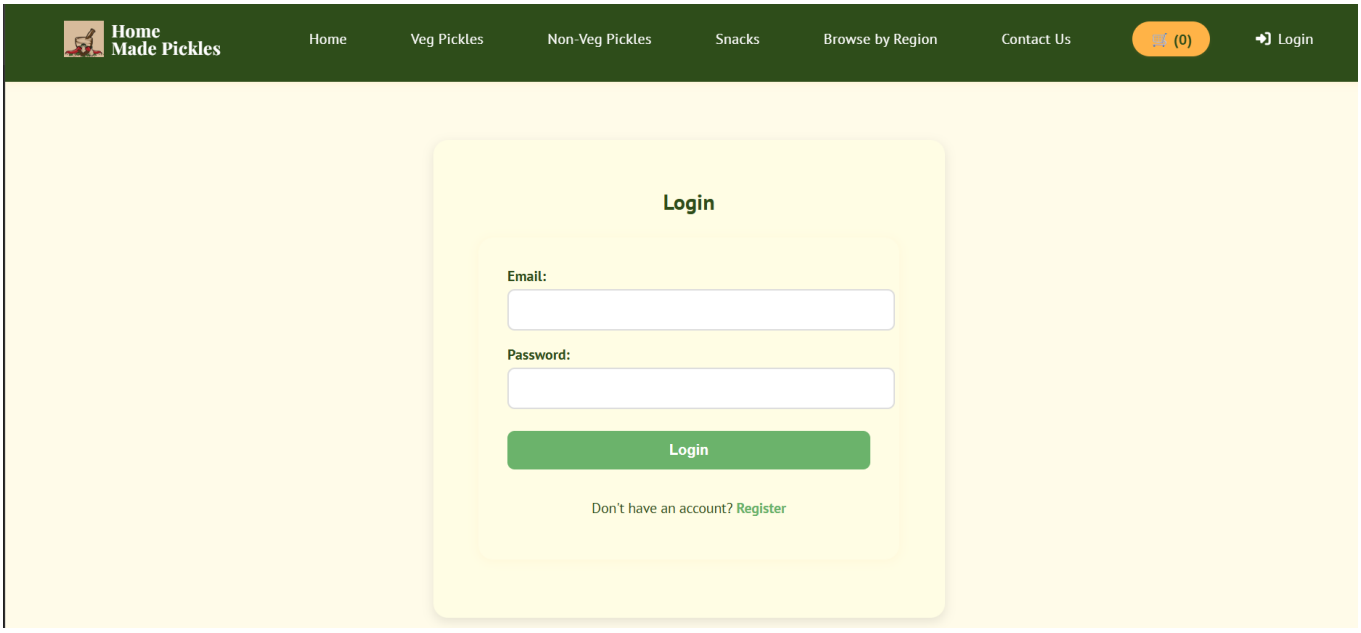
Acess the website through:

PublicIPs: http://54.92.203.98:5000

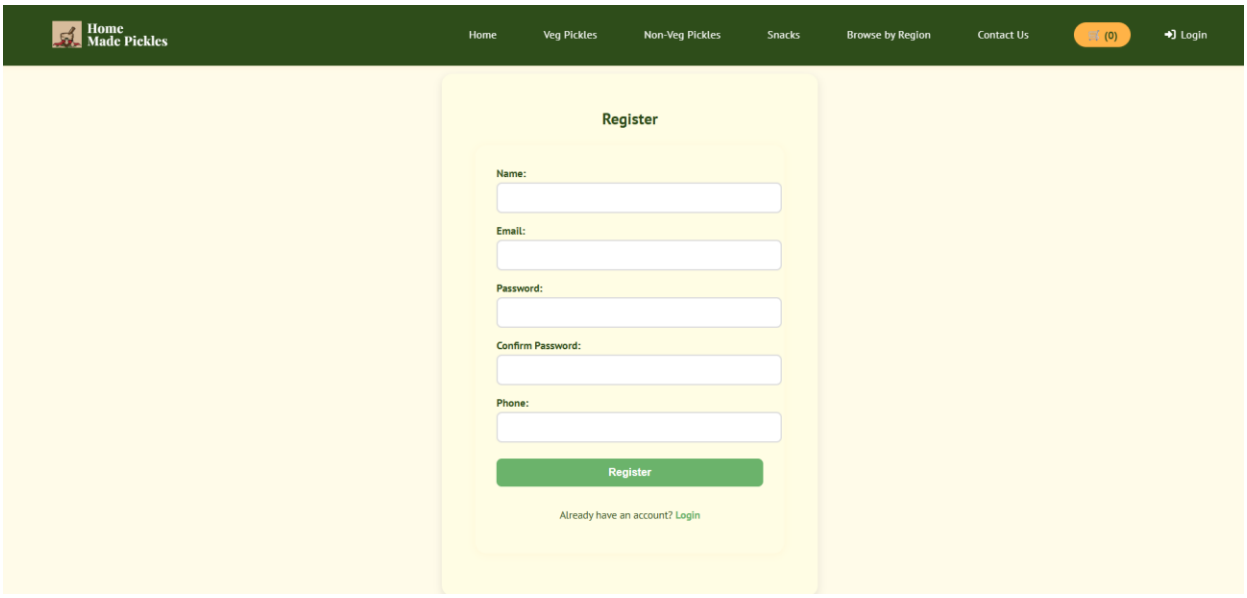
Milestone 8 : Testing and Deployment

Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

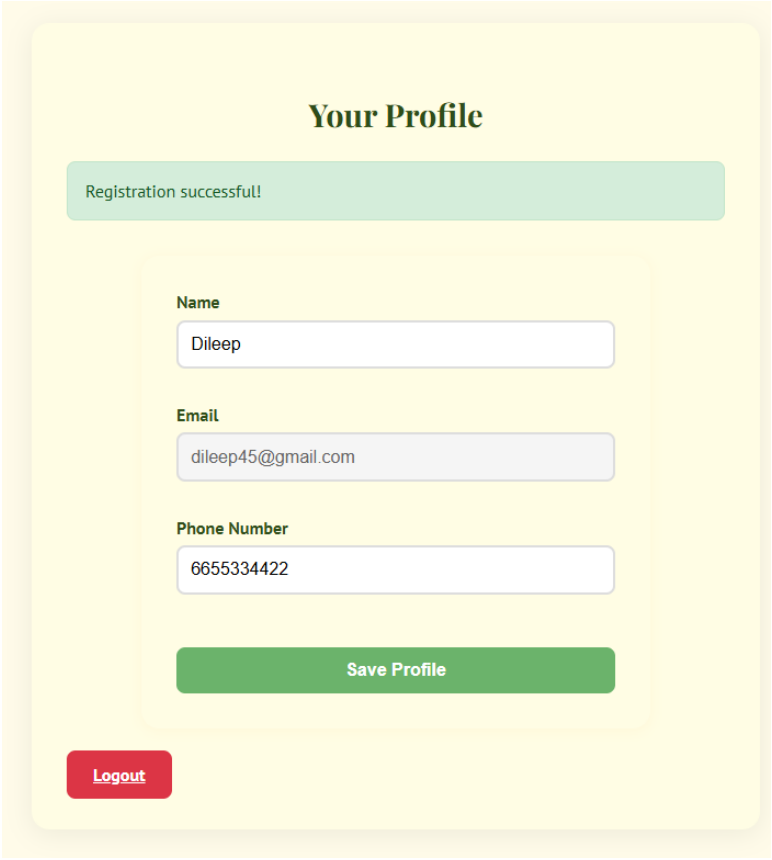
Login Page:



Register Page:

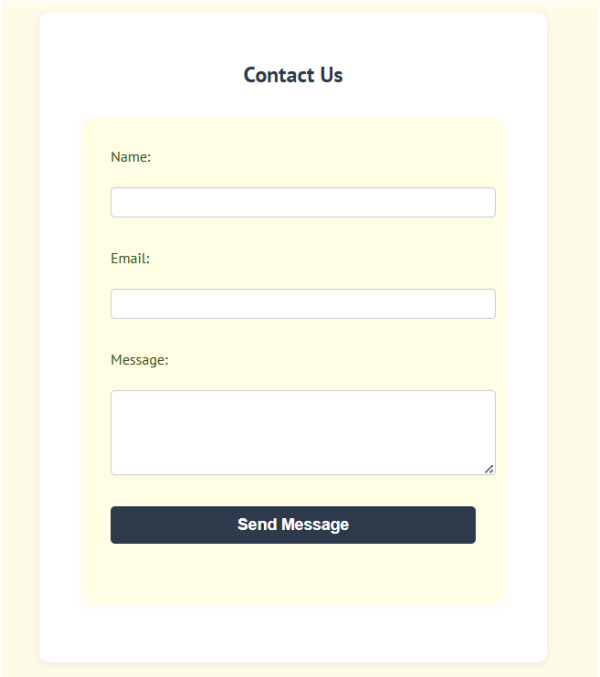


Profile page :

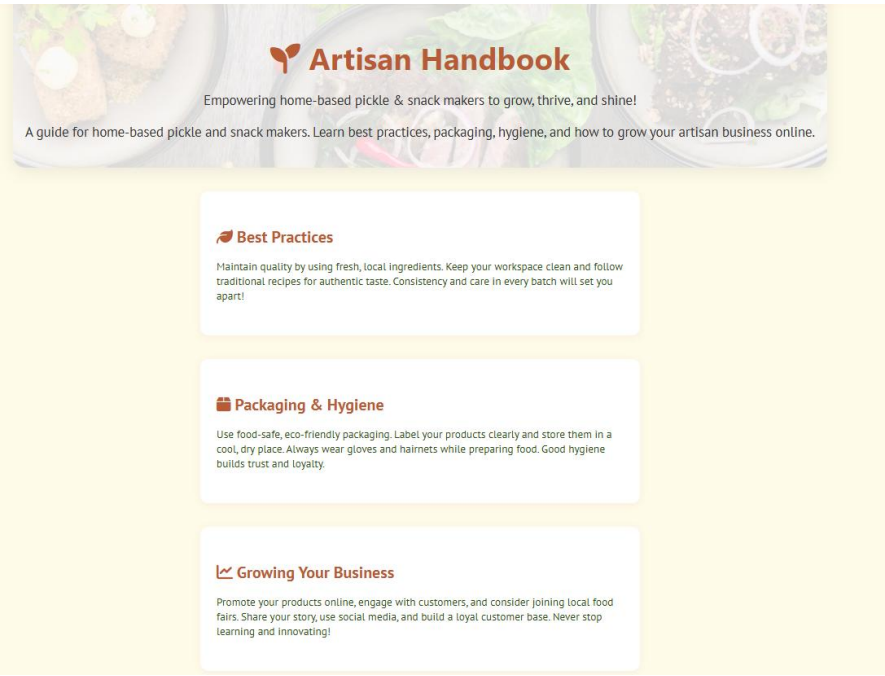


Home page:

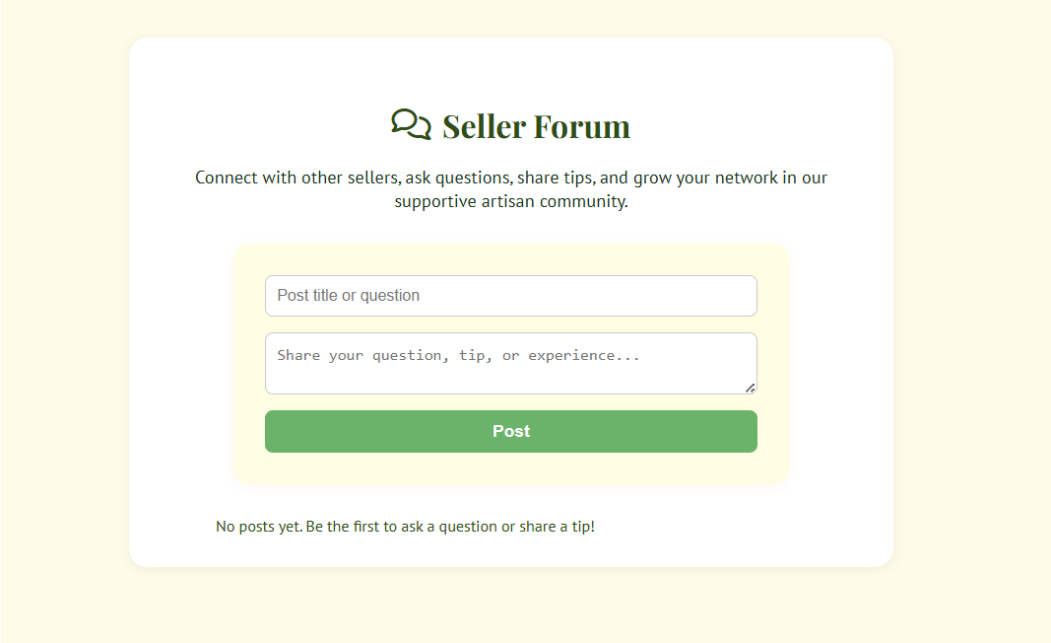
Contact Us page:



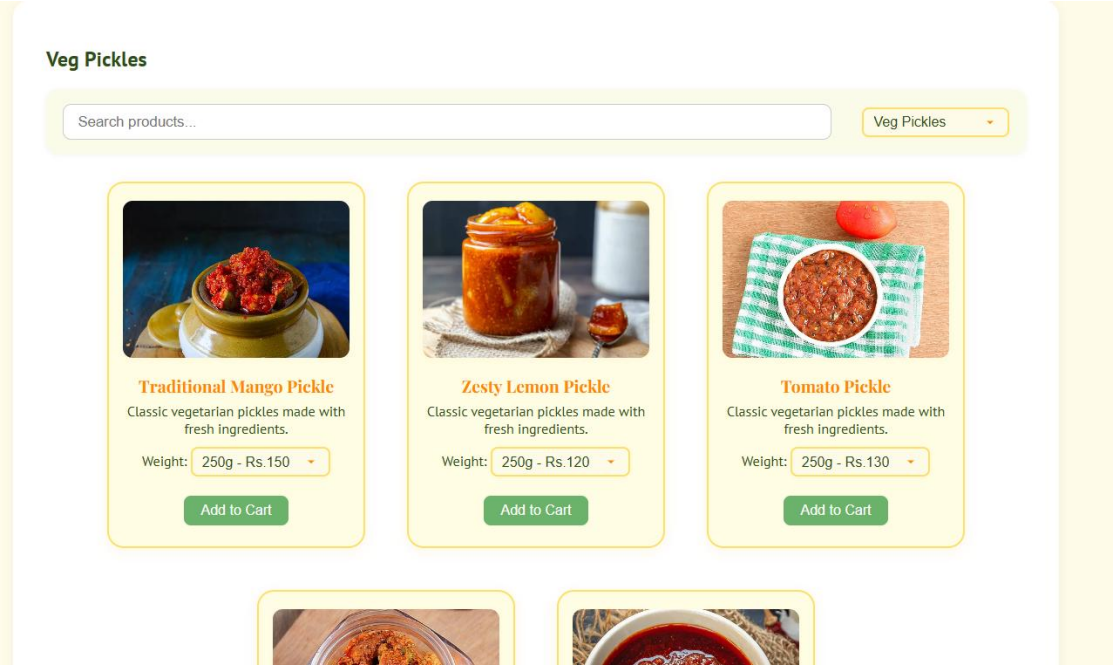
Artisan Handbook Page:



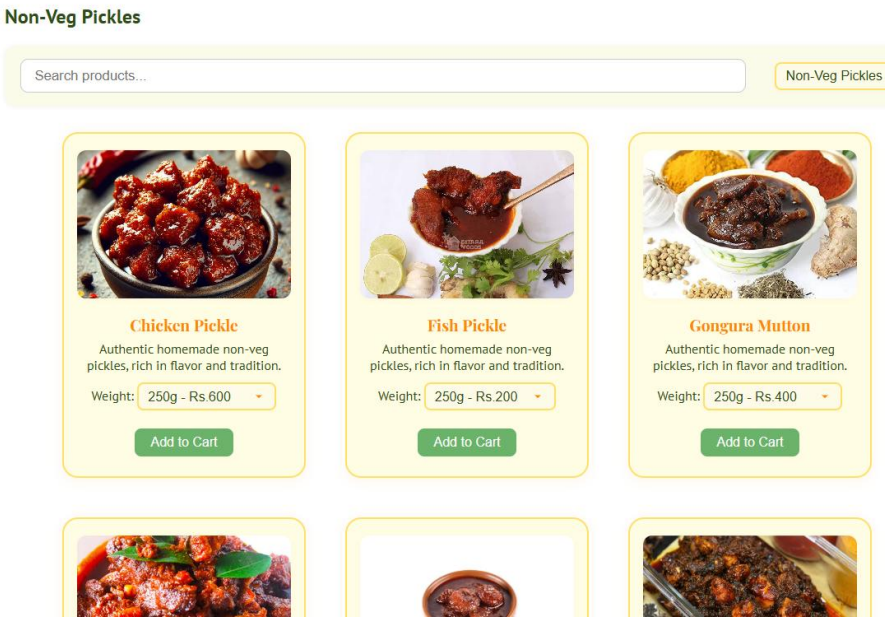
Seller forum page:



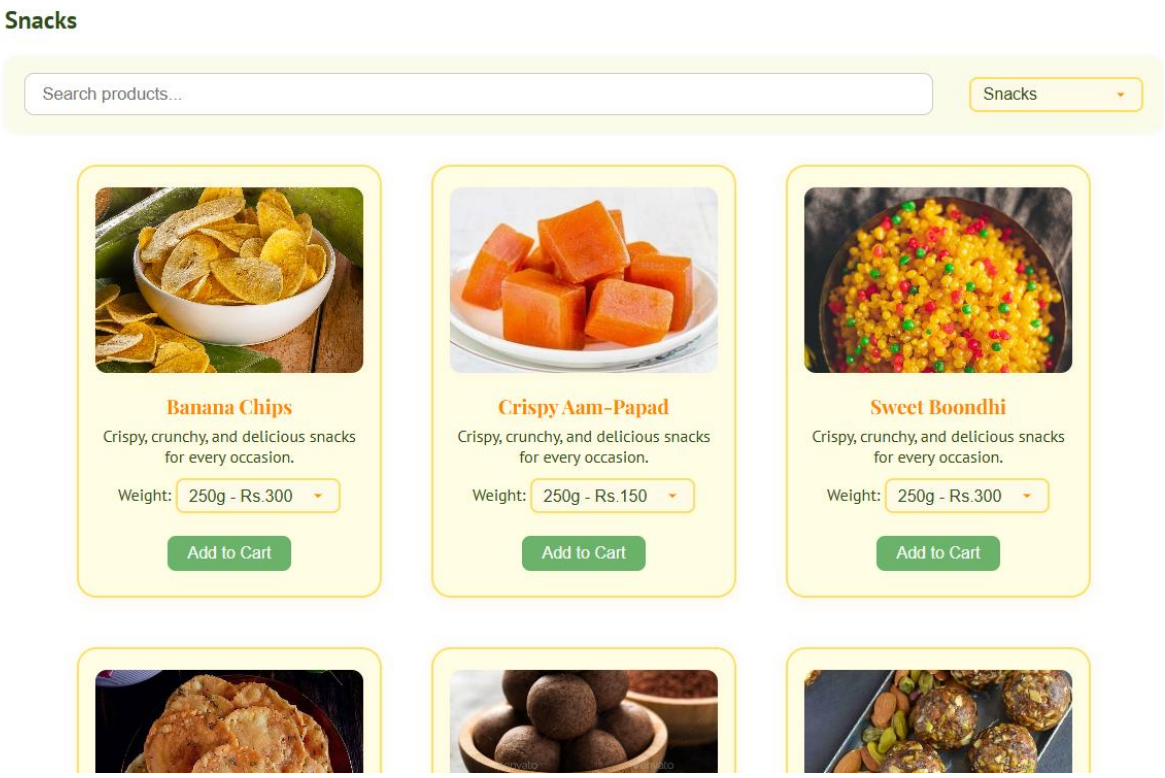
Veg Pickles Page :



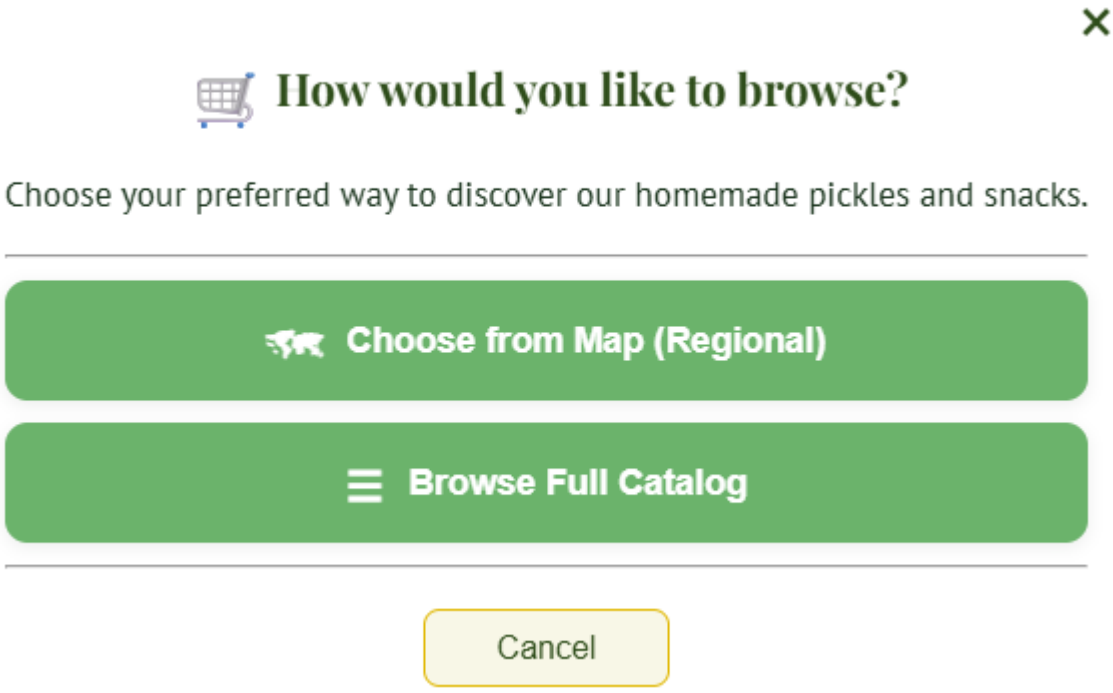
Non-Veg Pickles page:



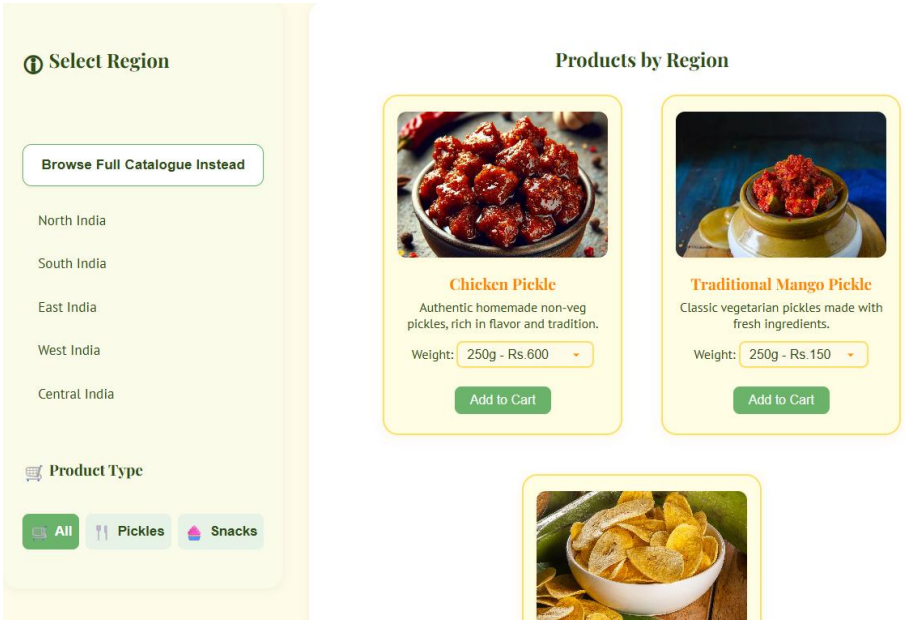
Snacks Page:



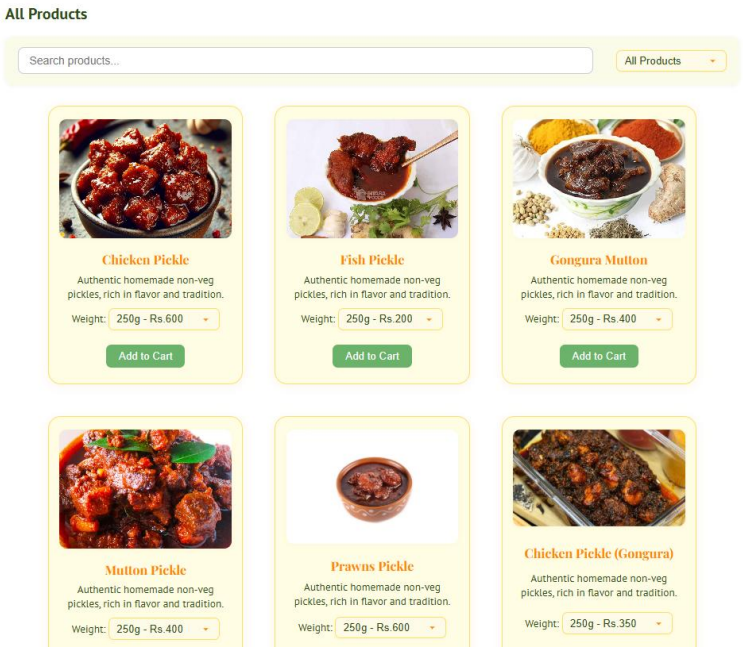
Category Page :



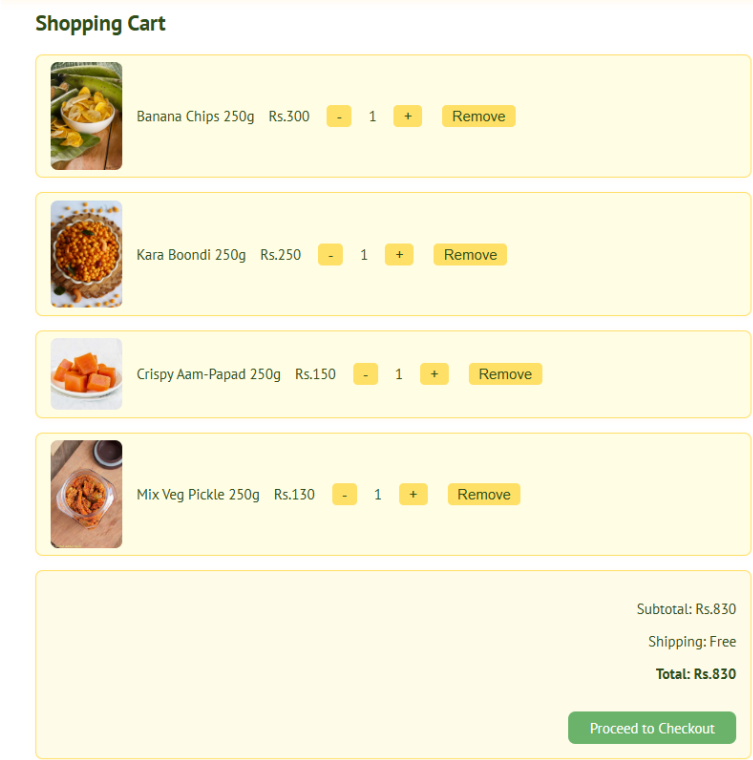
Regional Page:



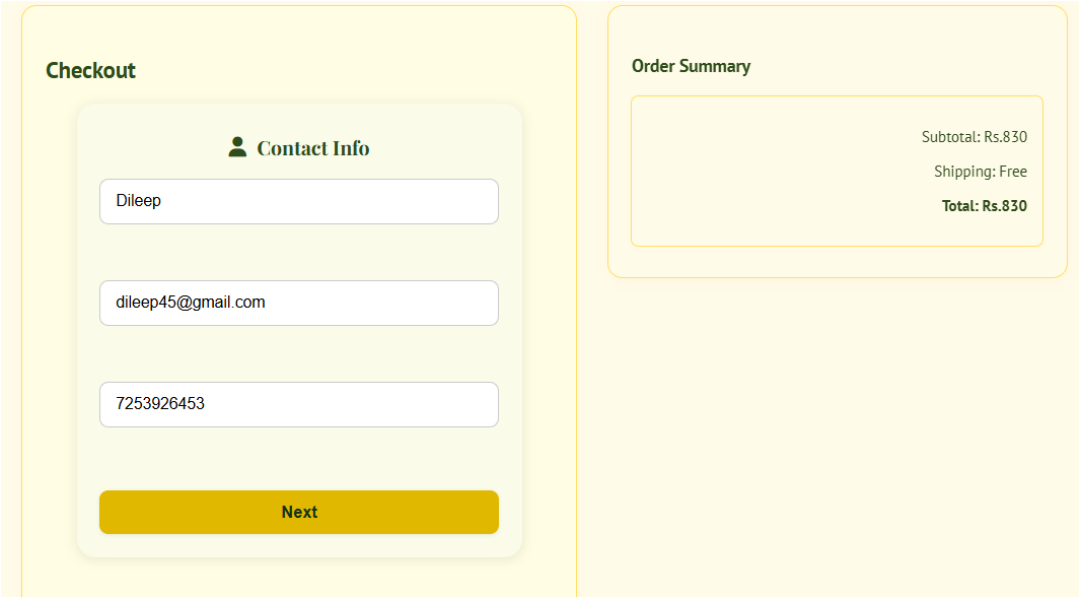
Products Page :



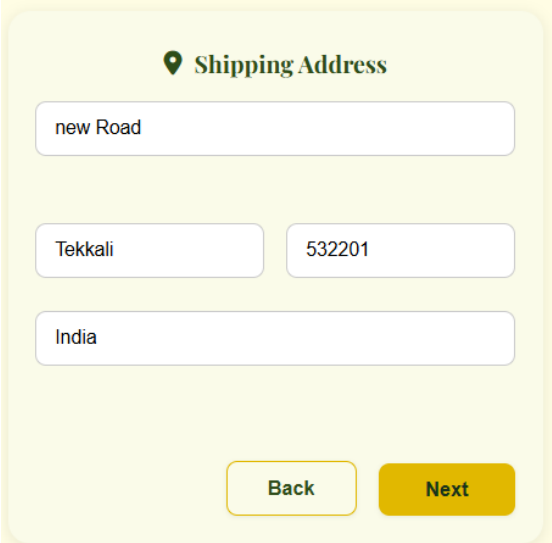
Cart Page:

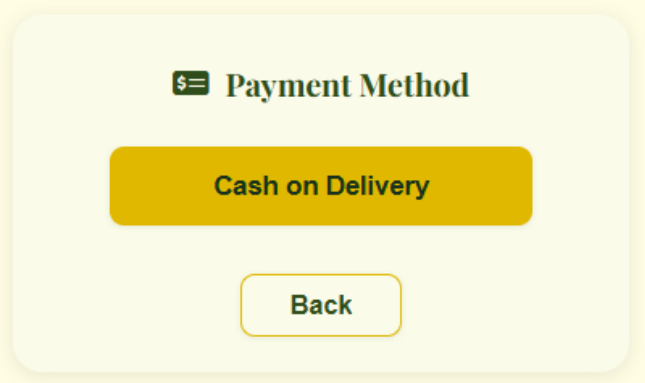


Check-Out Page:

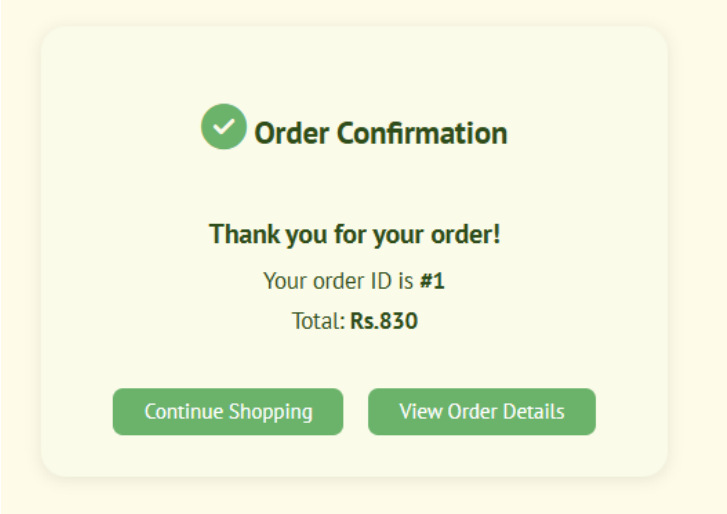


Address Page:

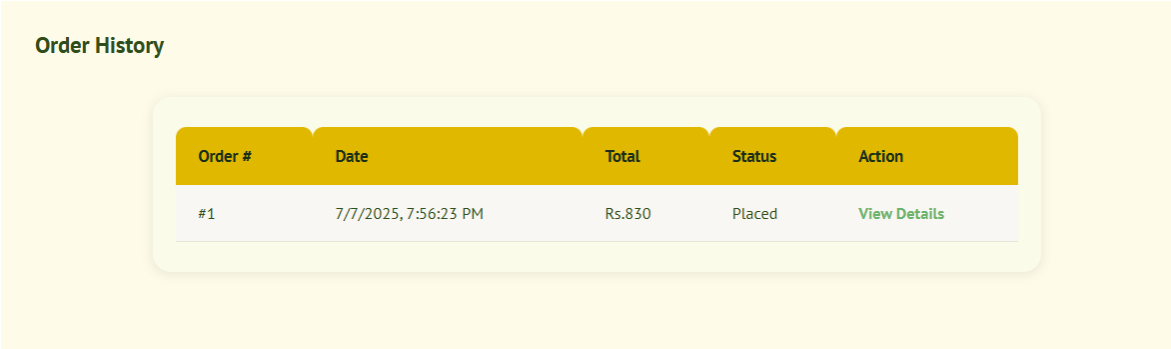




Confirmation Page:



Order History page:



Exit:

Functional testing to verify the Project Conclusion

This platform has been thoughtfully created to connect food enthusiasts with authentic, handcrafted pickles and snacks from local artisans. Using Flask as its foundation, the website provides a seamless shopping experience that makes discovering and ordering traditional recipes simple and enjoyable.

Key Features: The platform offers an intuitive browsing system with categories for vegetarian pickles, non-vegetarian pickles, and traditional snacks. Customers can choose from multiple weight options (250g to 1kg), enjoy secure user accounts, and experience smooth cart management with real-time updates. The regional discovery feature helps users explore local specialties while maintaining a mobile-friendly design that works across all devices.

Supporting Artisans and Heritage: By bringing these time-honored recipes online, the platform bridges the gap between small-scale producers and customers seeking preservative-free, authentic flavours. The cloud-ready architecture ensures reliable service as the community grows, while automated email notifications and efficient order management keep everything running smoothly.

Impact: This project demonstrates how modern technology can preserve and share culinary traditions, making it easier for people to access genuine homemade flavours. As the platform evolves, it continues to serve as a trusted marketplace where every jar represents the care and quality of traditional home cooking.

In essence, this website transforms how authentic pickles and snacks reach customers, ensuring that traditional flavours find their way to modern kitchens with the same love and quality they've always carried.