Homemade Pickles & Snacks: Taste The Best

Project Description:

This project is a user-friendly website designed to showcase and sell a variety of homemade pickles and snacks. It features rich visuals, engaging descriptions, and an easy-to-navigate layout that highlights the authenticity and flavour of each product. Customers can explore product categories like spicy pickles, sweet preserves, and crunchy munchies, all made with traditional recipes. The site includes a secure shopping cart system and responsive design for mobile accessibility. A blog section shares homemade tips, recipe stories, and cultural food traditions to deepen customer connection. Users can rate products, leave reviews, and sign up for seasonal bundle offers. SEO-optimized content boosts discoverability across search engines. It's a Flavourful blend of heritage, tech, and convenience. If you're planning to build or document this site further, I can help structure sections like product listings, customer experience features, or a compelling homepage narrative.

Scenario 1: Efficient Order Placement by Customers

In the "Flavour Cart" homemade pickles and snacks website, AWS EC2 powers a reliable and scalable infrastructure, allowing multiple customers to browse and order products simultaneously. For example, a user visits the site, explores the pickles section, and places an order for a mango pickle jar. Flask handles the backend workflow, processing user selections, updating inventory, and confirming orders in real time. The cloud-based architecture ensures a seamless and fast shopping experience, even during peak demand like festivals or seasonal sales.

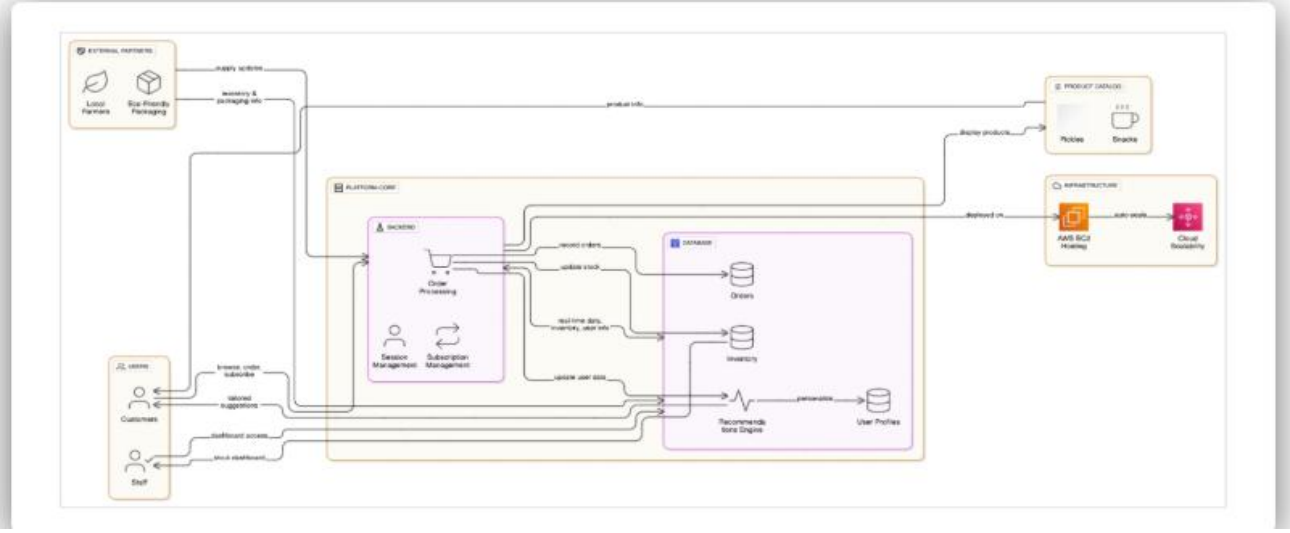
Scenario 2: Seamless Order Notifications for Kitchen Staff and Customers

When an order is placed, the system uses AWS SNS to instantly notify both the customer and the in-house kitchen team. Flask captures the order details and triggers an SNS email or SMS alert: the customer receives a confirmation with order tracking info, while the kitchen staff is alerted with the order details to initiate preparation. All order information is securely stored in AWS DynamoDB, ensuring timely fulfilment and status monitoring.

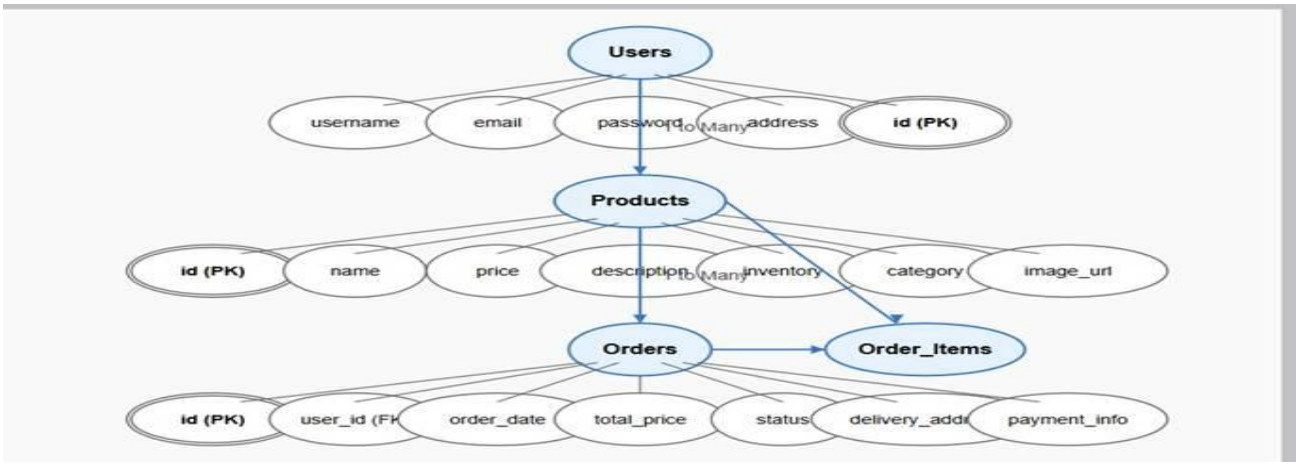
Scenario 3: Real-Time Inventory and Product Availability

The website allows users to explore the entire product range—spicy pickles, sweet treats, crunchy snacks—with up-to-date availability status. Flask fetches real-time data from DynamoDB, showing users whether an item is in stock or on backorder. Thanks to EC2, the site delivers a consistently smooth and responsive experience, even when accessed by many users across mobile or desktop platforms, ensuring customers always have access to the latest listings.

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. **VS Code Installation**: VISUALSTUDIO.COM
6. **Git Version Control**: Git Documentation

Project Work Flow:

1. AWS Account Setup and Login

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

Activity 1.2: Log in to the AWS Management Console.

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes 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

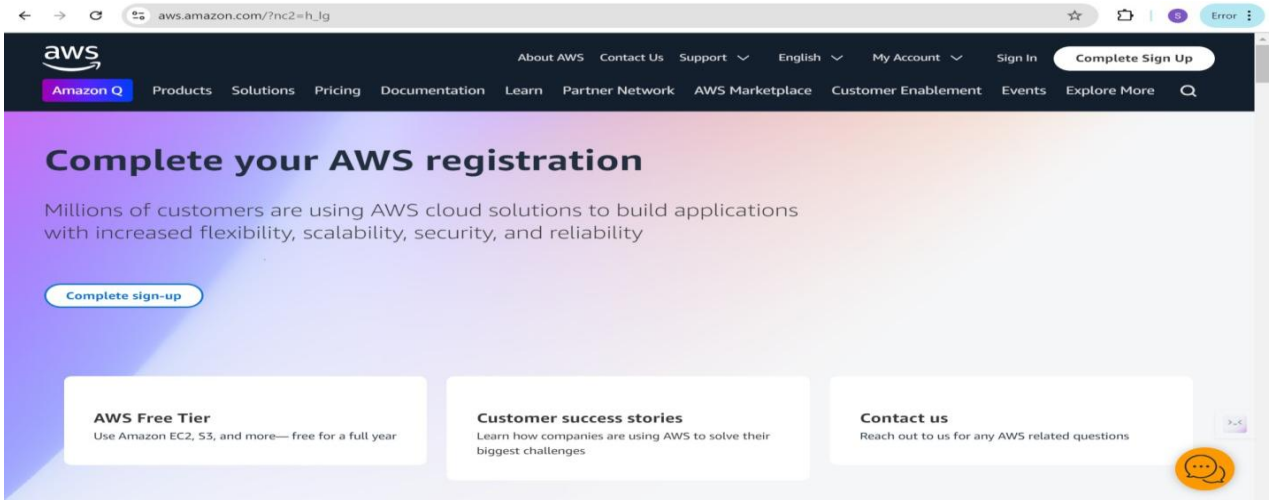
Activity 7.2: Run the Flask App

8. Testing and Deployment

Activity 8.1:Conduct functional testing to verify user signup, login, orders and notifications.

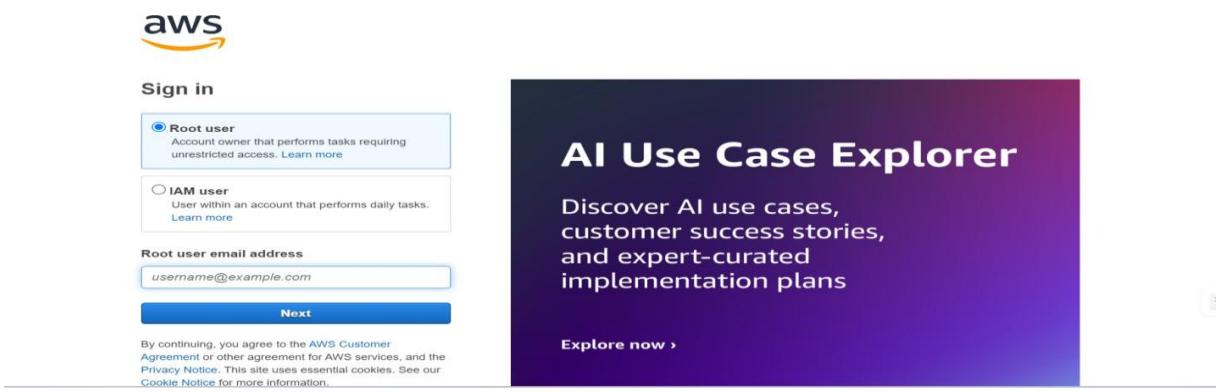
Milestone 1: AWS Account Setup and Login

- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.

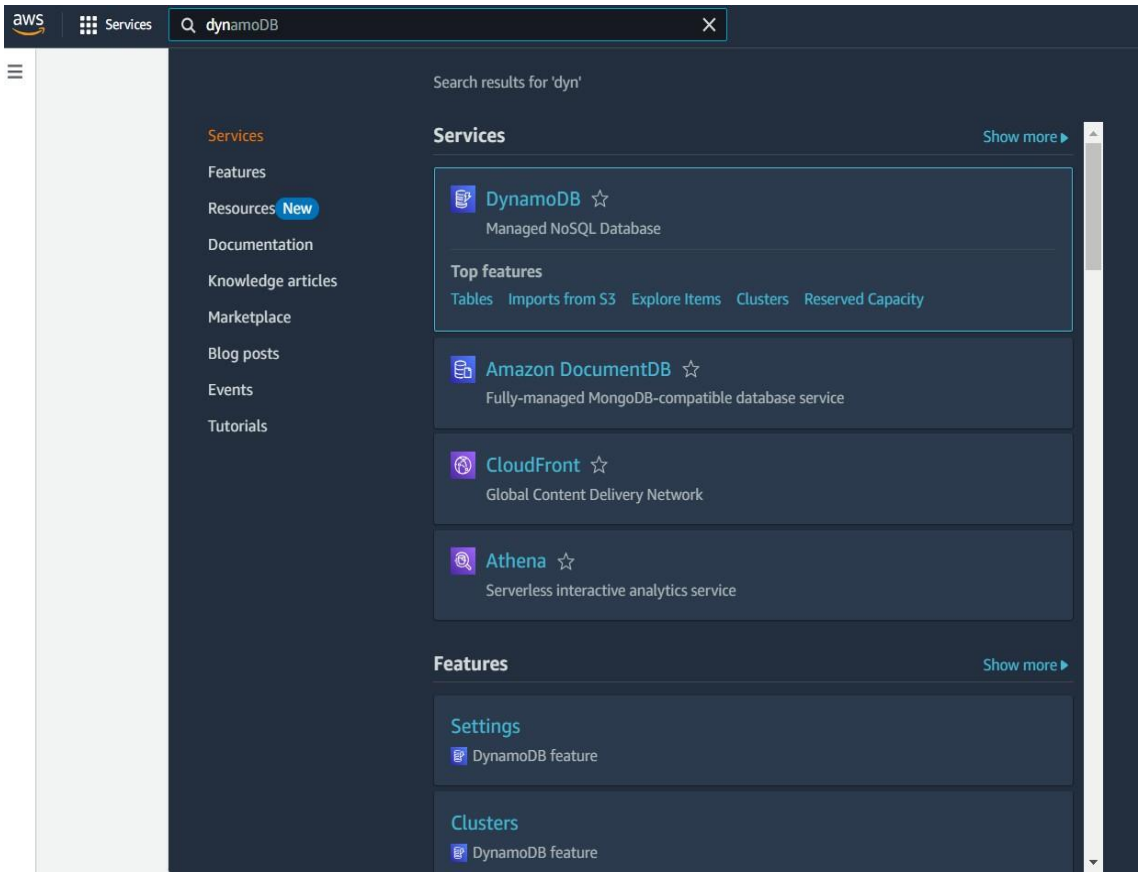


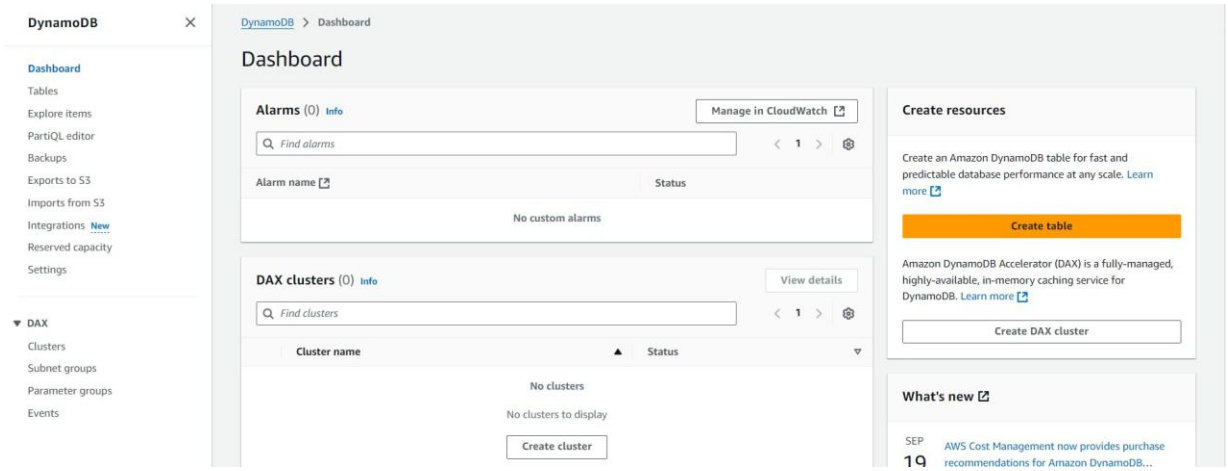
- **Activity 1.2: Log in to the AWS Management Console**

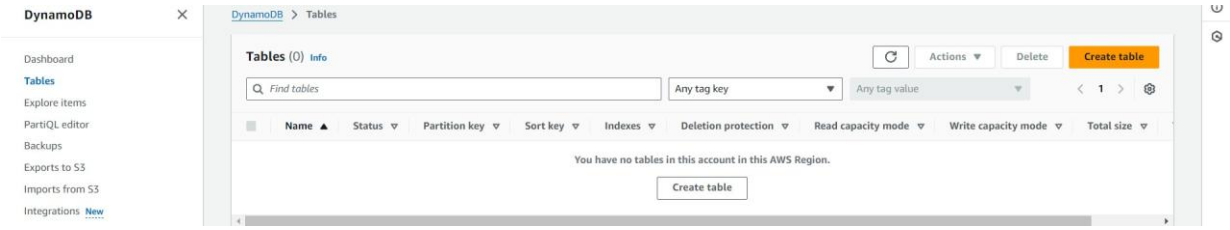
  - After setting up your account, log in to the AWS Management Console.



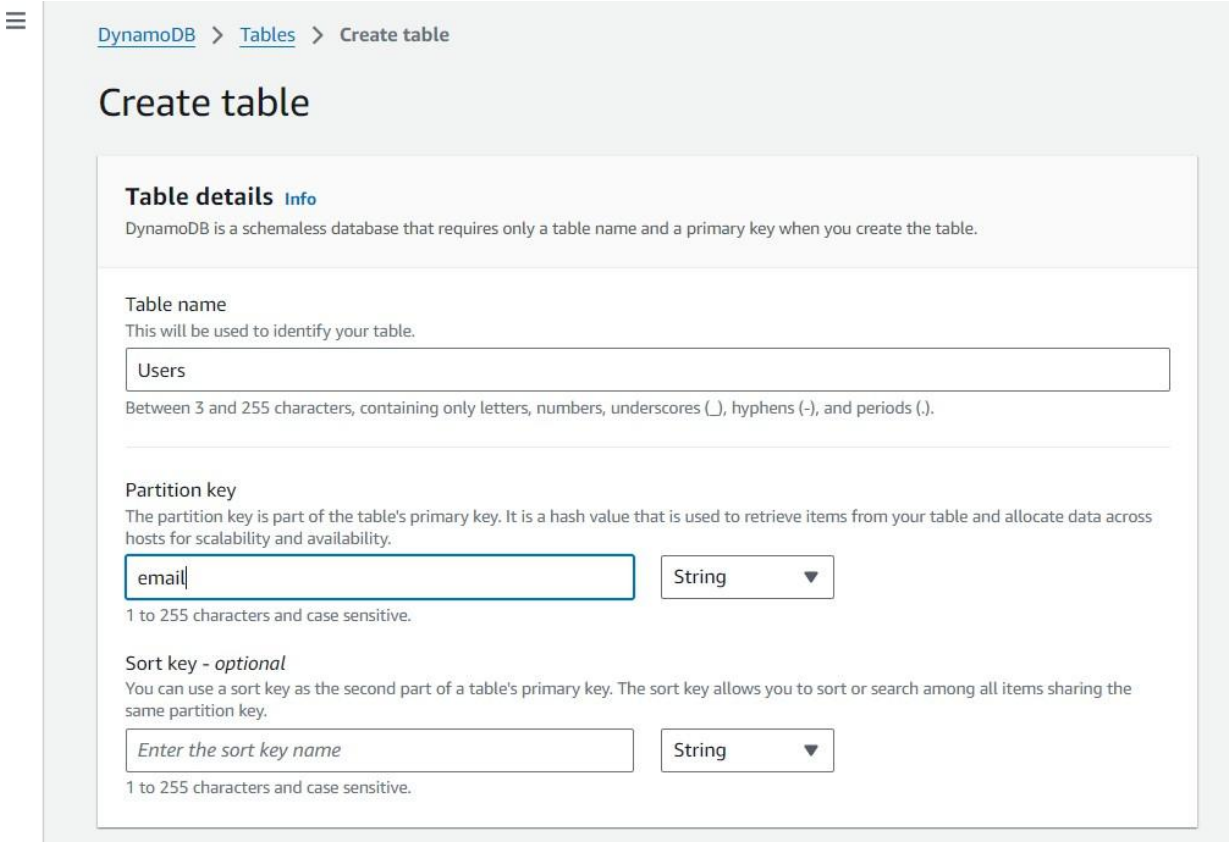
Milestone 2: DynamoDB Database Creation and Setup

- **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 Signup details and pickle requests.**
    - Create Users table with partition key “Email” with type String and click on create tables.



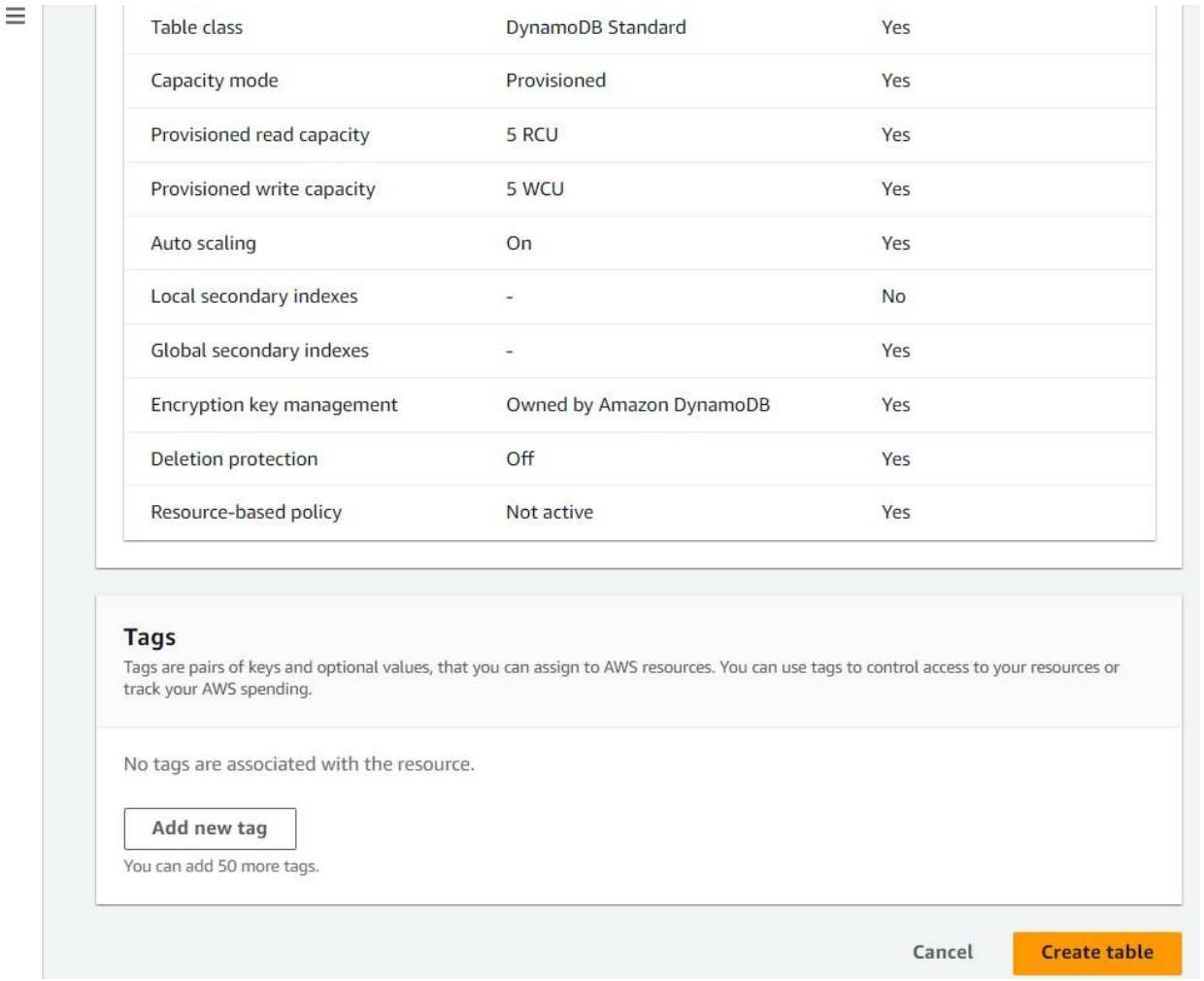
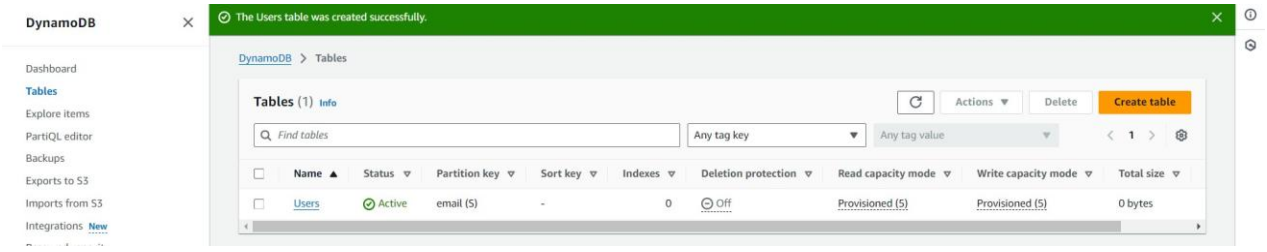


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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



- Follow the same steps to create a requests table with Email as the primary key for pickles order, contact requests data.

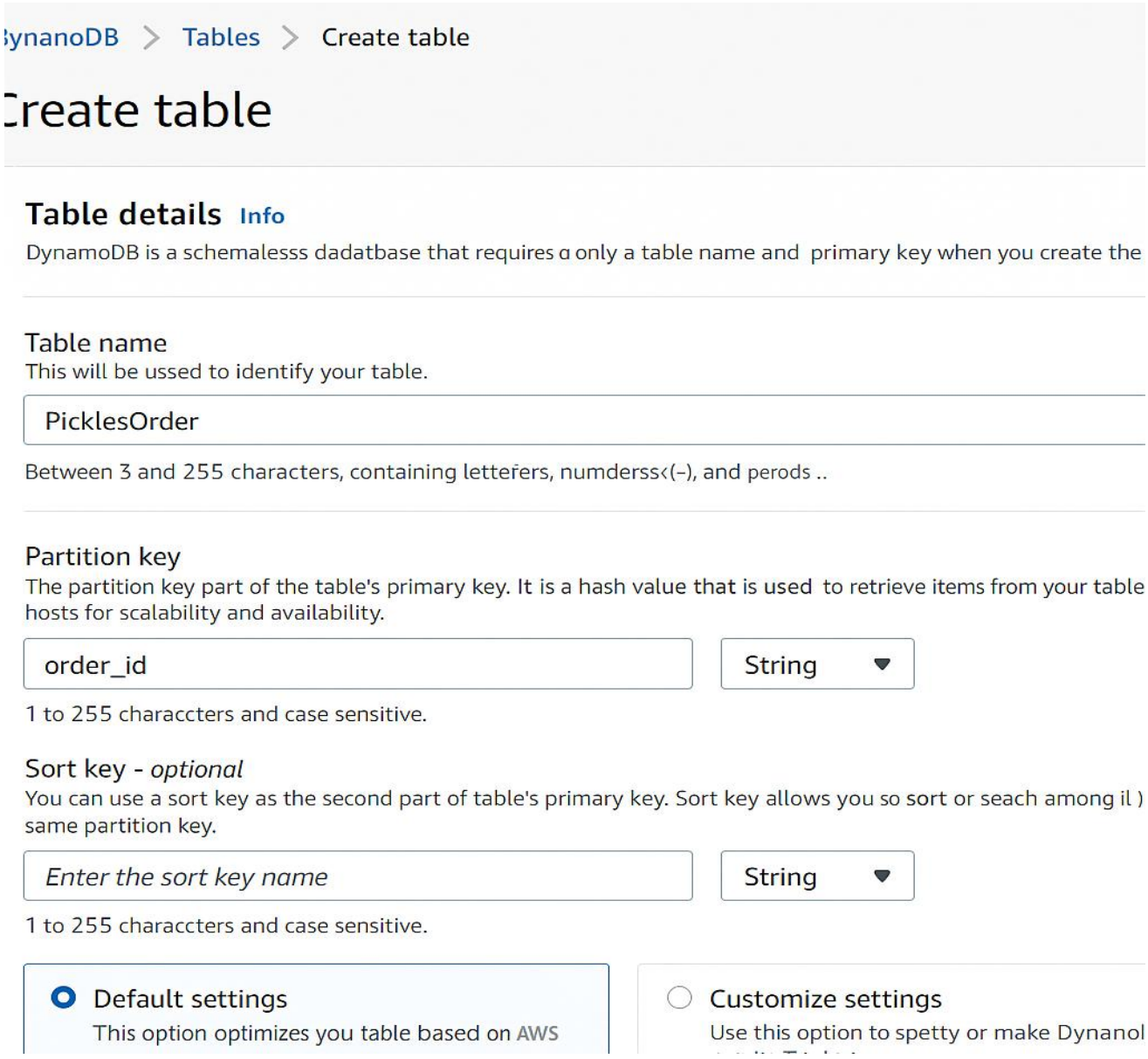

ynanoDB > Tables > Create table
reate table
Table details Info
DynamoDB is a schemalesss dadatbase that requires a only a table name and primary key when you create the
Table name
This will be ussed to identify your table.
PicklesOrder
Between 3 and 255 characters, containing letterers, numderss‹(-), and perods ..
Partition key
The partition key part of the table's primary key. It is a hash value that is used to retrieve items from your table hosts for scalability and availability.
order_id
String
1 to 255 characcters and case sensitive.
Sort key - optional
You can use a sort key as the second part of table's primary key. Sort key allows you so sort or seach among il) same partition key.
Enter the sort key name
String
1 to 255 characcters and case sensitive.
Default settings
This option optimizes you table based on AWS
Customize settings
Use this option to spetty or make Dynanol

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

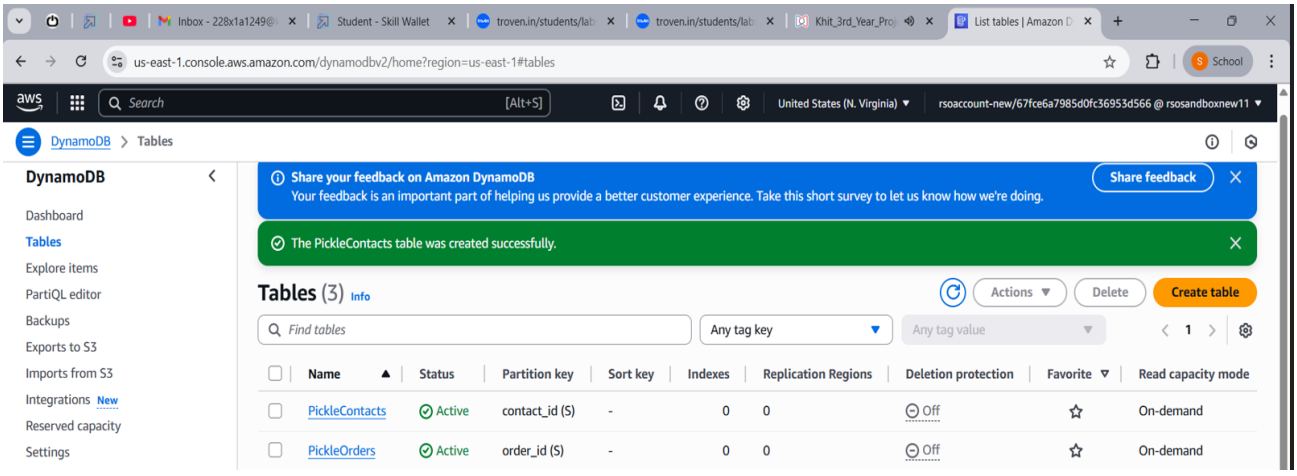


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

Tags

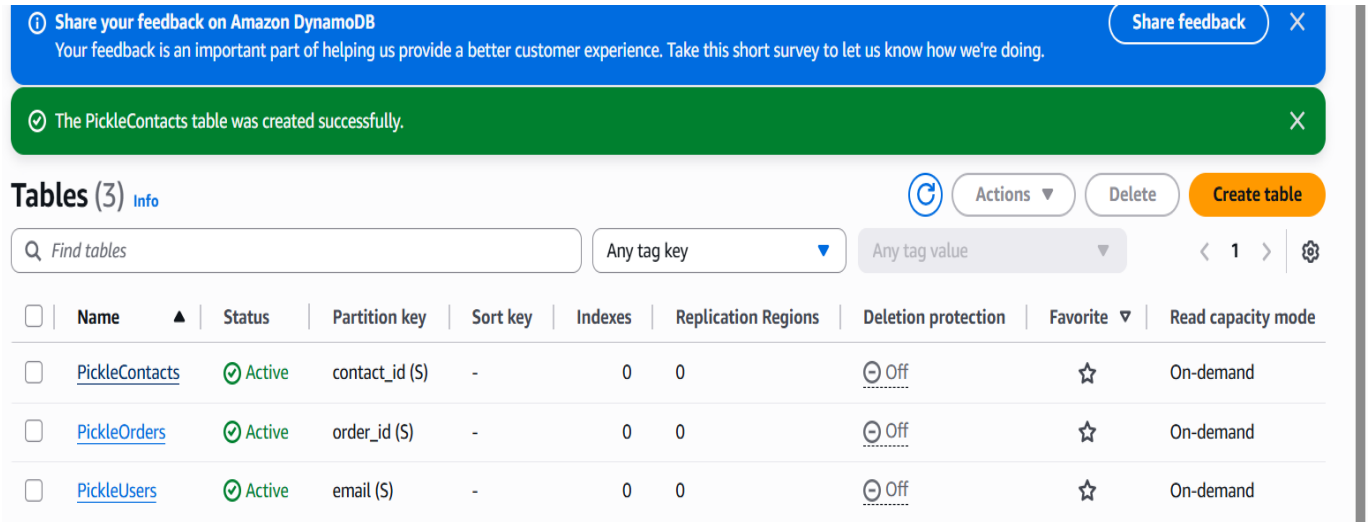
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

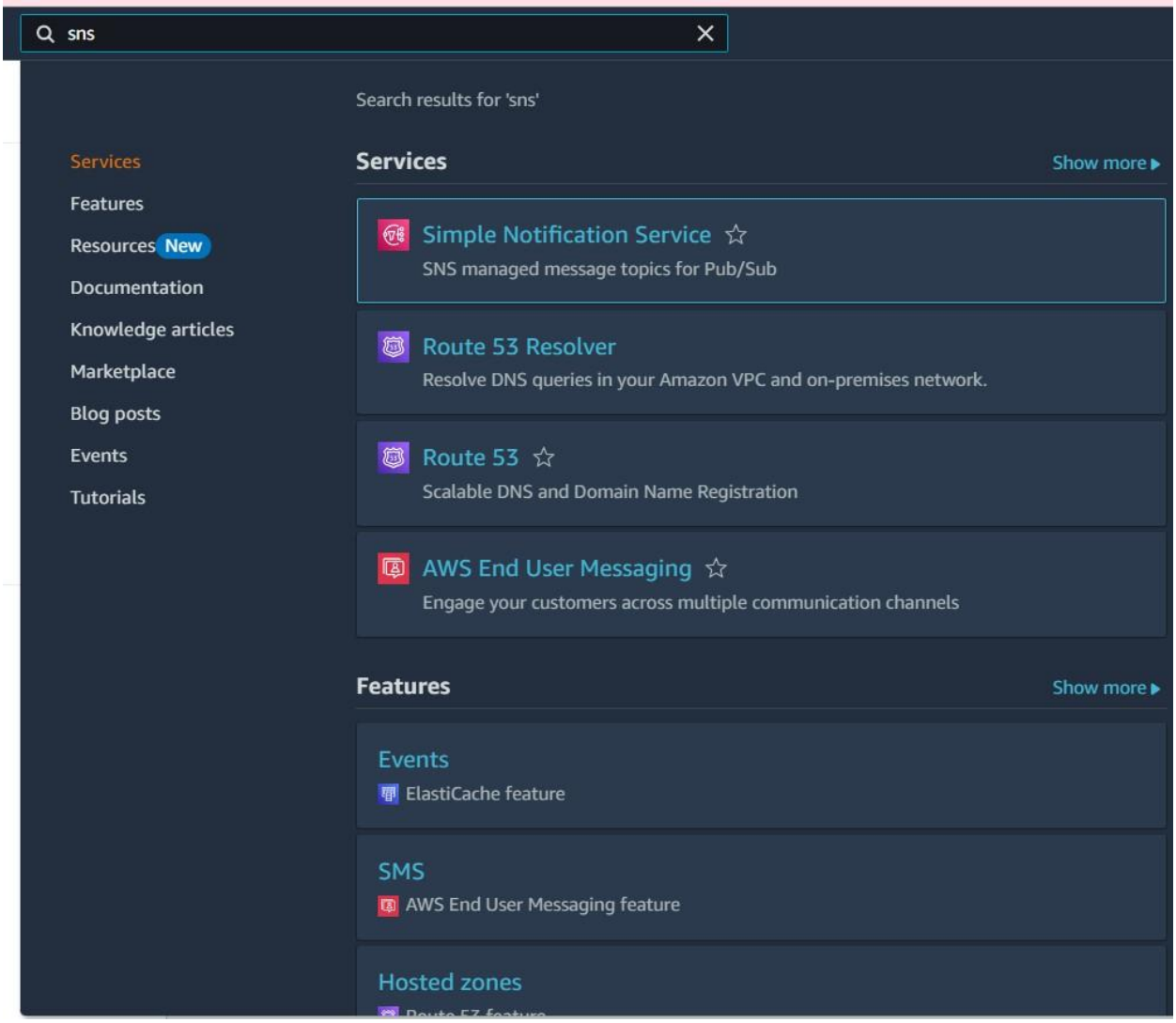
Cancel Create table

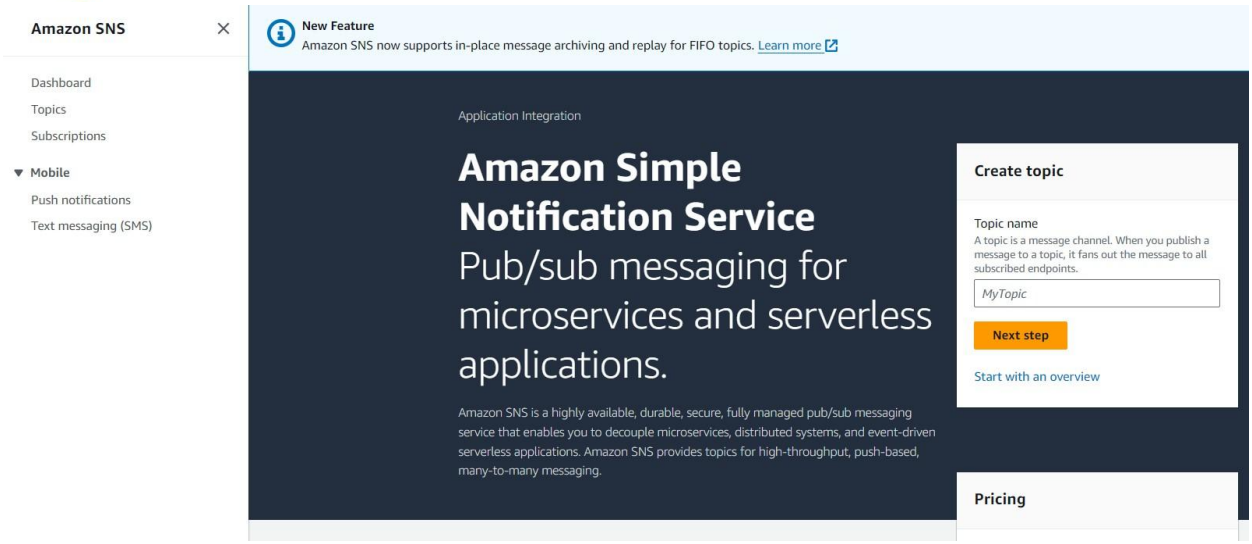


Milestone 3: SNS Notification Setup

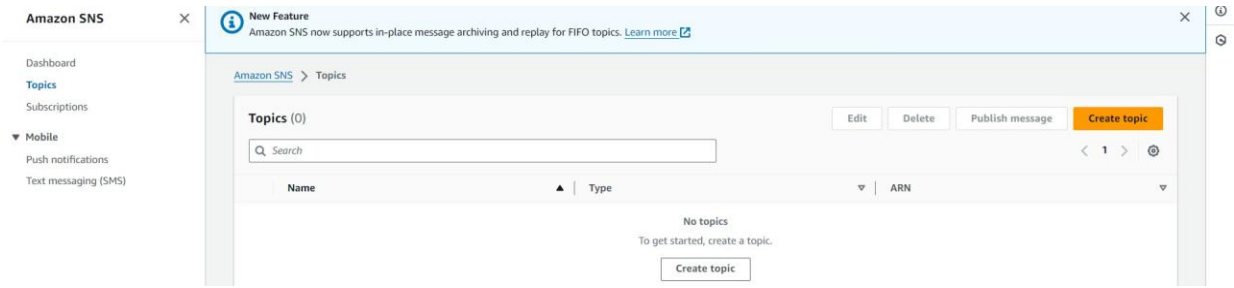
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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.

Milestone 4:Backend Development and Application Setup

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

File Explorer Structure

Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for images, and a templates/ directory containing all required HTML pages like home, login, signup, pickle-specific pages (e.g., Veg_Pickles.html, Non_Veg_Pickles.html,Snacks.html), and utility pages (e.g., order.html, contact.html).

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, request, render_template, redirect, url_for, session, flash,
jsonify
import boto3
import uuid
import os
import hashlib
from datetime import datetime
from botocore.exceptions import ClientError
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification.

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

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

- **Dynamodb Setup:**

```
# DynamoDB tables - Production ready
order_table = dynamodb.Table('PickleOrders')
user_table = dynamodb.Table('PickleUsers')
contact_table = dynamodb.Table('PickleContacts')
```

Description: initialize the DynamoDB resource for the us-east-1 region and set up access to the Users ,Orders, Contact tables for storing user details and pickles requests.

- **SNS Connection**

```
@app.route('/notify')
def notify():
    # Send a sample SNS message
    sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Message='A new order was received on Homemade Pickles & Snacks!',
        Subject='New Pickle Order Alert'
    )
    return "SNS notification sent!"
```

Description: Configure **SNS** to send notifications when a pickle request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region, name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

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

```python
@app.route('/home')
def home():
    return render_template('home.html')
```

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

  - **Signup Route :**

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            password = request.form['password']
            hashed_password = hash_password(password)
            timestamp = datetime.utcnow().isoformat()

            # Check if user already exists
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('User already exists', 'error')
                return render_template('signup.html')

            # Save user to DynamoDB
            user_table.put_item(Item={
                'email': email,
                'name': name,
                'password': hashed_password,
                'created_at': timestamp,
                'status': 'active'
            })
```

```
            # Send welcome email
            welcome_message = f"Dear {name},\n\nWelcome to Homemade Pickles &
            Snacks!\n\nYour account has been created successfully. You can now login and
            start ordering our delicious homemade pickles and snacks.\n\nThank you for
            joining us!\n\nBest regards,\nHomemade Pickles & Snacks Team"
            send_email_notification(email, 'Welcome to Homemade Pickles & Snacks!',
            welcome_message)

            flash('Account created successfully! Please login.', 'success')
            return redirect(url_for('index'))
        except Exception as e:
            flash(f'Signup failed: {str(e)}', 'error')
    return render_template('signup.html')
```

Description: define signup route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

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

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

            # Check user in DynamoDB
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                stored_password = response['Item'].get('password')
                if stored_password == hashed_password:
                    session['user_email'] = email
                    session['user_name'] = response['Item'].get('name')
                    return redirect(url_for('home'))

            flash('Invalid email or password', 'error')
        except Exception as e:
            flash(f'Login failed: {str(e)}', 'error')
    return render_template('login.html')
```

Description: define login route to validate user credentials against DynamoDB, check the password using encryption, update the login count on successful authentication, and redirect users to the home page.

- **Index Route :**

```
@app.route('/')
def index():
    return render_template('index.html')
```

Description: The index page serves as the homepage, welcoming users with an overview of your brand and guiding them to key sections like products, contact, or login.

- **Order route:**

```
@app.route('/order', methods=['GET', 'POST'])
def order():
    if request.method == 'POST':
        try:
            name = request.form['name']
            email = request.form['email']
            phone = request.form['phone']
            address = request.form['address']
            city = request.form['city']
            pincode = request.form['pincode']
            item = request.form['item']
            quantity = int(request.form['quantity'])
            notes = request.form.get('notes', '')
            order_id = str(uuid.uuid4())
            timestamp = datetime.utcnow().isoformat()

            # Save to DynamoDB with full order details
            order_table.put_item(Item={
                'order_id': order_id,
                'name': name,
                'email': email,
                'phone': phone,
                'address': address,
                'city': city,
                'pincode': pincode,
                'item': item,
                'quantity': quantity,
```

```
def order():
                'item': item,
                'quantity': quantity,
                'notes': notes,
                'timestamp': timestamp,
                'status': 'pending',
                'total_amount': quantity * 100  # Sample pricing
            })

            # Send email notifications
            customer_message = f"Dear {name},\n\nYour order has been placed
            successfully!\n\nOrder ID: {order_id}\nItem: {item}\nQuantity: {quantity}
            \n\nWe'll contact you soon for delivery details.\n\nThank you for choosing
            Homemade Pickles & Snacks!"
            admin_message = f"New Order Received!\n\nOrder ID: {order_id}\nCustomer:
            {name}\nEmail: {email}\nPhone: {phone}\nItem: {item}\nQuantity: {quantity}
            \nAddress: {address}, {city} - {pincode}\nNotes: {notes}"

            send_email_notification(email, 'Order Confirmation - Homemade Pickles &
            Snacks', customer_message)
            send_email_notification(ADMIN_EMAIL, f'New Order - {order_id}', admin_message)

            session['last_order_id'] = order_id
            return redirect(url_for('sucess'))
        except Exception as e:
            flash(f'Order processing failed: {str(e)}', 'error')
            return render_template('order.html')
    return render_template('order.html')
```

Description: The order page allows customers to review their selected items, enter shipping details, and confirm payment to complete the purchase. It acts as the final step in the checkout process, ensuring a smooth and secure transaction.

- **Aws-info Route:**

```python
@app.route('/aws-info')
def aws_info():
    """Display AWS service information"""
    try:
        # Get EC2 instance info
        instance_id = get_instance_info()

        # Get IAM role info
        try:
            sts = boto3.client('sts', region_name='us-east-1')
            identity = sts.get_caller_identity()
            account_id = identity['Account']
            role_arn = identity.get('Arn', 'No role attached')
        except:
            account_id = 'Unknown'
            role_arn = 'No role attached'

        info = {
            'instance_id': instance_id,
            'account_id': account_id,
            'role_arn': role_arn,
            'region': 'us-east-1'
        }
        return jsonify(info)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Description: The Aws-info route displays details about your website's integration with AWS, such as database status, storage usage, or service configurations. It helps admins monitor backend health and ensure smooth operation of the homemade pickles and snacks platform**.**

- **About route:**

```
@app.route('/about')
def about():
    return render_template('about.html')
```

Description: The about route displays a heartfelt introduction to your homemade pickles and snacks brand, sharing its origin, mission, and values. It builds trust and connection by telling your story and celebrating what makes your products unique.

- **Success route:**

```
@app.route('/sucess')
def sucess():
    return render_template('sucess.html')
```

Description: The success route displays a confirmation message after a successful order or form submission, reassuring users that their action was completed. It adds a friendly closing touch to the user journey with gratitude or next-step instructions.

- **Cart route:**

```
@app.route('/cart')
def cart():
    return render_template('cart.html')
```

Description: The cart route displays all items a user has added for purchase, allowing them to review quantities and total cost before checkout. It acts as a shopping basket, giving users the option to update or remove products before finalizing their order

- **Logout route:**

```python
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))
```

Description: The logout route securely ends the user's session, logging them out of their account. It helps protect personal information and ensures a fresh login is required for future activity**.**

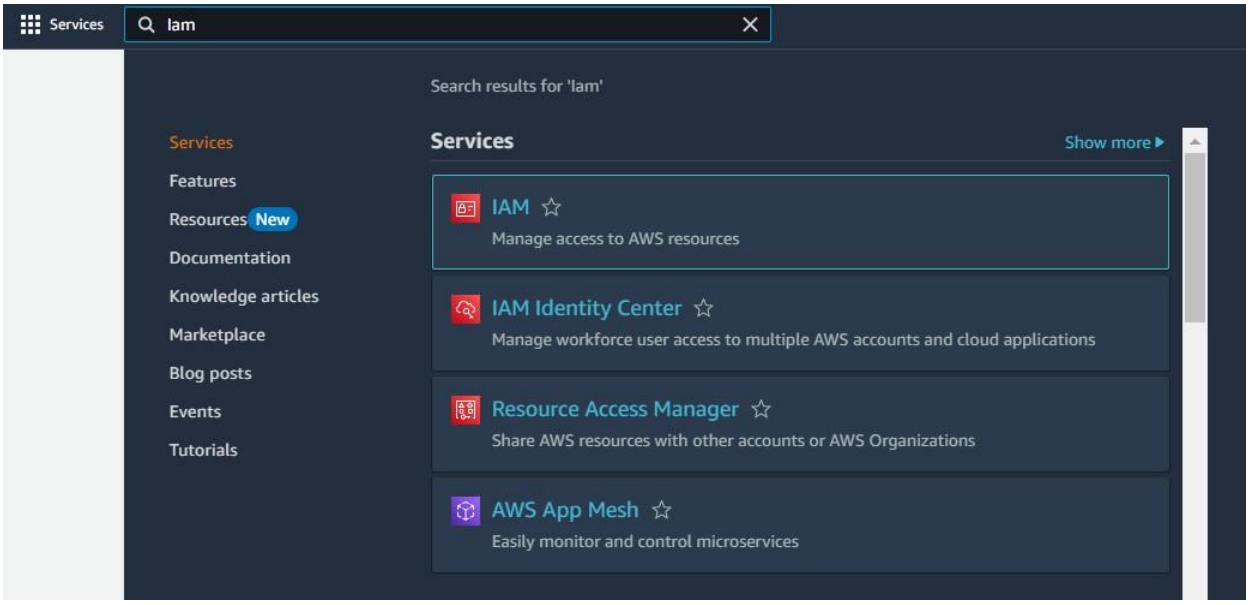
- **Deployment Code:**

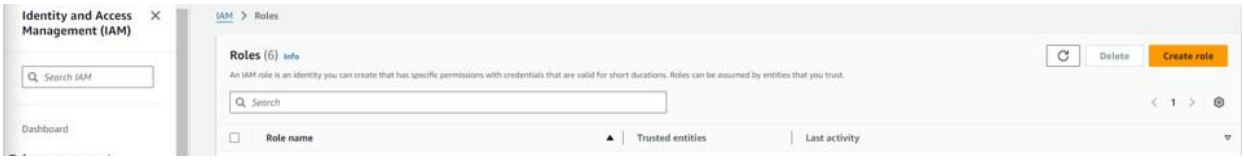
```python
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    debug = os.environ.get('DEBUG', 'False').lower() == 'true'
    app.run(host='0.0.0.0', port=port, debug=debug)
```

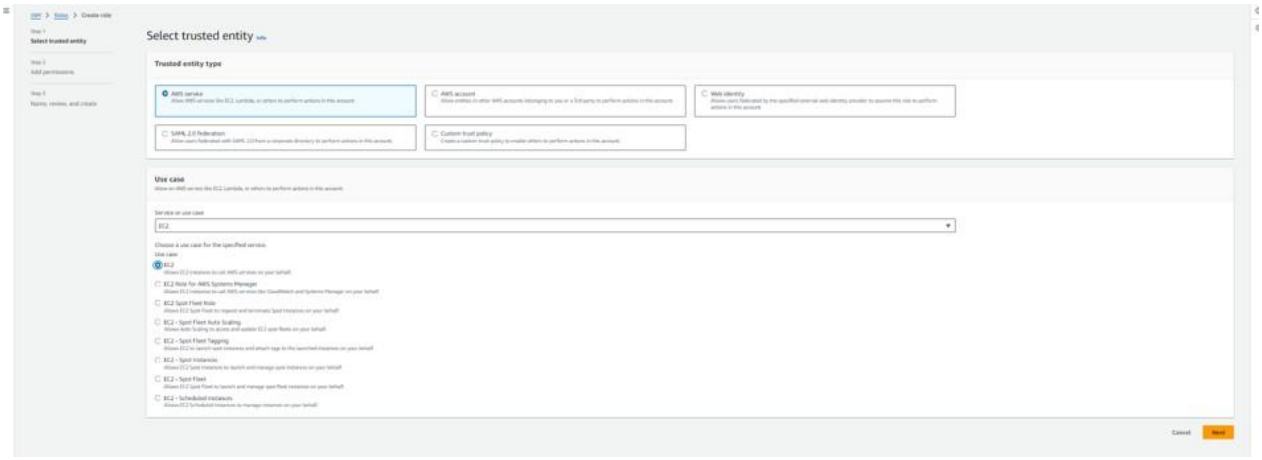
Description: The deployment code sets up and launches the homemade pickles and snacks website on a cloud platform like AWS or Heroku, ensuring public access. It typically includes configurations for web hosting, environment variables, and service integration to keep the app running smoothly.

Milestone 5: IAM Role Setup

- **Activity 5.1:Create IAM Role.**
    - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



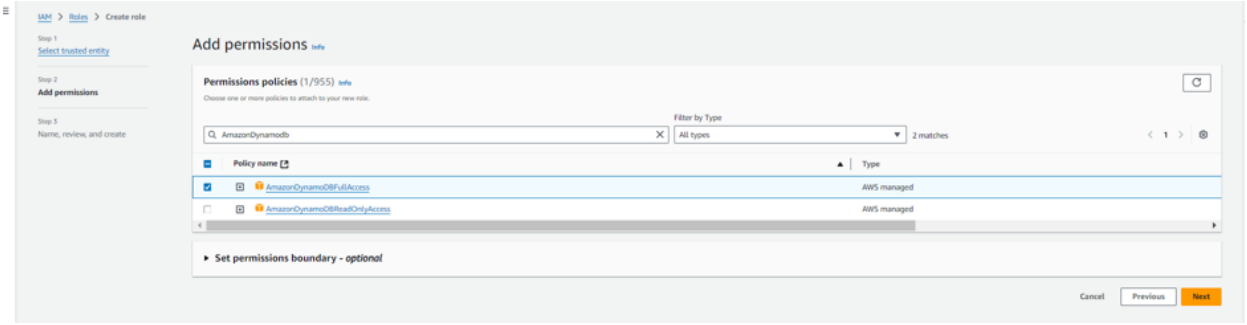


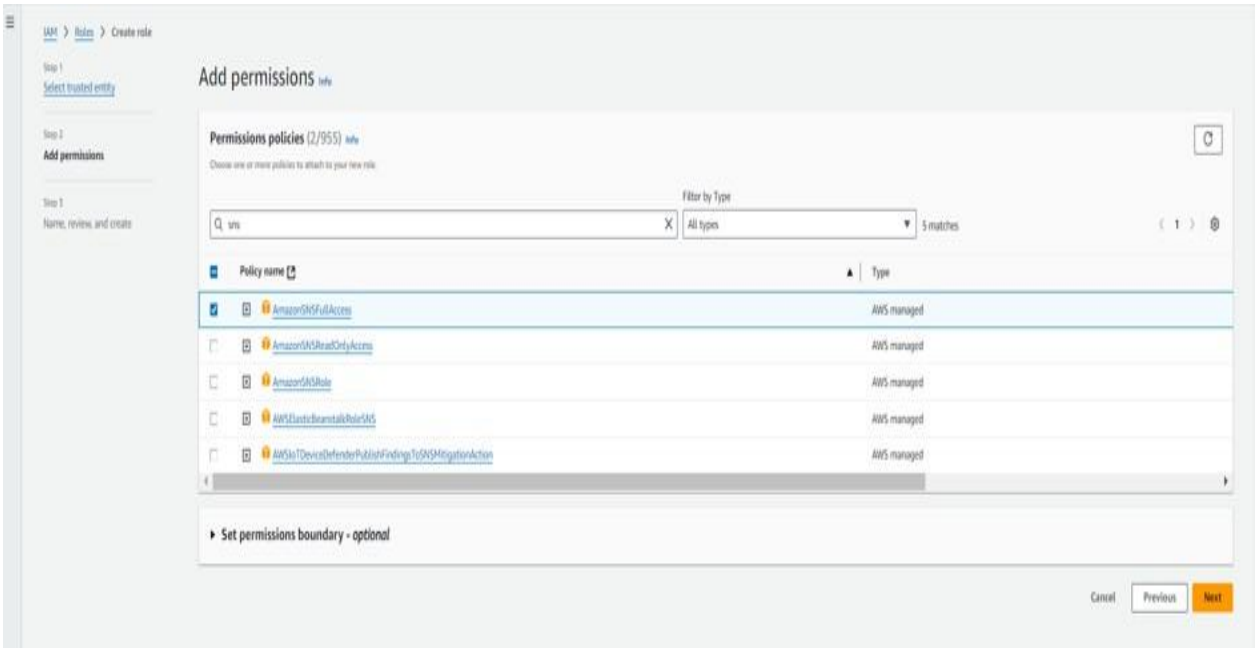


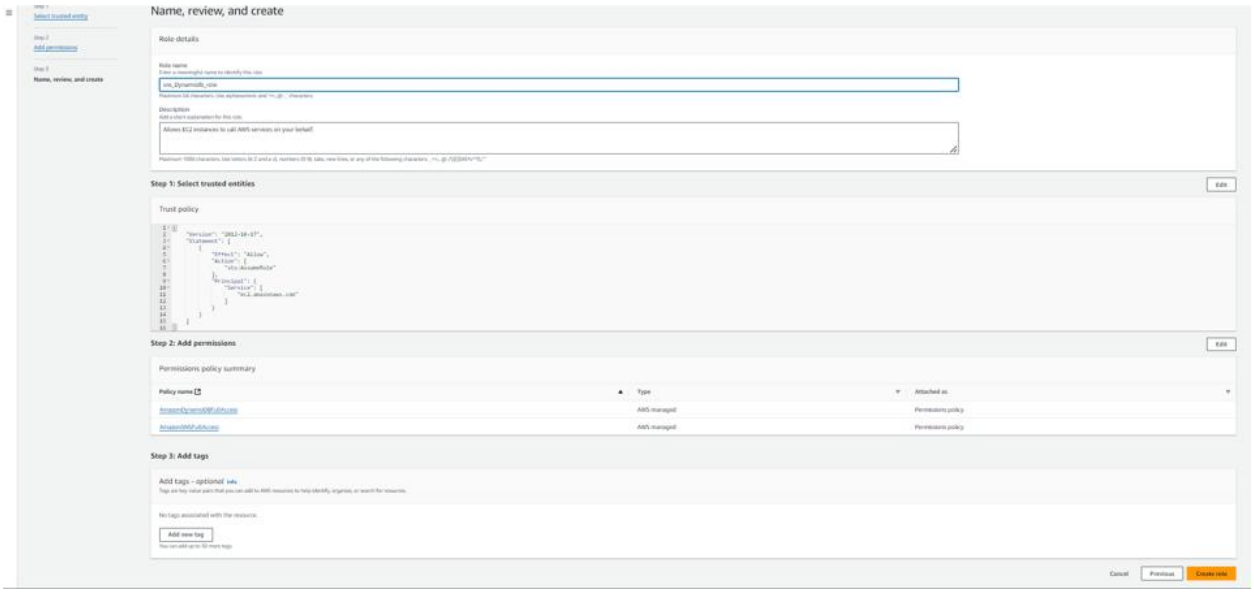
- **Activity 5.2: Attach Policies.**

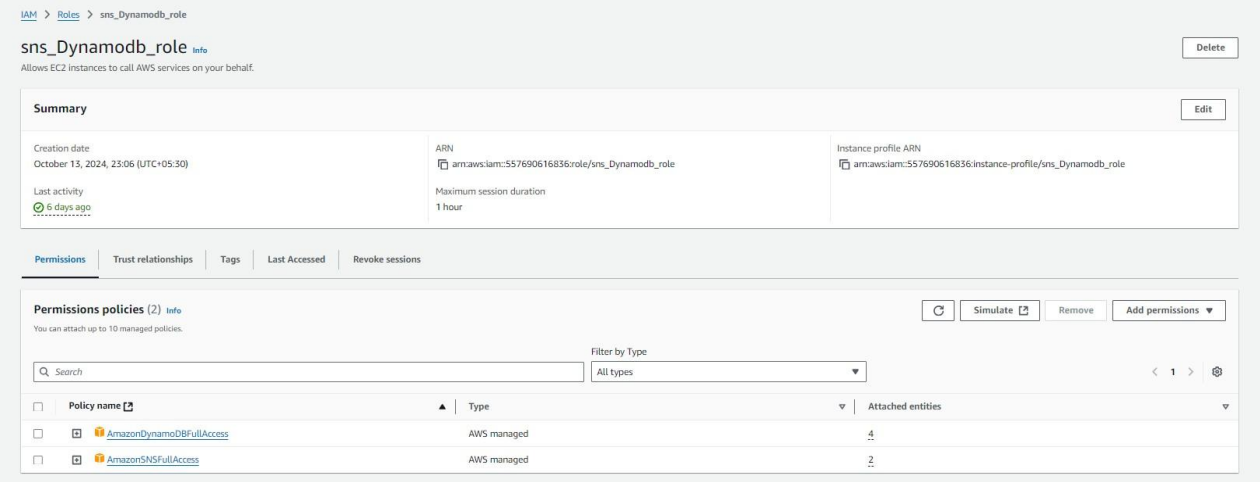
  Attach the following policies to the role:

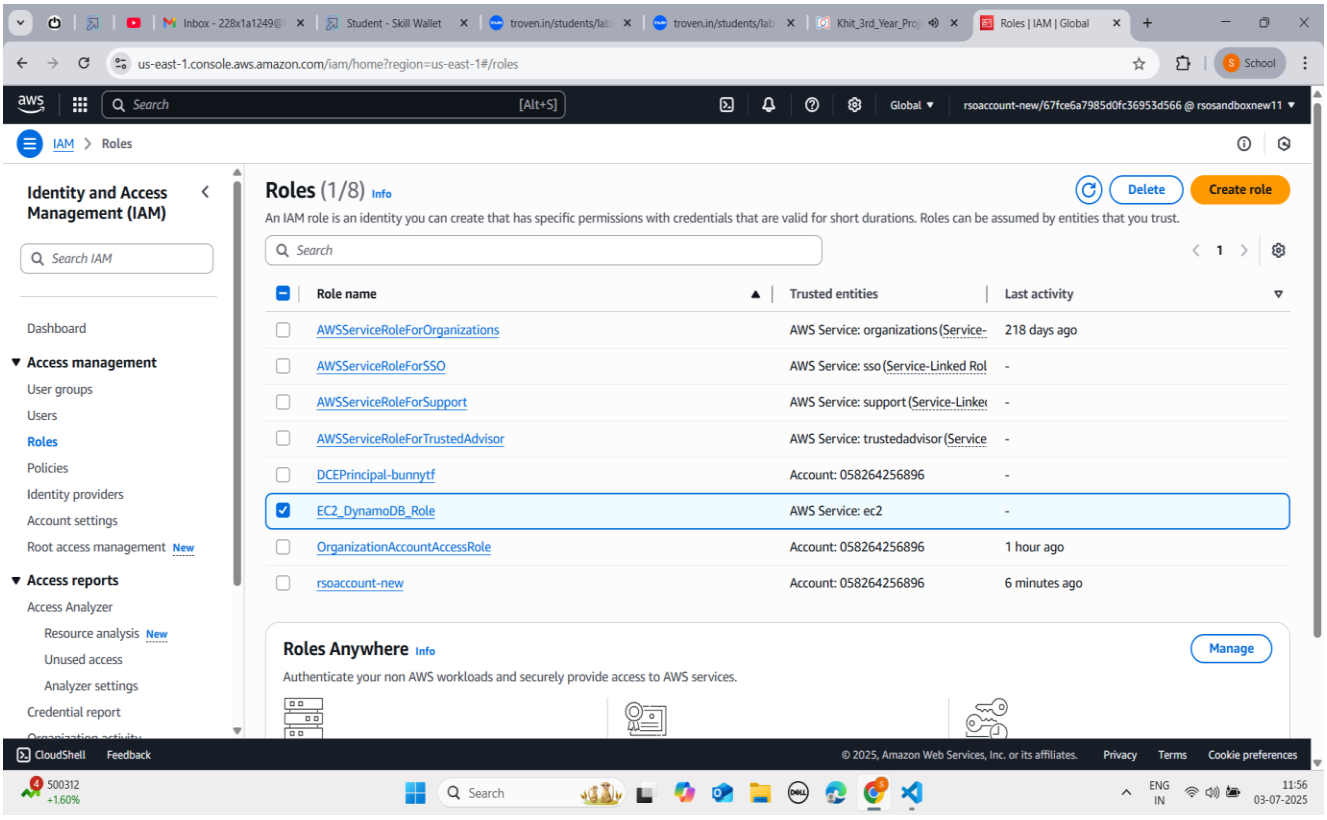
  - AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
  - AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.




IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
1
Policy name
Type
AmazonSNSFullAccess
AWS managed
AmazonSNSReadOnlyAccess
AWS managed
AmazonSNSRole
AWS managed
AWSElasticBeanstalkRoleSNS
AWS managed
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction
AWS managed
Set permissions boundary - optional
Cancel
Previous
Next


Name, review, and create
Role details
Role name
sns_dynamodb_role
Description
Allows EC2 instances to call AWS services on your behalf.
Step 1: Select trusted entities
Edit
Trust policy
Step 2: Add permissions
Edit
Permissions policy summary
Policy name
Type
Attached as
AWS managed
Permissions policy
AWS managed
Permissions policy
Step 3: Add tags
Add tags - optional Info
No tags associated with the resource.
Add new tag
Cancel
Previous
Create role

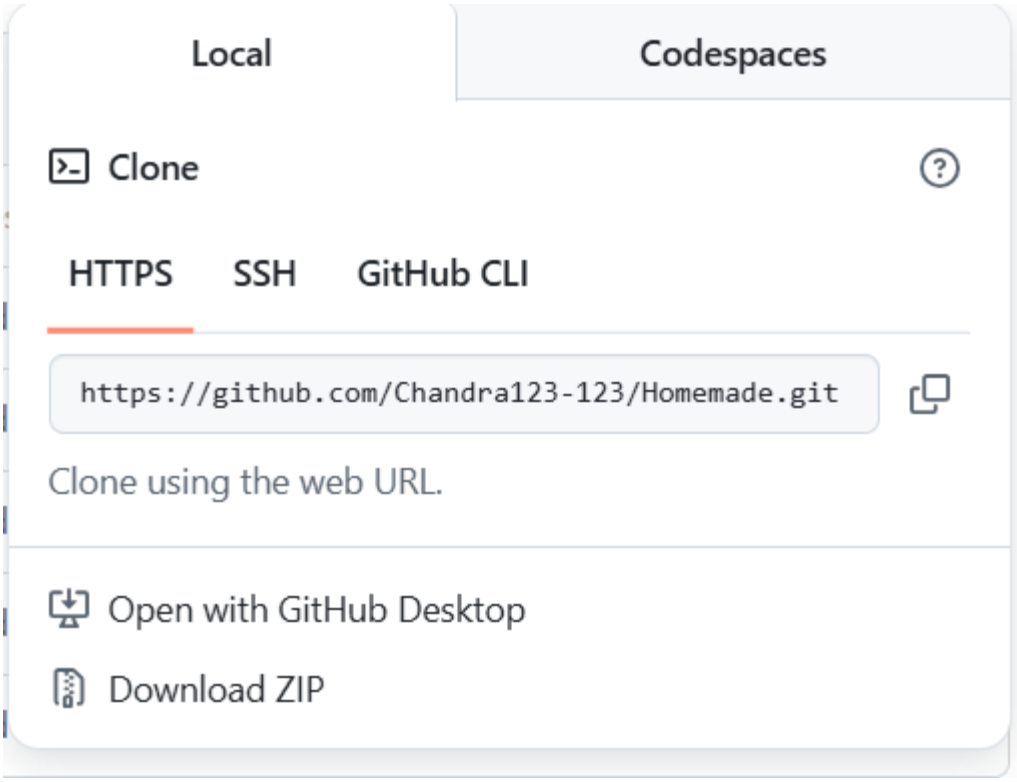

IAM > Roles > sns_Dynamodb_role
sns_Dynamodb_role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
October 13, 2024, 23:06 (UTC+05:30)
ARN
arn:aws:iam::557690616836:role/sns_Dynamodb_role
Instance profile ARN
arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role
Last activity
6 days ago
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
4
AmazonSNSFullAccess
AWS managed
2


us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles
Search
[Alt+S]
Global
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report
Roles (1/8) Info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Delete
Create role
Search
Role name
Trusted entities
Last activity
AWSServiceRoleForOrganizations
AWS Service: organizations (Service-
218 days ago
AWSServiceRoleForSSO
AWS Service: sso (Service-Linked Rol
-
AWSServiceRoleForSupport
AWS Service: support (Service-Linke
-
AWSServiceRoleForTrustedAdvisor
AWS Service: trustedadvisor (Service
-
DCEPrincipal-bunnytf
Account: 058264256896
-
EC2_DynamoDB_Role
AWS Service: ec2
-
OrganizationAccountAccessRole
Account: 058264256896
1 hour ago
rsoaccount-new
Account: 058264256896
6 minutes ago
Roles Anywhere Info
Manage
Authenticate your non AWS workloads and securely provide access to AWS services.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Milestone 6: EC2 Instance Setup

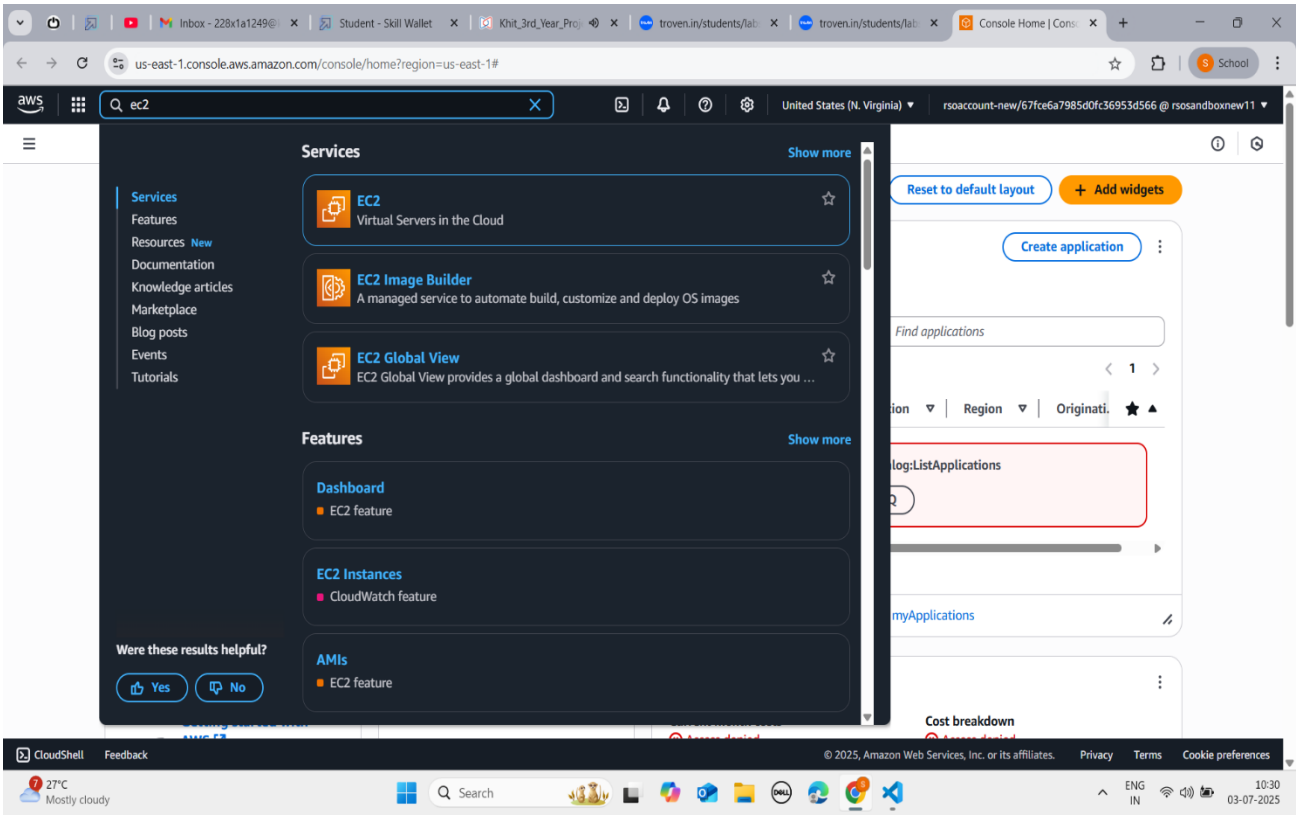
- **Note: Load your Flask app and Html files into GitHub repository.**

static/images	Delete static/images/images - Copy.jpg
templates	Add files via upload
README-AWS.md	Add files via upload
app.py	Add files via upload
aws-setup.py	Add files via upload
deploy.py	Add files via upload

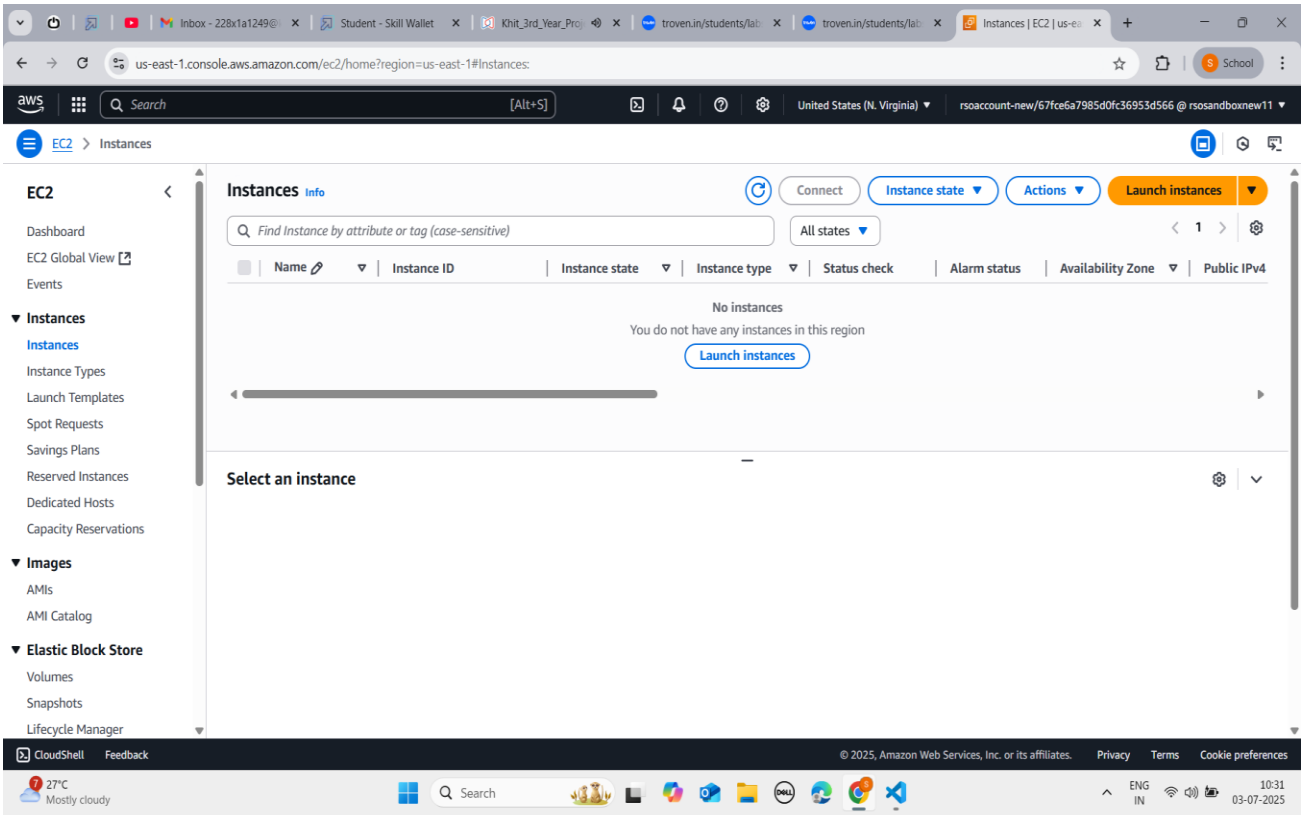


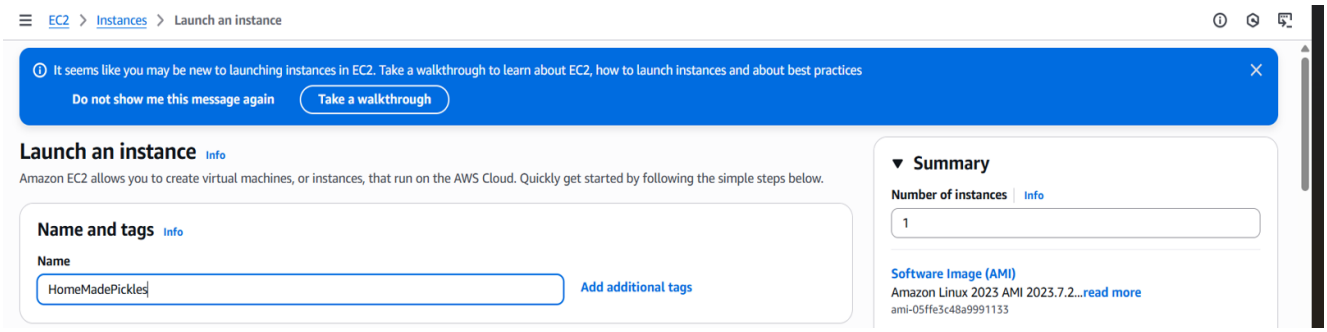
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

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

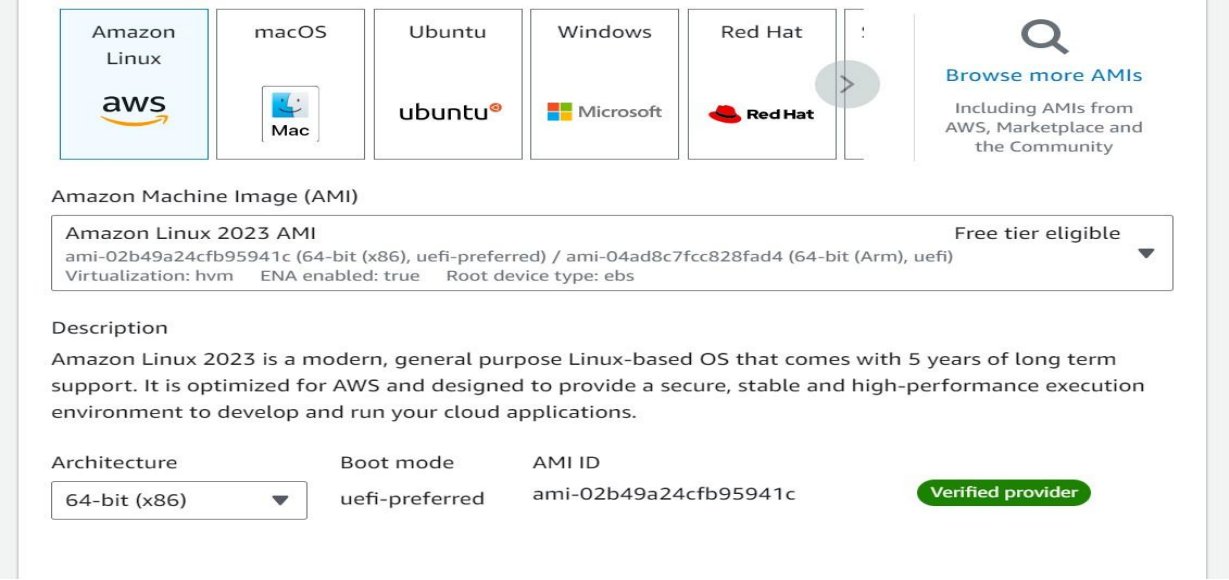


- Click on Launch instance to launch EC2 instance

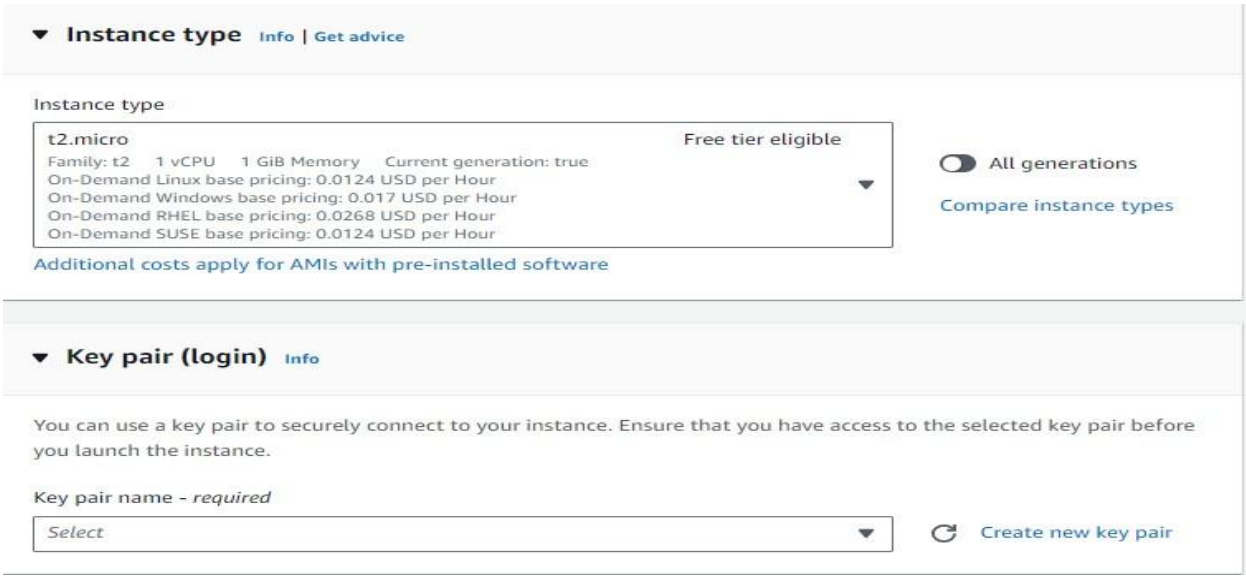


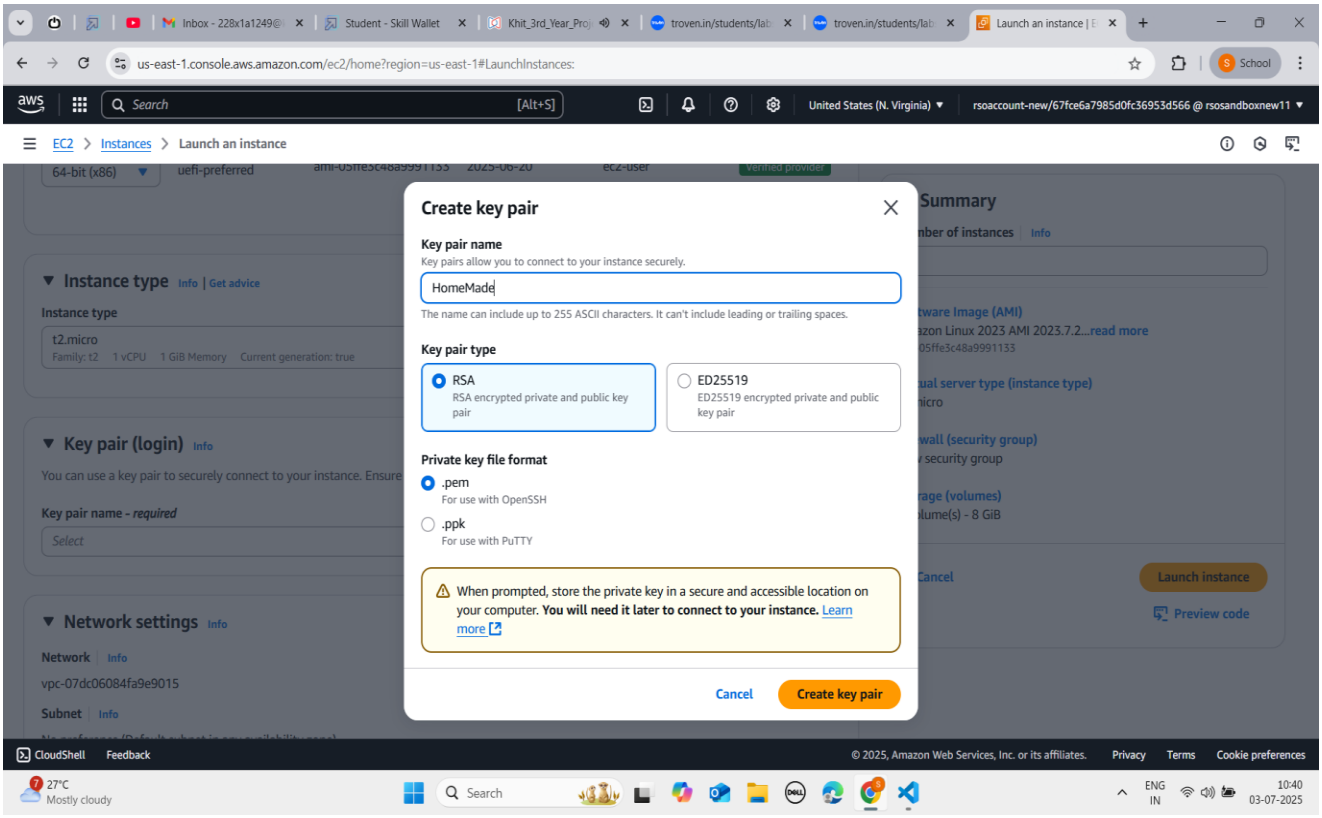


- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



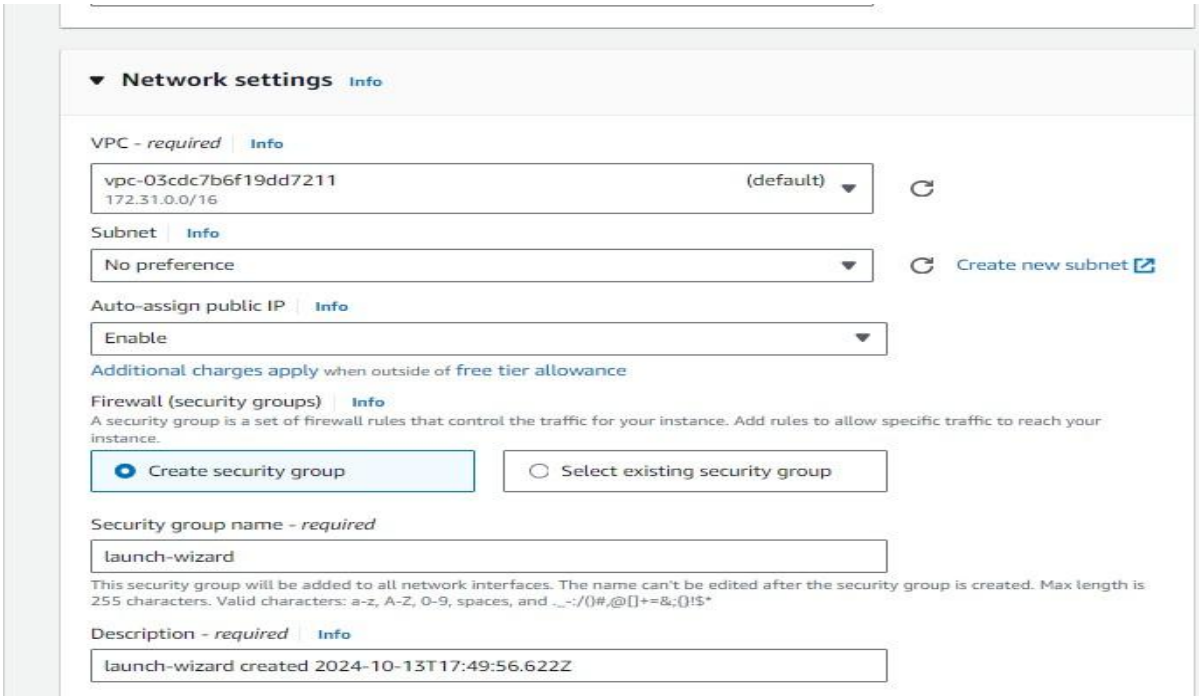
- Create and download the key pair for Server access.

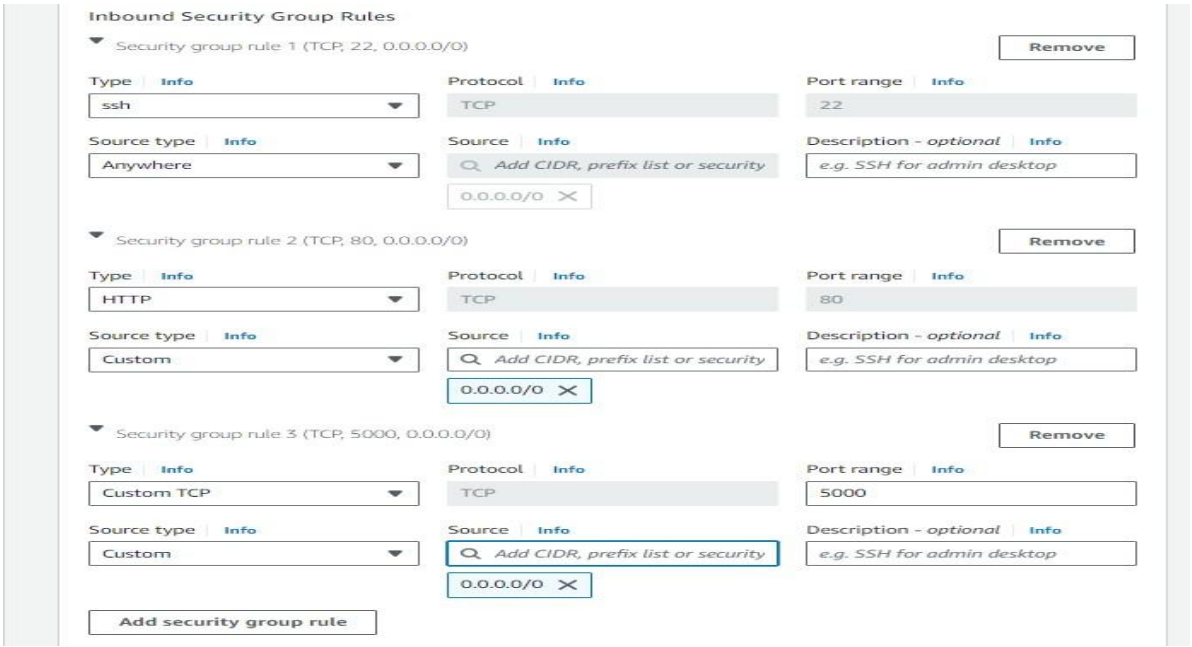


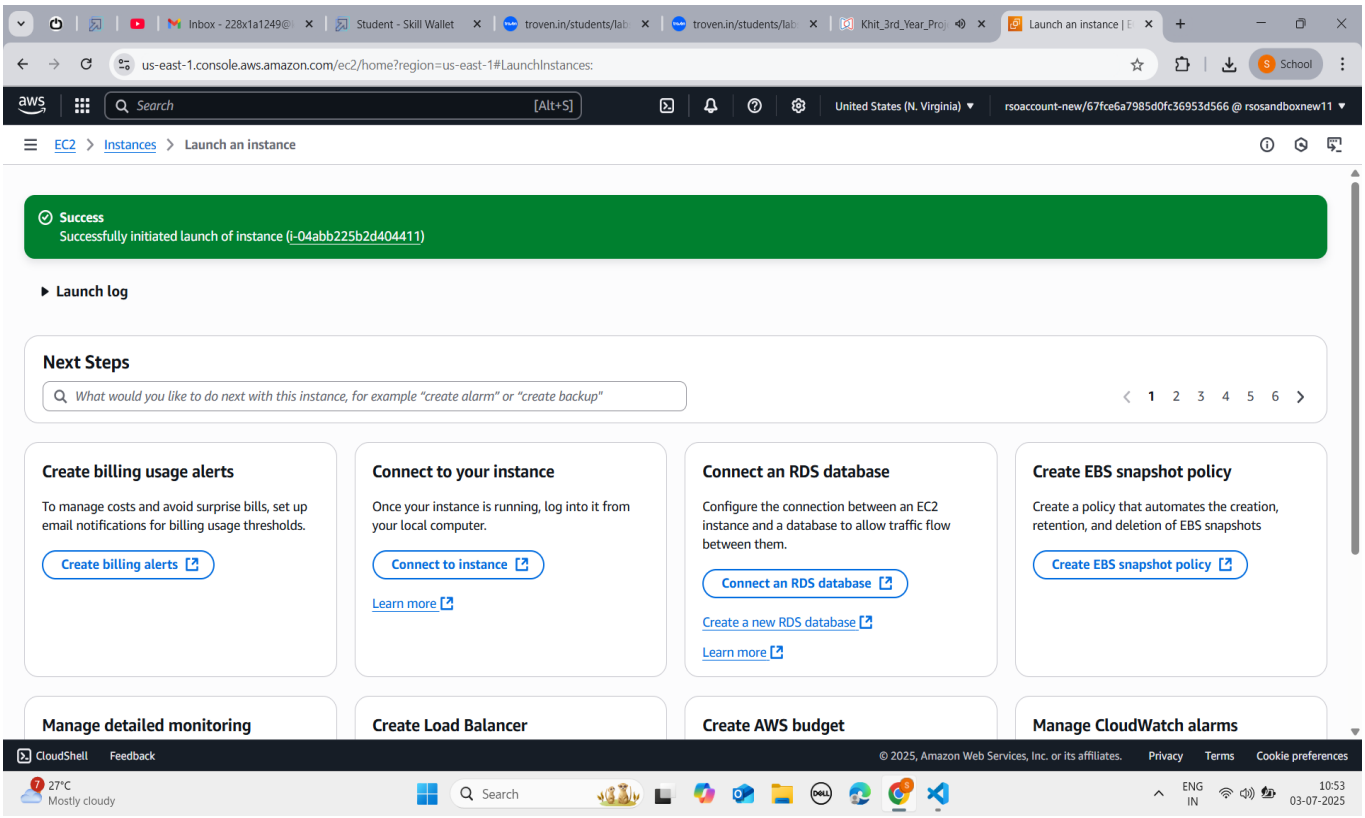

Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
HomeMade
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key pair
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


HomeMade.pem

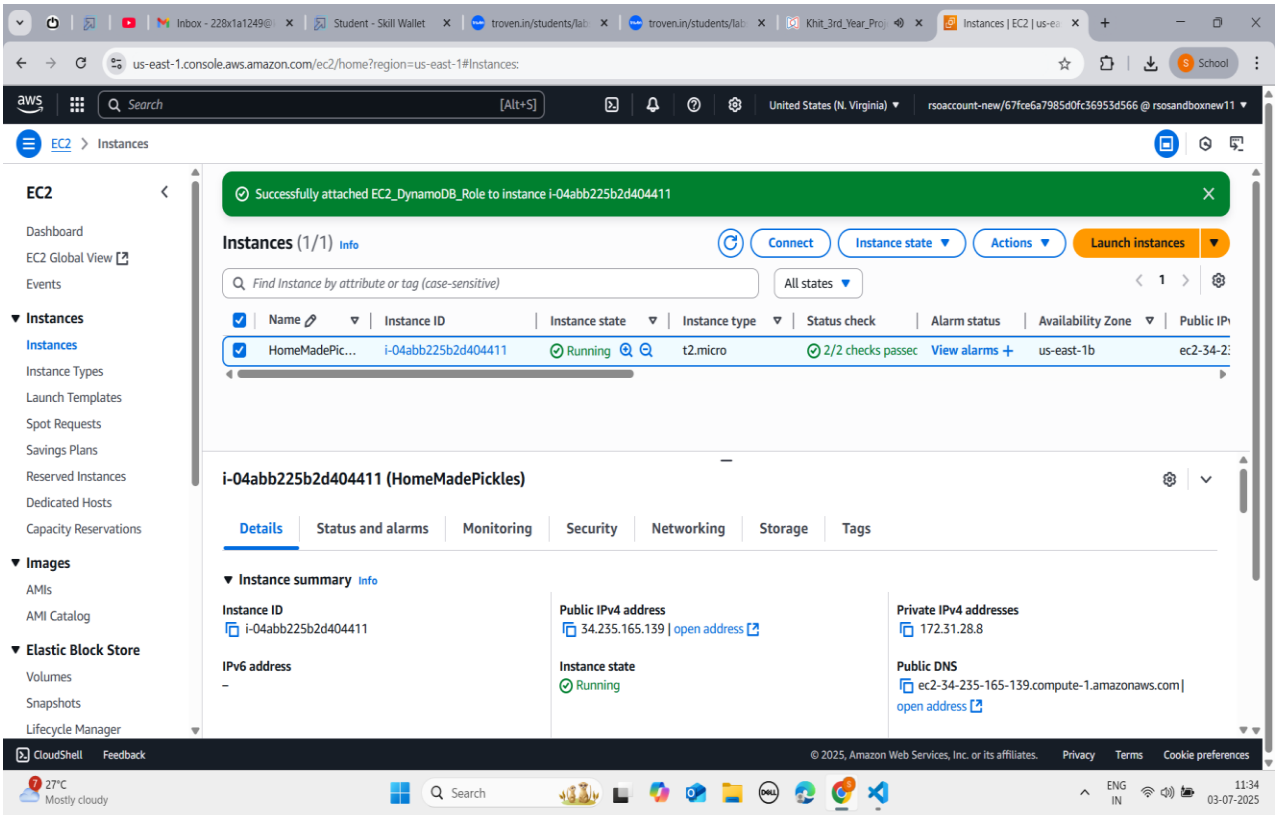
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



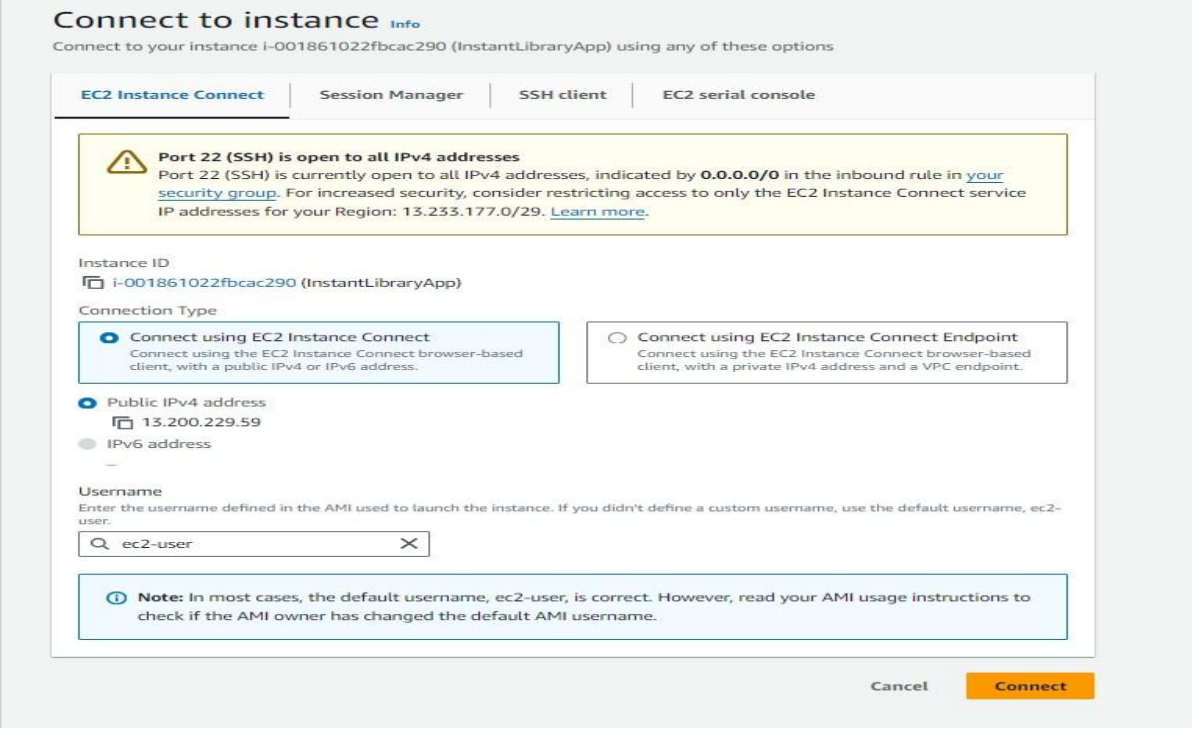


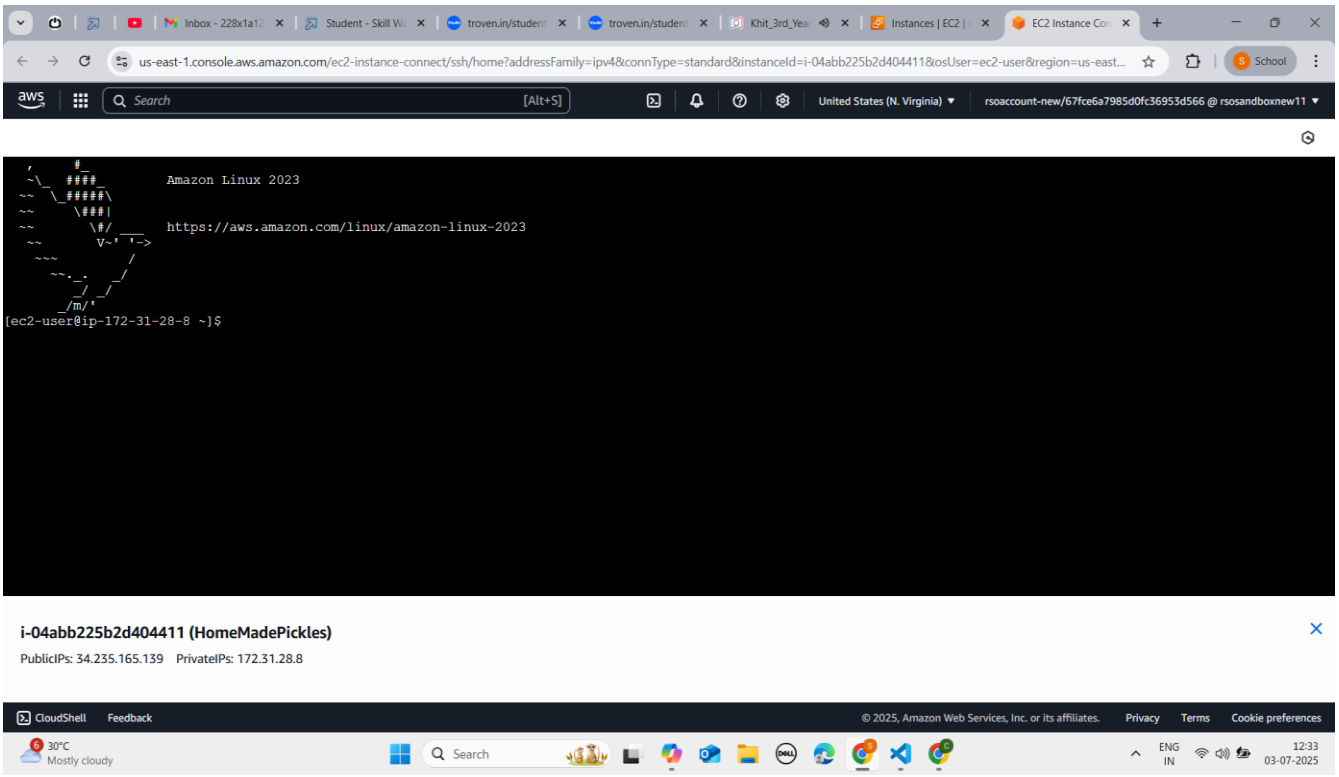


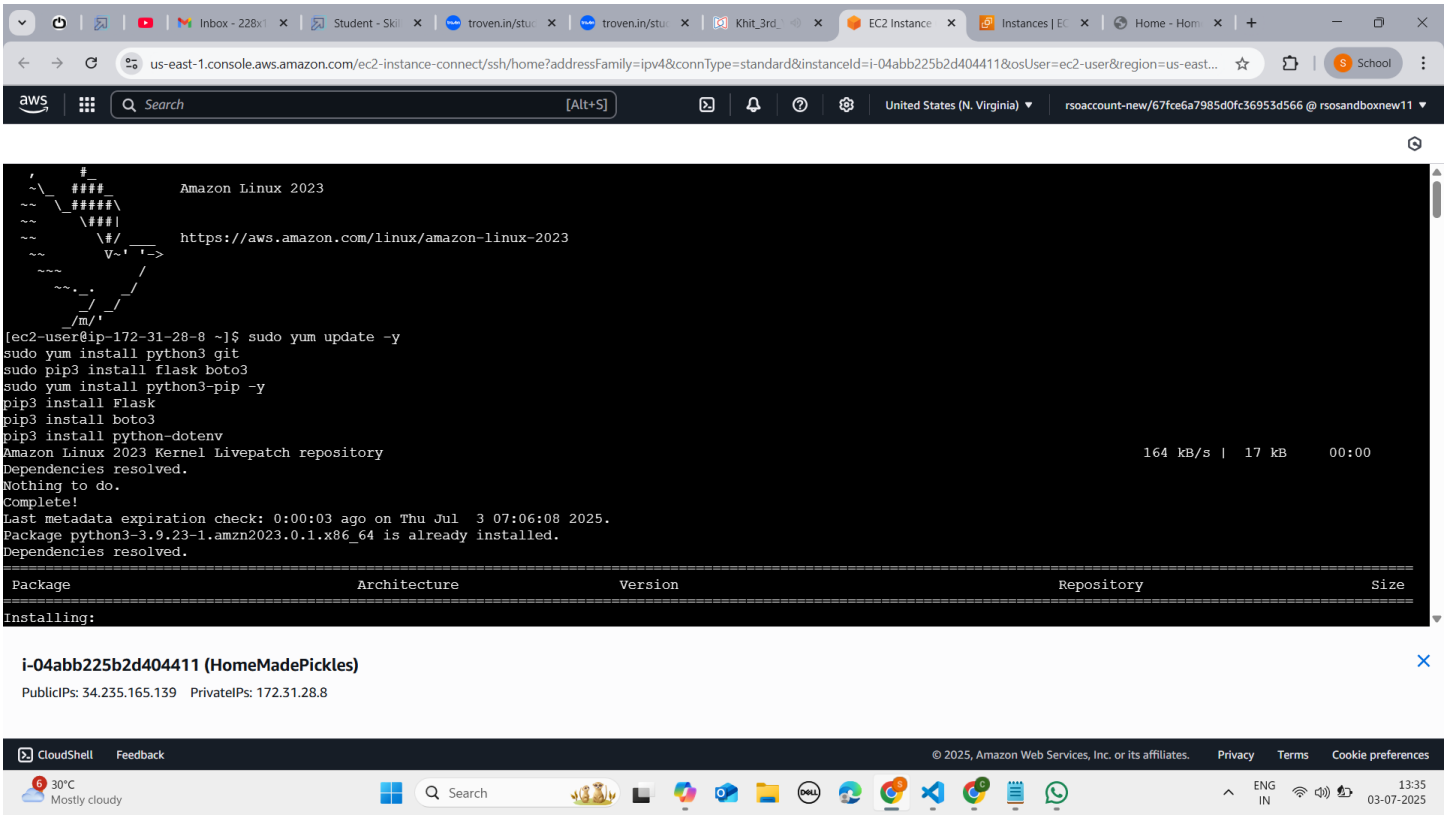
To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

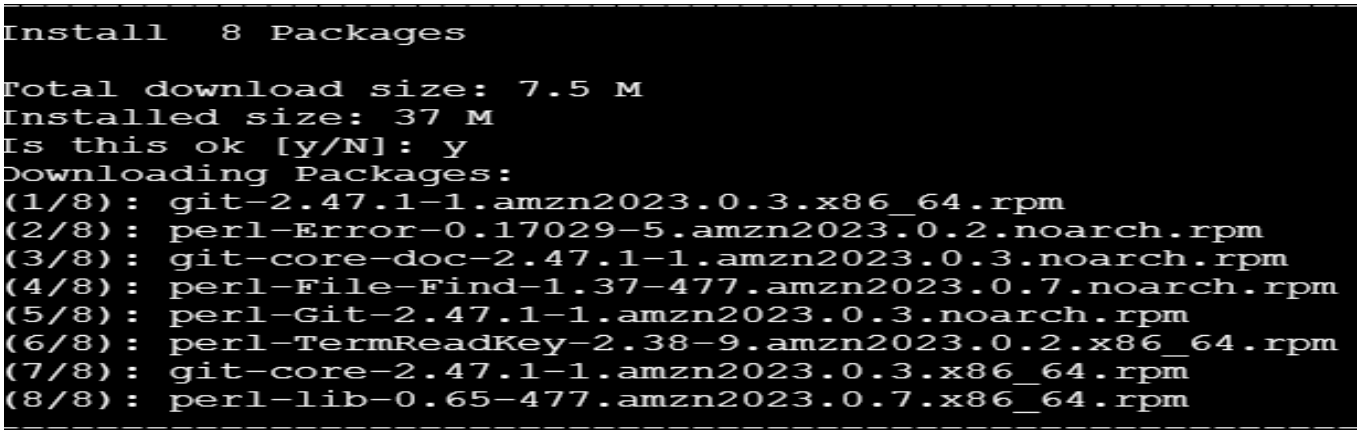

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:
Search [Alt+S]
United States (N. Virginia)
EC2 > Instances
Successfully attached EC2_DynamoDB_Role to instance i-04abb225b2d404411
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
HomeMadePic...
i-04abb225b2d404411
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
i-04abb225b2d404411 (HomeMadePickles)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-04abb225b2d404411
Public IPv4 address
34.235.165.139 | open address
Private IPv4 addresses
172.31.28.8
IPv6 address
–
Instance state
Running
Public DNS
ec2-34-235-165-139.compute-1.amazonaws.com | open address
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
27°C
Mostly cloudy
Search
ENG
IN
11:34
03-07-2025

- Now connect the EC2 with the files






Inbox - 228x1
Student - Ski
troven.in/stu
troven.in/stu
Khit_3rd_
EC2 Instance
Instances | EC
Home - Hom
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
School
aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-28-8 ~]$ sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
sudo yum install python3-pip -y
pip3 install Flask
pip3 install boto3
pip3 install python-dotenv
Amazon Linux 2023 Kernel Livepatch repository
164 kB/s | 17 kB 00:00
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:00:03 ago on Thu Jul 3 07:06:08 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Package
Architecture
Version
Repository
Size
Installing:
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
30°C
Mostly cloudy
Search
ENG
IN
13:35
03-07-2025


Install 8 Packages
Total download size: 7.5 M
Installed size: 37 M
Is this ok [y/N]: y
Downloading Packages:
(1/8): git-2.47.1-1.amzn2023.0.3.x86_64.rpm
(2/8): perl-Error-0.17029-5.amzn2023.0.2.noarch.rpm
(3/8): git-core-doc-2.47.1-1.amzn2023.0.3.noarch.rpm
(4/8): perl-File-Find-1.37-477.amzn2023.0.7.noarch.rpm
(5/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm
(6/8): perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64.rpm
(7/8): git-core-2.47.1-1.amzn2023.0.3.x86_64.rpm
(8/8): perl-lib-0.65-477.amzn2023.0.7.x86_64.rpm

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
sudo yum install python3-pip -y
pip3 install Flask
pip3 install boto3
pip3 install python-dotenv
```

Verify Installations:

flask --version

git --version

Activity 7.2:Clone Your Flask Project from GitHub

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

Run: 'git clone https://github.com/Chandra123-123/Homemade.git'

Note: change your-github-username and your-repository-name with your credentials

here: 'git clone https://github.com/Chandra123-123/Homemade.git'

- This will download your project to the EC2 instance.

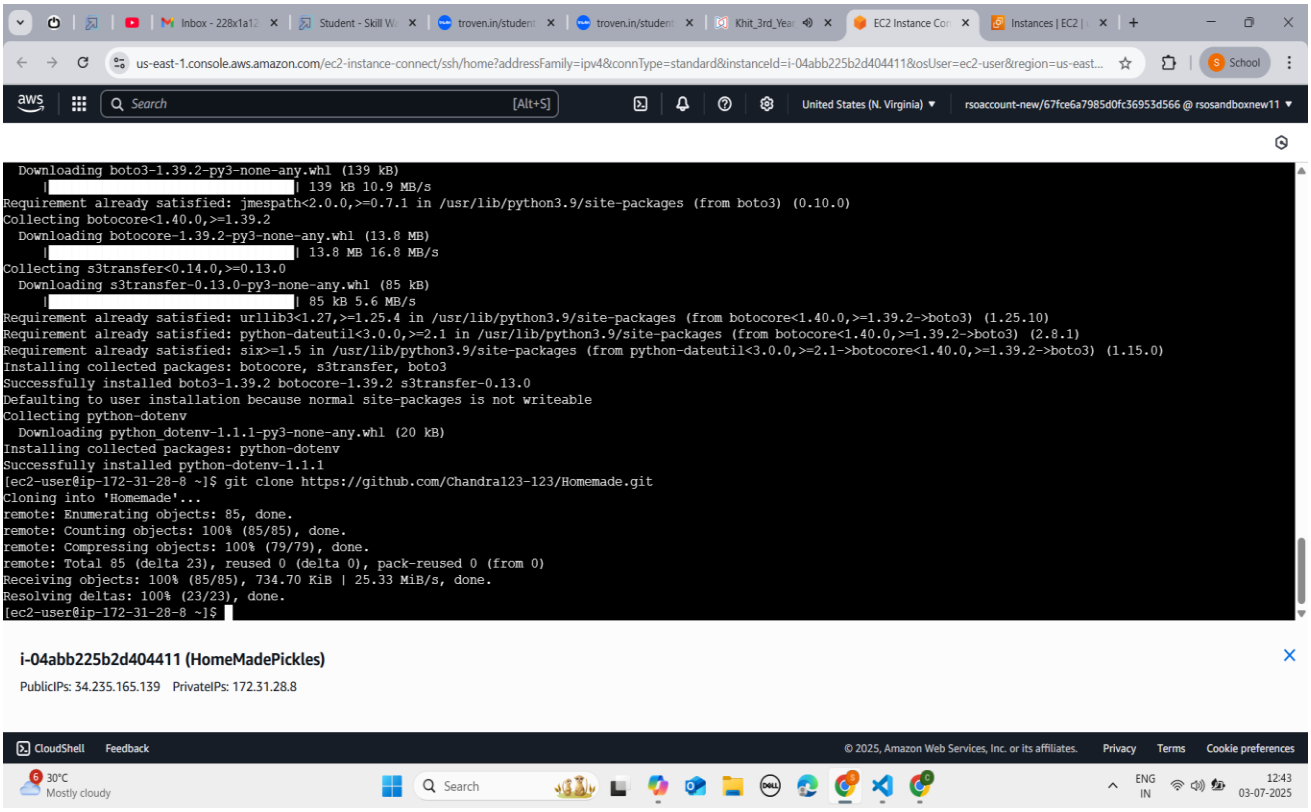
To navigate to the project directory, run the following command:

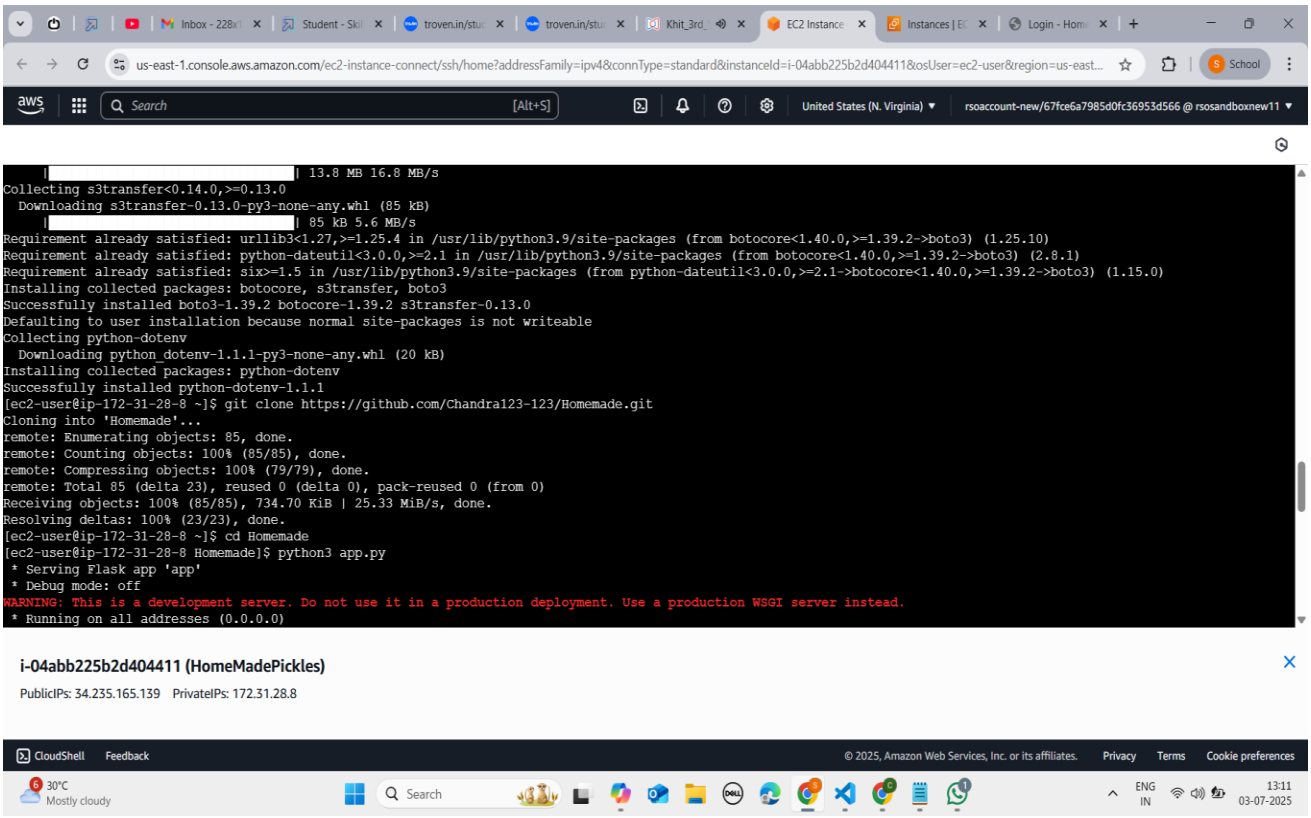
cd Homemade

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

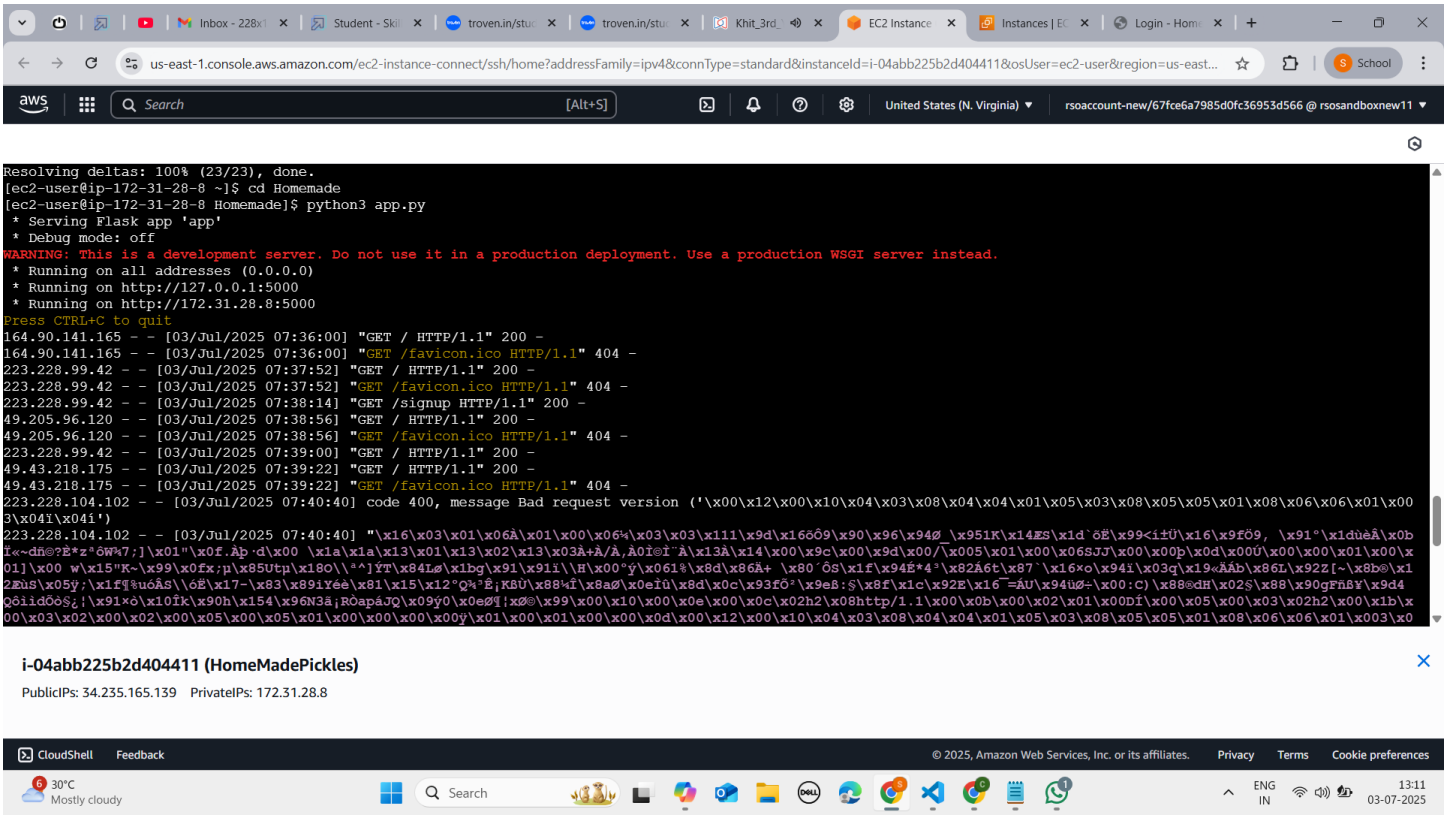

us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
Downloading boto3-1.39.2-py3-none-any.whl (139 kB)
| 139 kB 10.9 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.2
Downloading botocore-1.39.2-py3-none-any.whl (13.8 MB)
| 13.8 MB 16.8 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
| 85 kB 5.6 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.2->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.2 botocore-1.39.2 s3transfer-0.13.0
Defaulting to user installation because normal site-packages is not writeable
Collecting python-dotenv
Downloading python_dotenv-1.1.1-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.1
[ec2-user@ip-172-31-28-8 ~]$ git clone https://github.com/Chandra123-123/Homemade.git
Cloning into 'Homemade'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (79/79), done.
remote: Total 85 (delta 23), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (85/85), 734.70 KiB | 25.33 MiB/s, done.
Resolving deltas: 100% (23/23), done.
[ec2-user@ip-172-31-28-8 ~]$
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
30°C
Mostly cloudy
Search
ENG
IN
12:43
03-07-2025

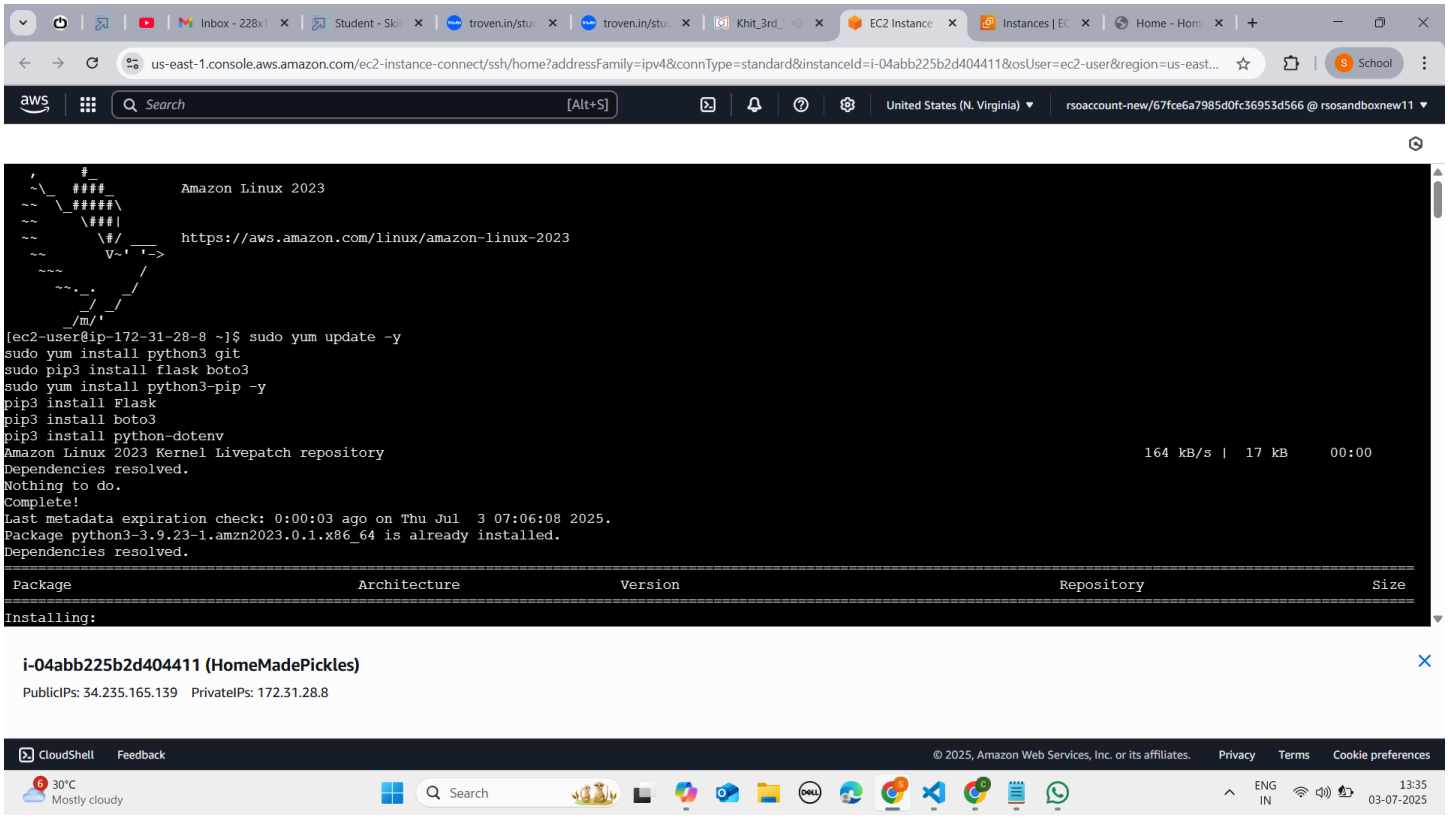

us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
| 13.8 MB 16.8 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
| 85 kB 5.6 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.2->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.2->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.2 botocore-1.39.2 s3transfer-0.13.0
Defaulting to user installation because normal site-packages is not writeable
Collecting python-dotenv
Downloading python_dotenv-1.1.1-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.1
[ec2-user@ip-172-31-28-8 ~]$ git clone https://github.com/Chandra123-123/Homemade.git
Cloning into 'Homemade'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (79/79), done.
remote: Total 85 (delta 23), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (85/85), 734.70 KiB | 25.33 MiB/s, done.
Resolving deltas: 100% (23/23), done.
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
30°C
Mostly cloudy
Search
ENG
IN
13:11
03-07-2025

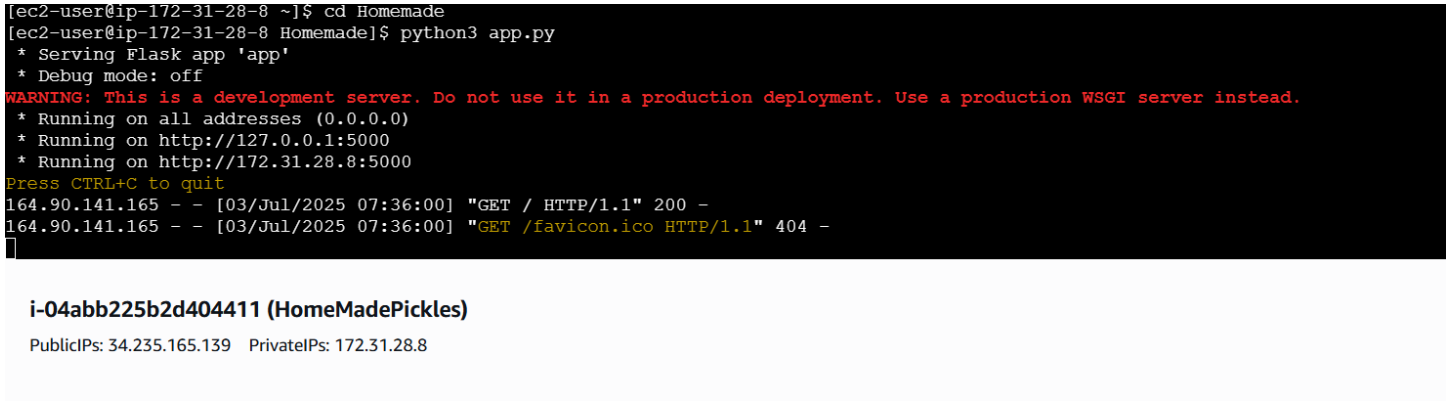
```
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
```

i-04abb225b2d404411 (HomeMadePickles)

PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8



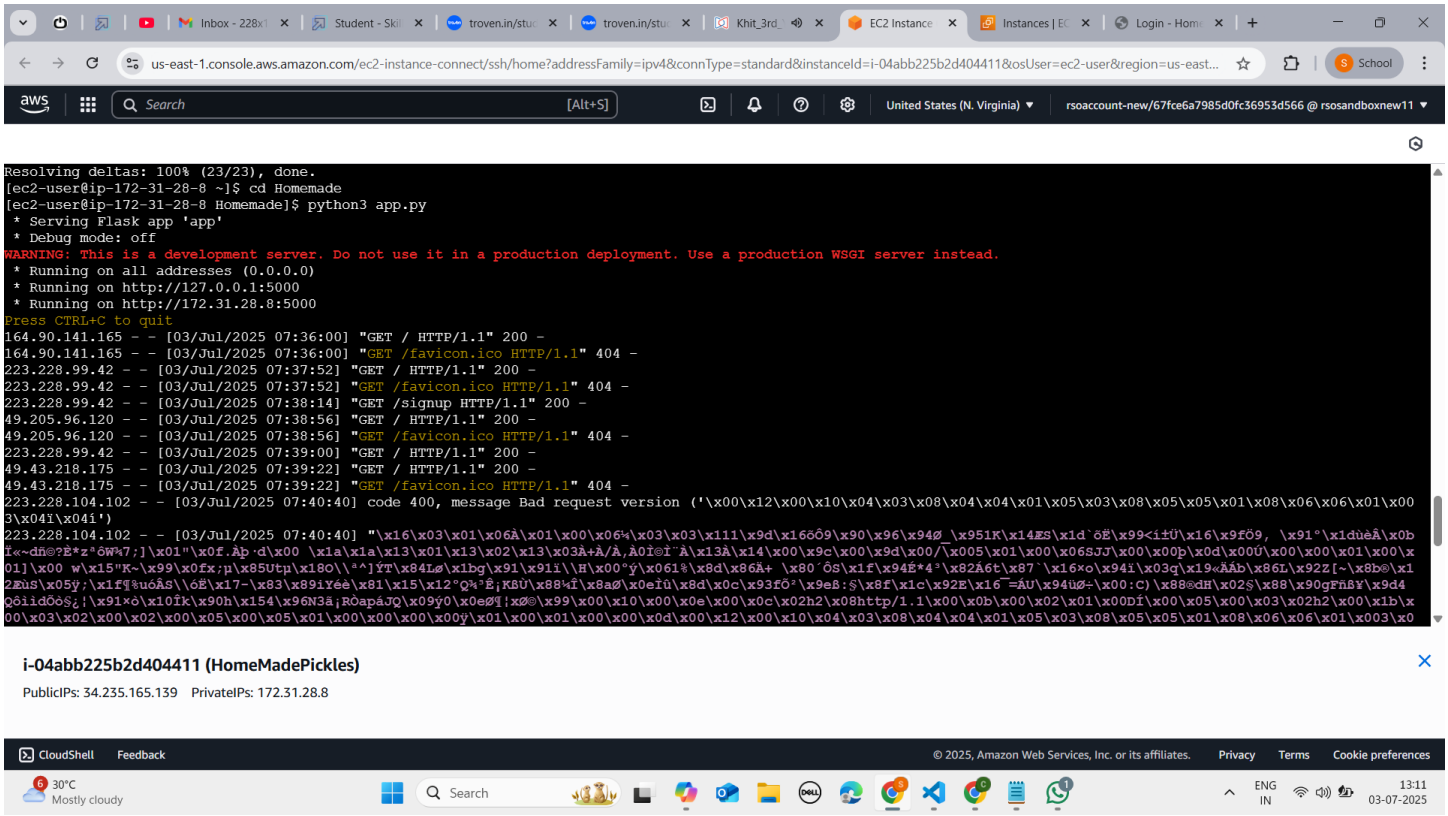




Verify the Flask app is running:

http://34.235.165.139

- Run the Flask app on the EC2 instance


Inbox - 228x
Student - Skil
troven.in/stuc
troven.in/stuc
Khit_3rd_
EC2 Instance
Instances | EC
Login - Home
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-04abb225b2d404411&osUser=ec2-user&region=us-east...
School
aws
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew11
Resolving deltas: 100% (23/23), done.
[ec2-user@ip-172-31-28-8 ~]$ cd Homemade
[ec2-user@ip-172-31-28-8 Homemade]$ python3 app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.28.8:5000
Press CTRL+C to quit
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET / HTTP/1.1" 200 -
164.90.141.165 - - [03/Jul/2025 07:36:00] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:37:52] "GET / HTTP/1.1" 200 -
223.228.99.42 - - [03/Jul/2025 07:37:52] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:38:14] "GET /signup HTTP/1.1" 200 -
49.205.96.120 - - [03/Jul/2025 07:38:56] "GET / HTTP/1.1" 200 -
49.205.96.120 - - [03/Jul/2025 07:38:56] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.42 - - [03/Jul/2025 07:39:00] "GET / HTTP/1.1" 200 -
49.43.218.175 - - [03/Jul/2025 07:39:22] "GET / HTTP/1.1" 200 -
49.43.218.175 - - [03/Jul/2025 07:39:22] "GET /favicon.ico HTTP/1.1" 404 -
223.228.104.102 - - [03/Jul/2025 07:40:40] code 400, message Bad request version ('\x00\x12\x00\x10\x04\x03\x08\x04\x04\x01\x05\x03\x08\x05\x05\x01\x08\x06\x06\x01\x003\x04ï\x04í')
223.228.104.102 - - [03/Jul/2025 07:40:40] "\x16\x03\x01\x06Â\x01\x00\x06¾\x03\x03\x111\x9d\x16ôÔ9\x90\x96\x94Ø_\x951K\x14ÆS\x1d`õÊ\x99<íŧÚ\x16\x9fÖ9, \x91°\x1dùèÂ\x0bÍ«~dñ@?È*z³ôW¾7;]\x01"\x0f.Àþ·d\x00 \x1a\x1a\x13\x01\x13\x02\x13\x03À+Ã/Ã,À0Ì@Ï¨Ã\x13Ã\x14\x00\x9c\x00\x9d\x00/\x005\x01\x00\x06SJ7\x00\x00þ\x0d\x00Ú\x00\x00\x01\x00\x01]\x00 w\x15"K~\x99\x0fx;µ\x85Utµ\x18O\\ª^]ÝT\x84Lø\xlbg\x91\x91ï\\H\x00°ý\x061*\x8d\x86Ã+ \x80´ÓS\x1f\x94É*4³\x82Â6t\x87`\x16×o\x94ï\x03q\x19«ÃÀb\x86L\x92Z[~\x8b®\x12ÆùS\x05ÿ;\x1f¶%uóÃS\\óË\x17-\x83\x89iÝée\x81\x15\x12°Q¾³È¡KßÙ\x88¼Í\x8aØ\x0eÍû\x8d\x0c\x93fÔ²\x9eß:§\x8f\x1c\x92E\x16¯=ÃU\x94üØ÷\x00:C)\x88®dH\x02§\x88\x90gFñß¥\x9d4Qôiid©ò§¿¦\x91×ò\x10Îk\x90h\x154\x96N3ã¡RÒapáJQ\x09ý0\x0eØ¶¦xØ®\x99\x00\x10\x00\x0e\x00\x0c\x02h2\x08http/1.1\x00\x0b\x00\x02\x01\x00Dí\x00\x05\x00\x03\x02h2\x00\x1b\x00\x03\x02\x00\x02\x00\x05\x00\x05\x01\x00\x00\x00\x00ÿ\x01\x00\x01\x00\x00\x0d\x00\x12\x00\x10\x04\x03\x08\x04\x04\x01\x05\x03\x08\x05\x05\x01\x08\x06\x06\x01\x003\x0
i-04abb225b2d404411 (HomeMadePickles)
PublicIPs: 34.235.165.139 PrivateIPs: 172.31.28.8
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
30°C
Mostly cloudy
Search
ENG
IN
13:11
03-07-2025

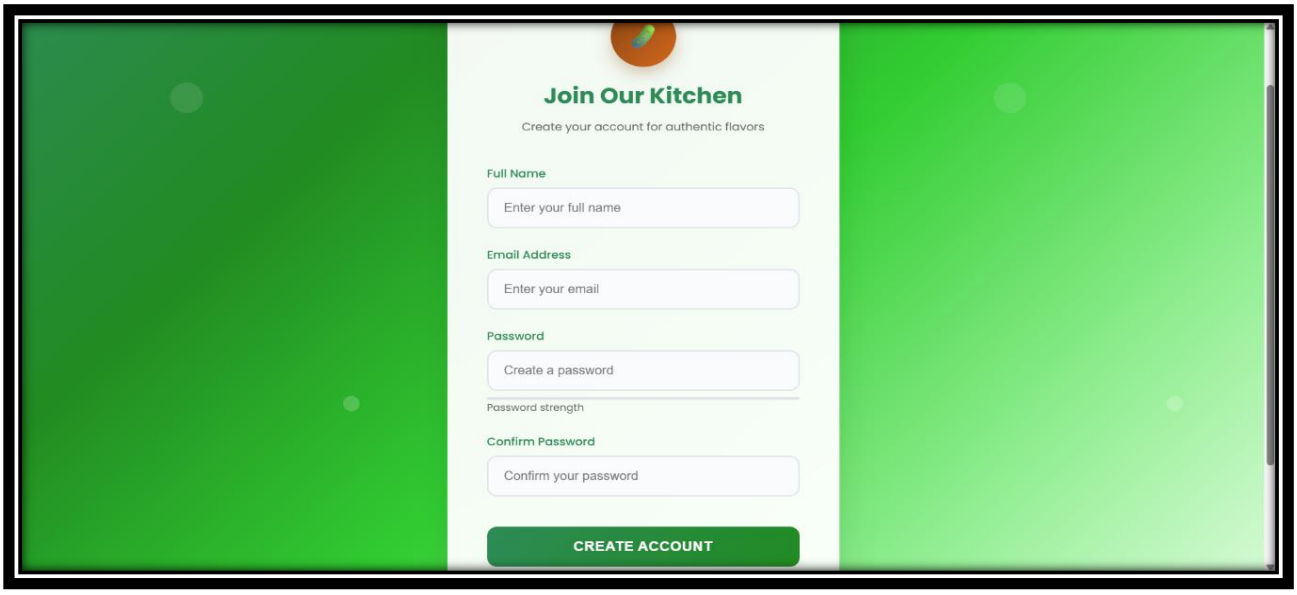
Access the website through:

Public IPs: http://34.235.165.139

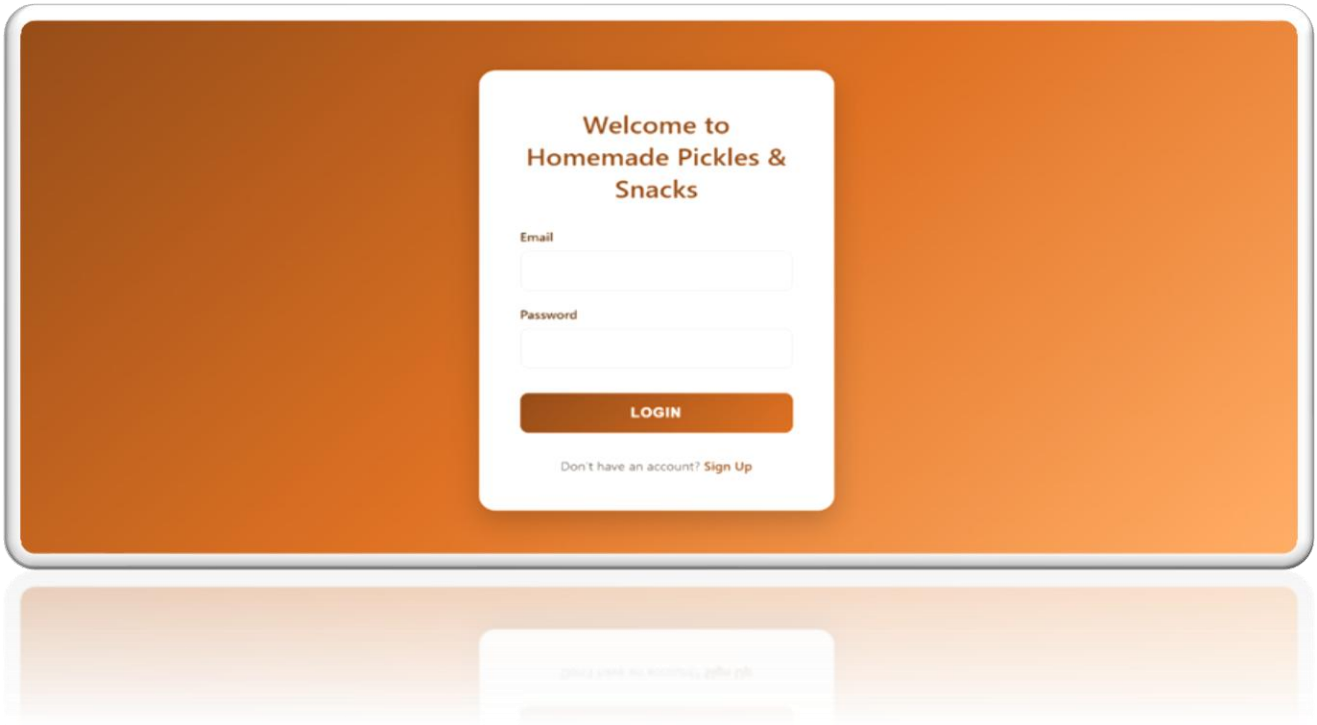
Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user signup, login, pickles requests, and notifications.**

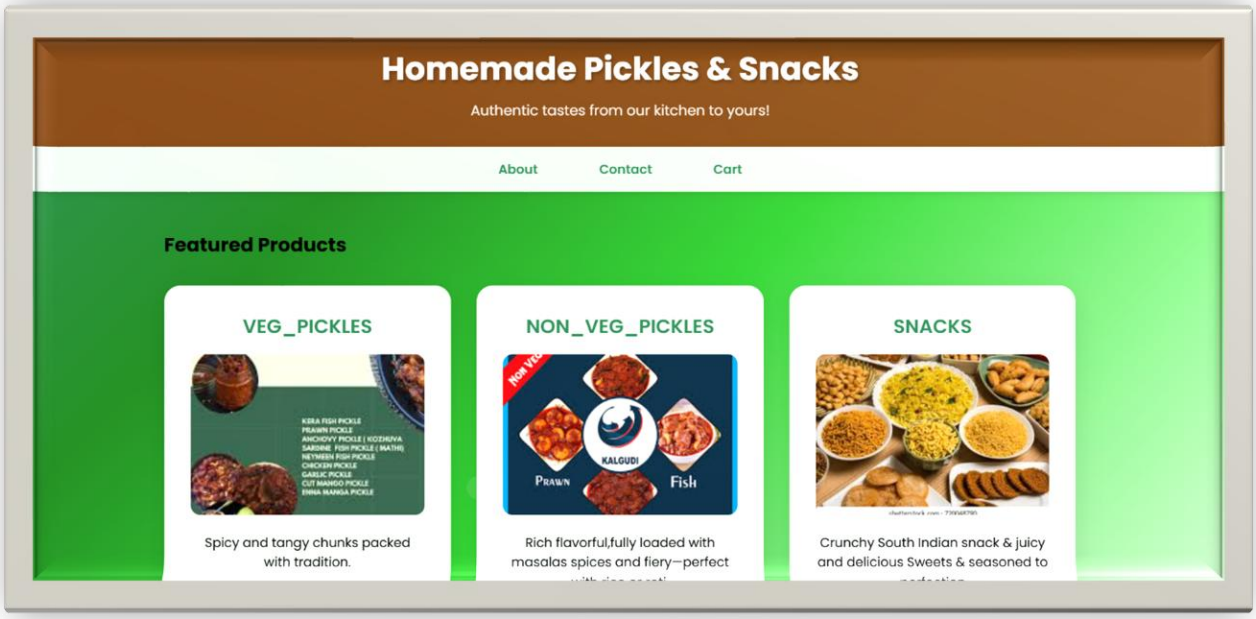
Signup Page:



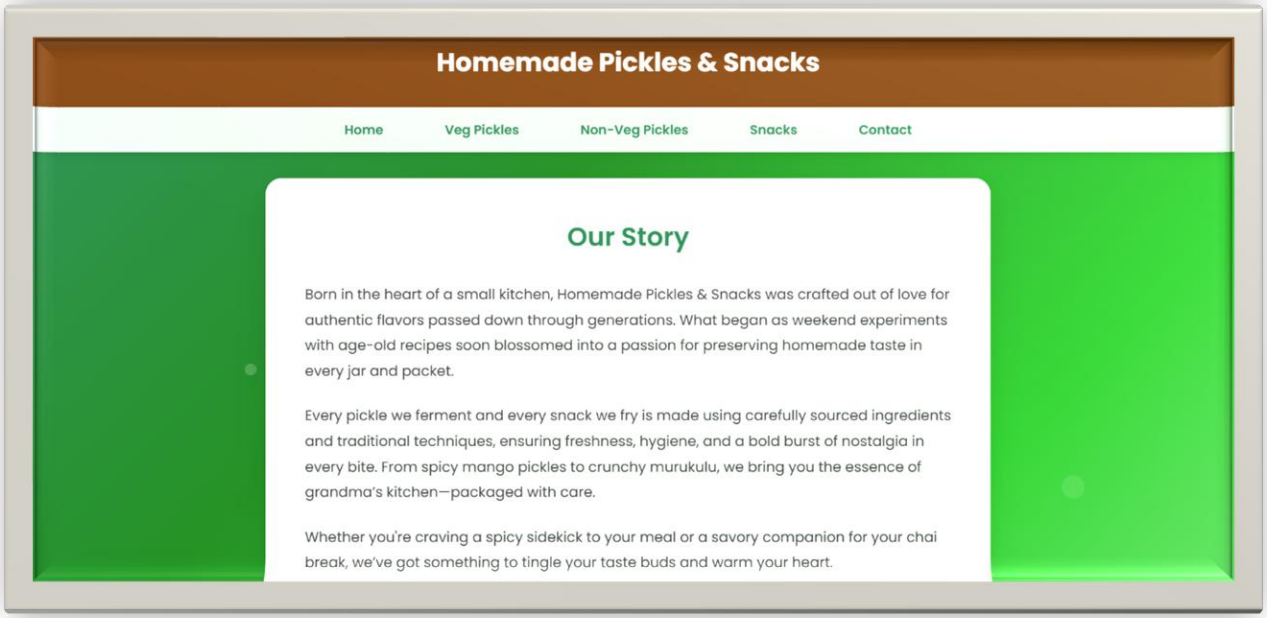
Login Page:


Welcome to Homemade Pickles & Snacks
Email
Password
LOGIN
Don't have an account? Sign Up

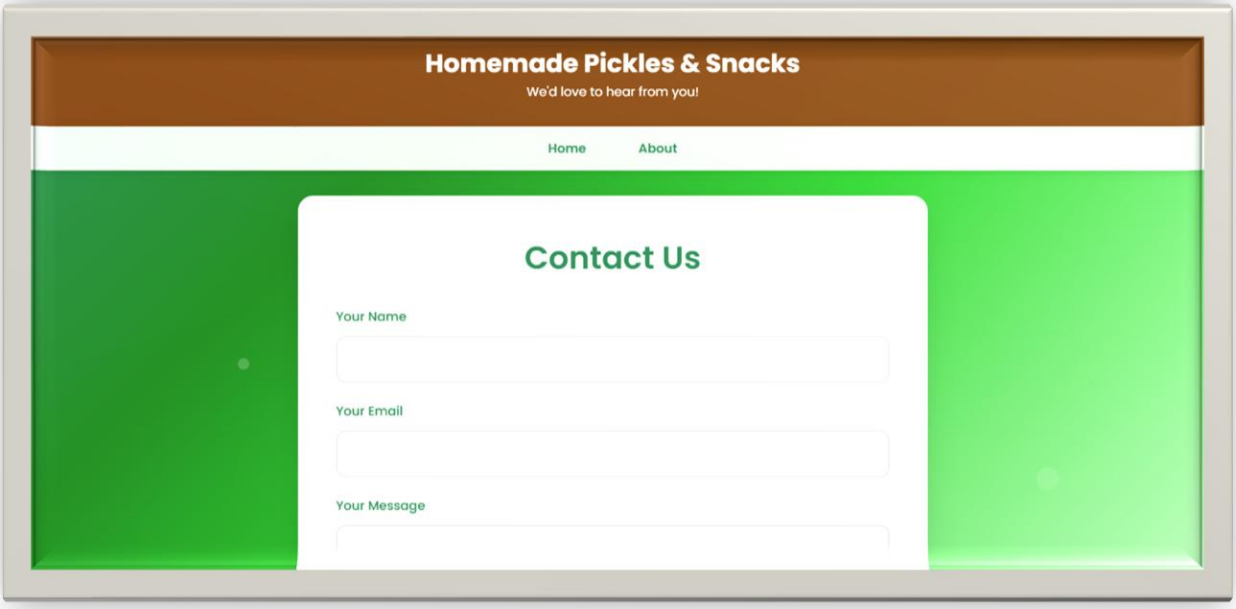
Home page:


Homemade Pickles & Snacks
Authentic tastes from our kitchen to yours!
About
Contact
Cart
Featured Products
VEG_PICKLES
Spicy and tangy chunks packed with tradition.
NON_VEG_PICKLES
KALGUDI
Prawn
Fish
Rich flavorful,fully loaded with masalas spices and fiery—perfect
SNACKS
Crunchy South Indian snack & juicy and delicious Sweets & seasoned to

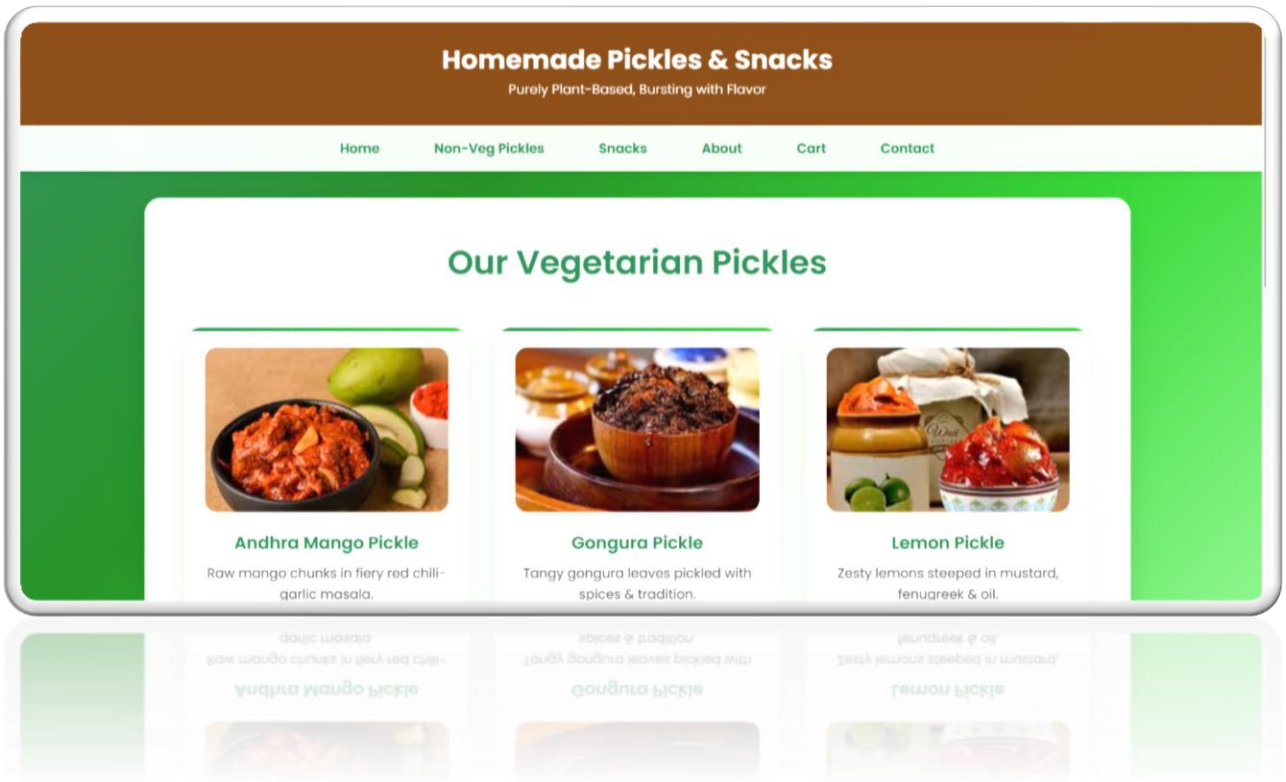
About Us page:



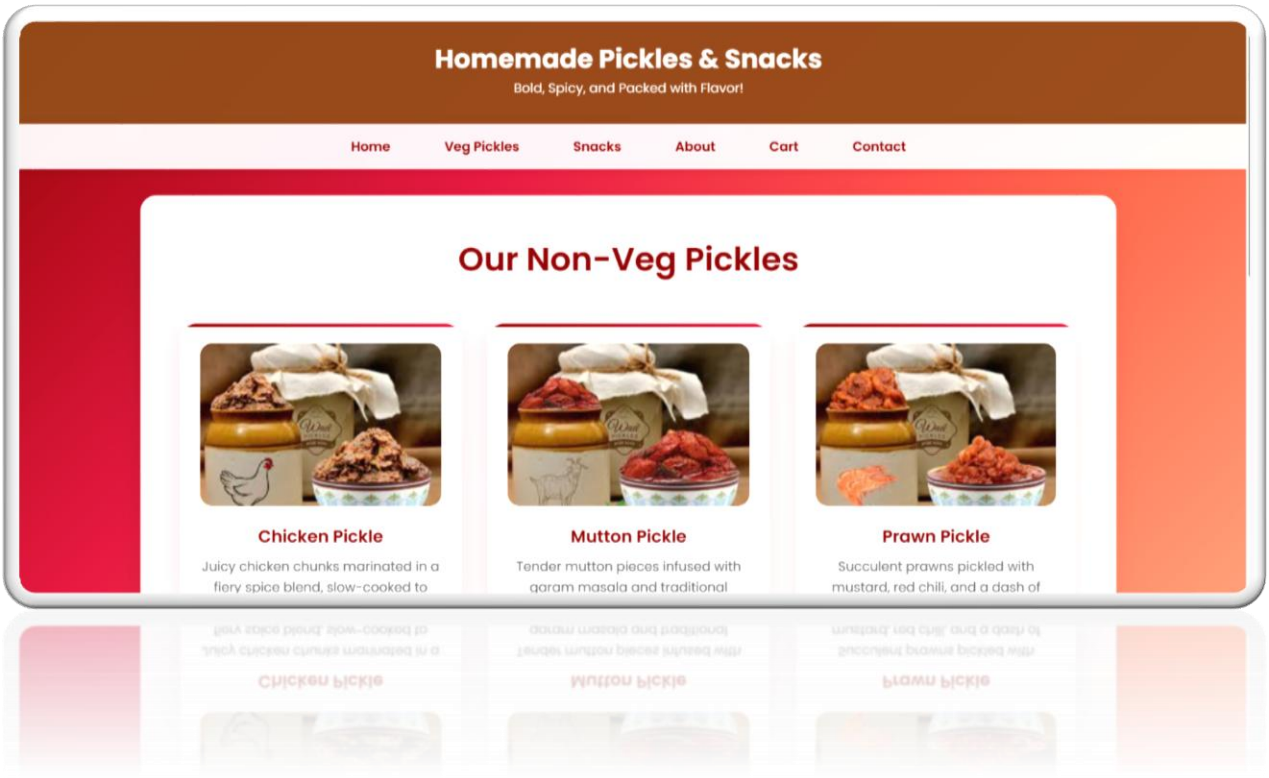
Contact Page:



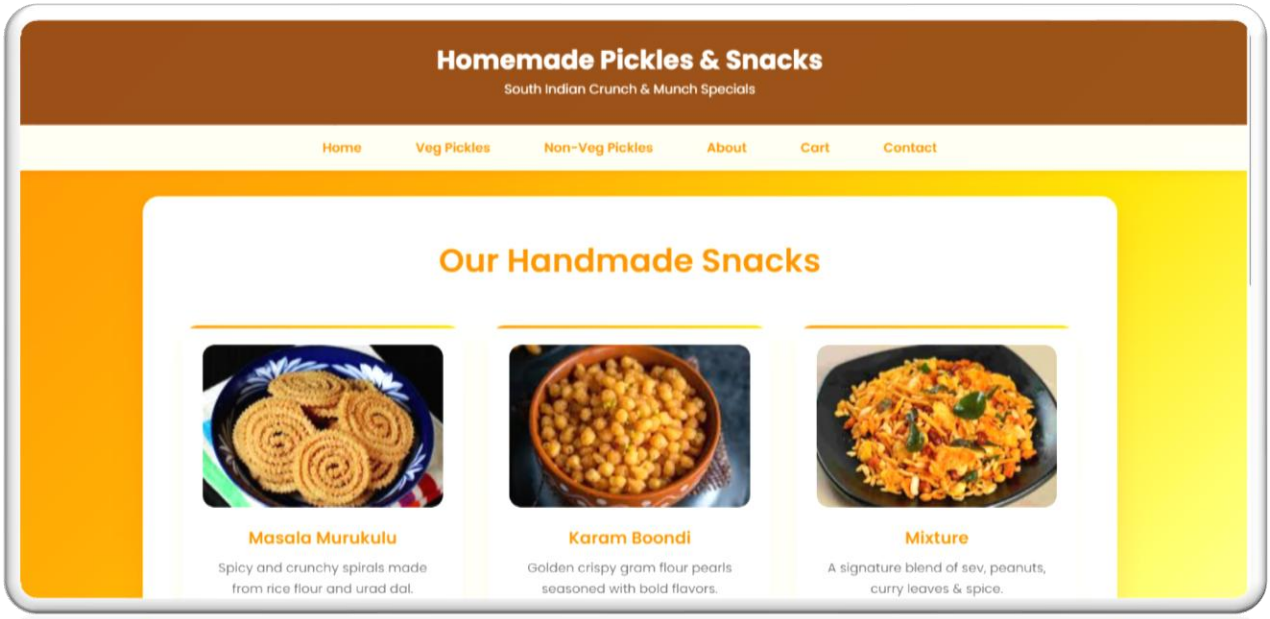
Veg-Pickles page :



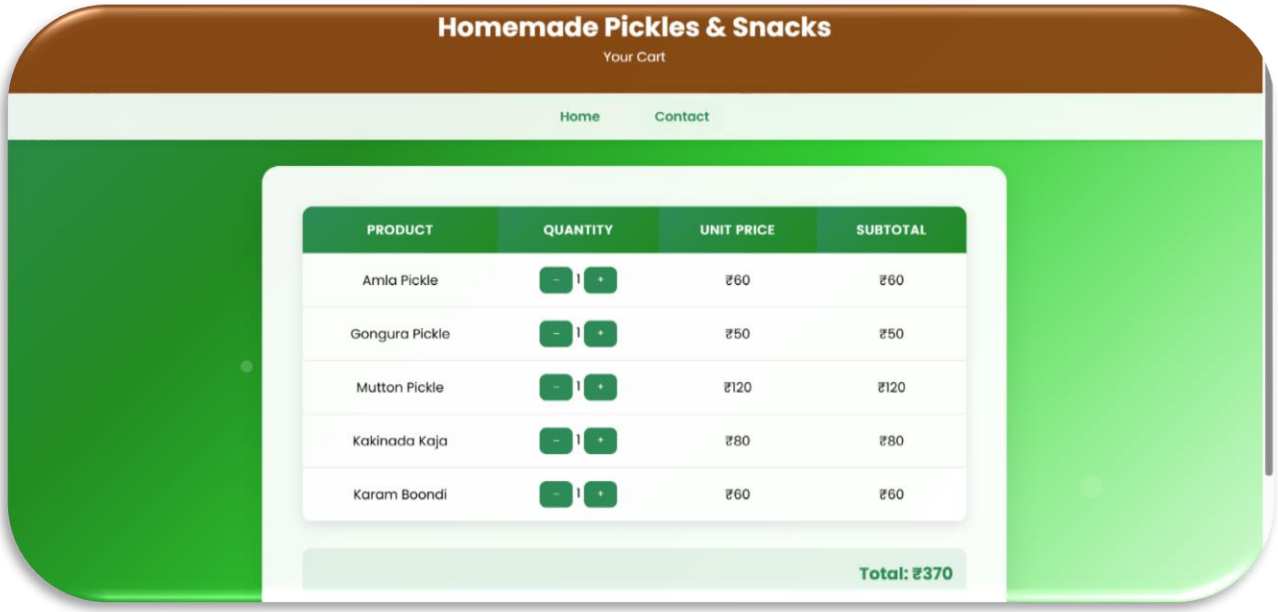
Non-Veg-Pickles page:



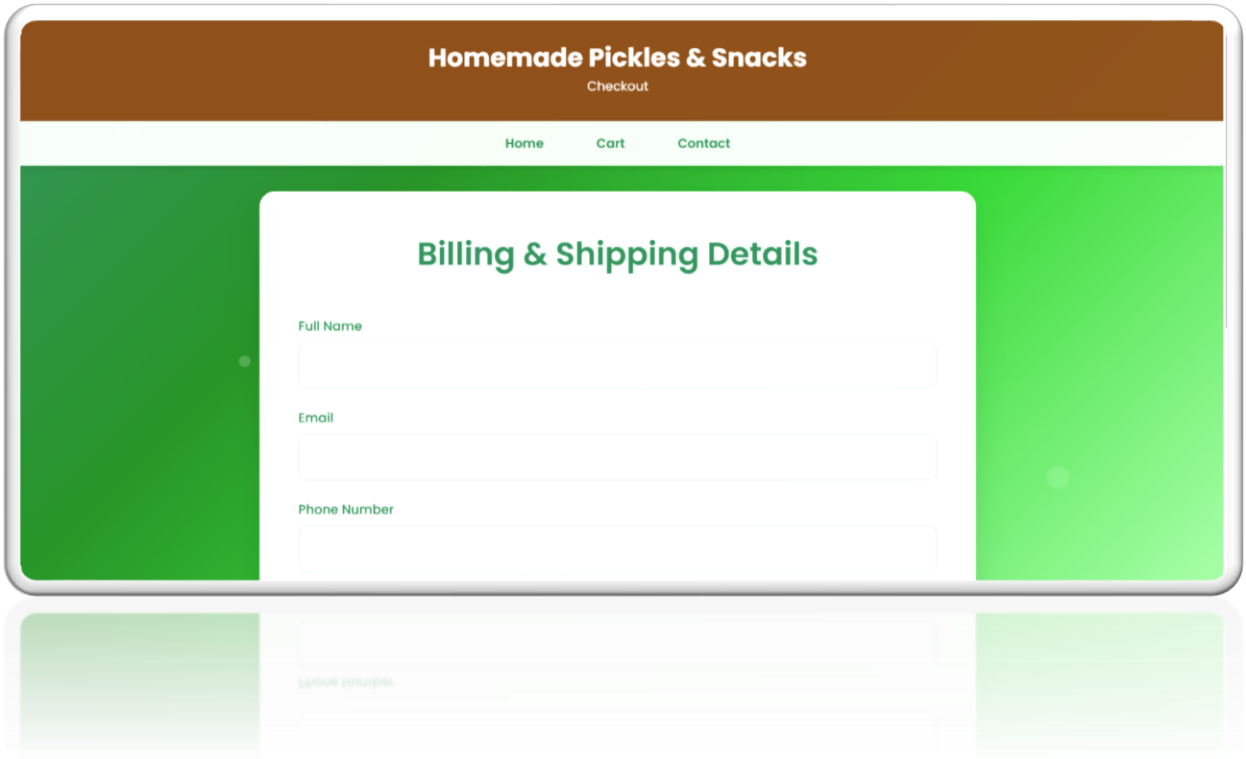
Snacks Page:


Homemade Pickles & Snacks
South Indian Crunch & Munch Specials
Home
Veg Pickles
Non-Veg Pickles
About
Cart
Contact
Our Handmade Snacks
Masala Murukulu
Spicy and crunchy spirals made from rice flour and urad dal.
Karam Boondi
Golden crispy gram flour pearls seasoned with bold flavors.
Mixture
A signature blend of sev, peanuts, curry leaves & spice.

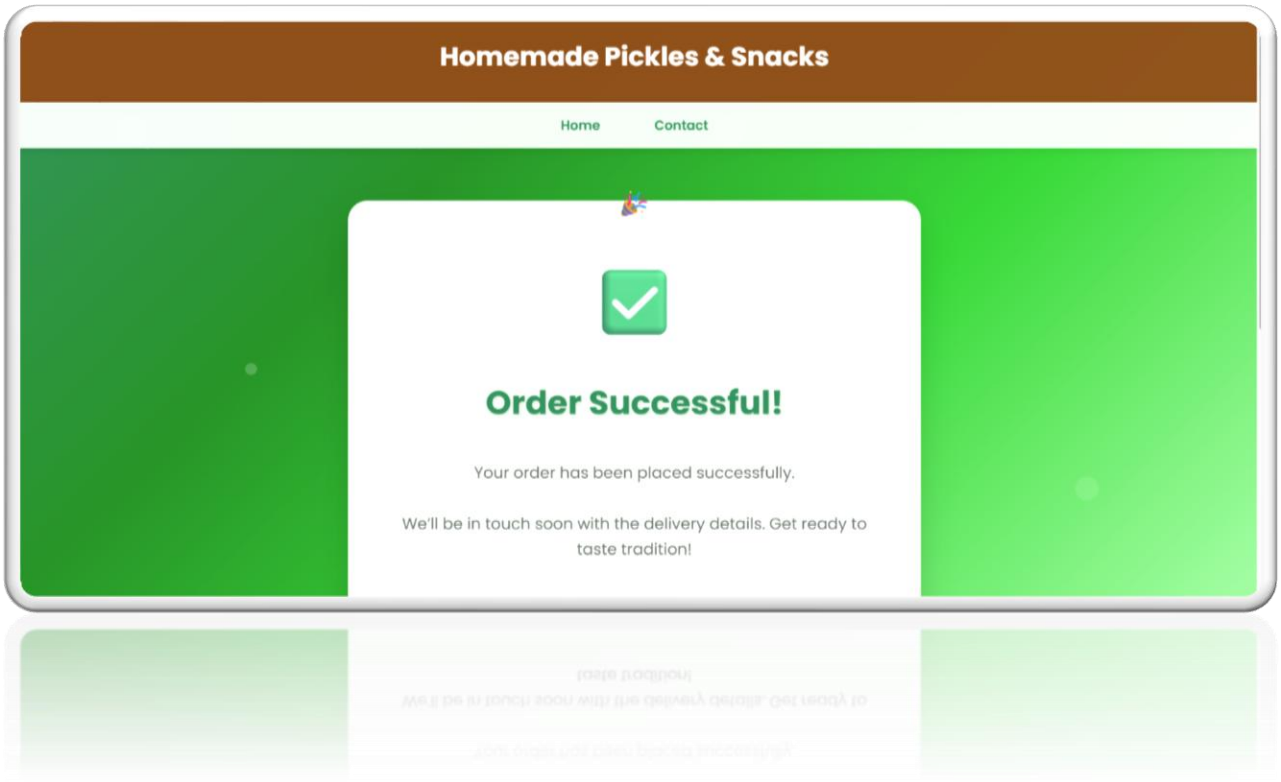
Cart Page:


Homemade Pickles & Snacks
Your Cart
Home
Contact
PRODUCT
QUANTITY
UNIT PRICE
SUBTOTAL
Amla Pickle
₹60
₹60
Gongura Pickle
₹50
₹50
Mutton Pickle
₹120
₹120
Kakinada Kaja
₹80
₹80
Karam Boondi
₹60
₹60
Total: ₹370

Check-out Page:

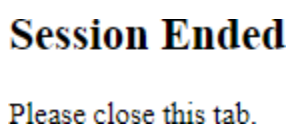


Success Page:



Your order has been placed successfully! We Will get backto you soon

Exit:



Conclusion:

The Homemade Pickles and Snacks Website has been successfully developed and deployed using a robust cloud-native architecture to ensure high performance, scalability, and customer satisfaction. By leveraging **AWS EC2** for reliable hosting, **DynamoDB** for secure product and order data management, and **SNS** for real-time customer and staff notifications, the platform delivers a seamless end-to-end experience for home made food lovers. This system addresses the need for efficient online access to traditional food offerings, allowing customers to conveniently browse, order, and track their favourite pickles and snacks. The cloud infrastructure enables the site to scale effortlessly during high-demand periods like festive seasons or promotions, without compromising responsiveness or user experience.

The integration of **Flask with AWS services** ensures that backend operations—such as order processing, inventory updates, and customer messaging—function smoothly in real time. Thorough testing has validated the stability of all core features, from product browsing and secure checkout to order confirmation notifications. In conclusion, this website serves as a modern platform that blends tradition with technology, providing a delightful shopping experience while streamlining business operations. It stands as a strong example of how cloud-based solutions can elevate local, handmade products to a broader digital audience