Pickle Paradise: Cloud-Based Online Pickle & Snack Store

Project Description

Pickle Paradise is an e-commerce platform designed to cater to customers who love traditional Indian pickles and snacks. With increasing demand and the desire to reach a wider audience, the platform was developed using Flask for backend logic, AWS EC2 for deployment, and DynamoDB for scalable database operations. AWS SNS handles real-time customer notifications for orders and inquiries. The system allows users to browse, register, log in, add items to the cart, and place orders—all through a user-friendly web interface.

Scenarios

1. Seamless Online Ordering

Using Flask and EC2, customers can explore a curated selection of pickles (veg, non-veg) and snacks, view prices and images, and place orders with just a few clicks. DynamoDB stores user data and order history, ensuring scalability and speed.

2. Instant Notifications via AWS SNS

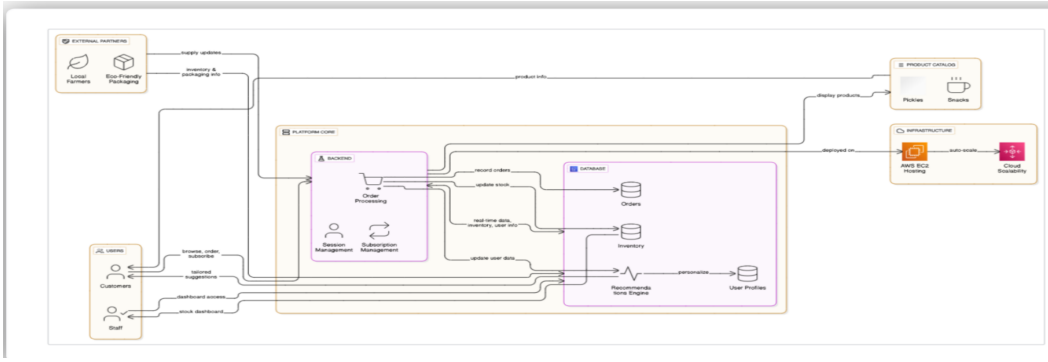
Whenever a user places an order, AWS SNS sends confirmation emails both to the customer and the admin. Flask backend handles the logic while SNS ensures users are updated instantly about their order status.

3. Easy Inventory & Category Management

Admins can view and manage inventory (manually or via additional interface), while customers easily switch between pages like Snacks, Veg Pickles, Non-Veg Pickles, Cart, and Checkout. Flask handles page routing and real-time data rendering.

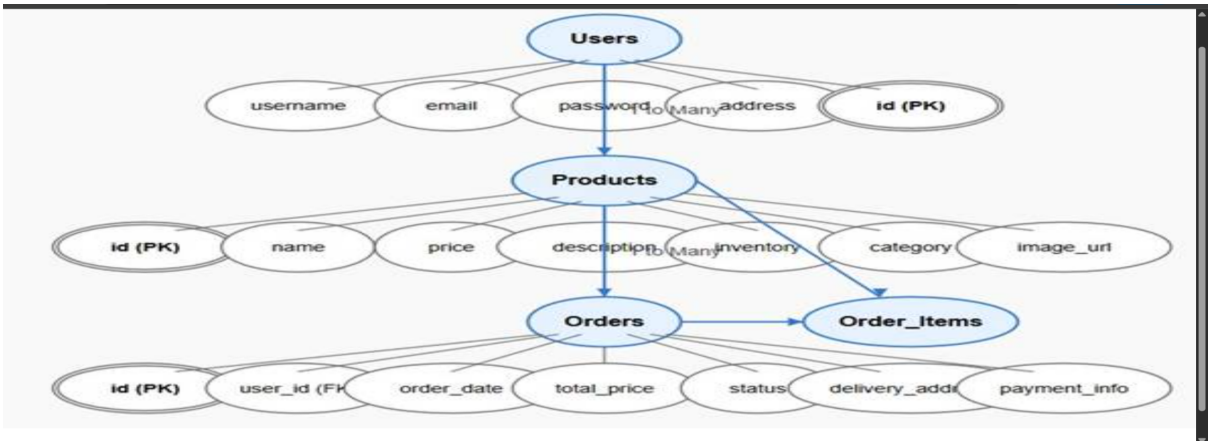
AWS Architecture

The diagram below represents the complete architecture of the Pickle Paradise system:



Entity Relationship (ER) Diagram

The ER diagram below outlines the structure of the application's database:



Pre-requisites

1.AWS Account Setup
2. IAM Configuration
3. Amazon EC2 Basics
4. DynamoDB Basic
5. AWS SNS Setup
6. Git Version Control Basics

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 Book 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

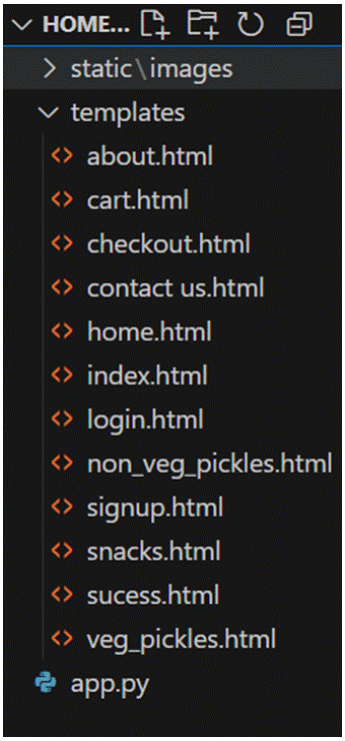
Milestone 8. Testing and Deployment

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

Milestone 1. Backend Development and Application Setup

Develop the Backend Using Flask.

File Explorer Structure:



Description of the code :

? Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for, session
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
from datetime import datetime
import json,uuid
```

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

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

```python
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')  # e.g., 'us-east-1'
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

```python
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': 'Gongura Prawns', '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': 'Kakarakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 11, 'name': 'Chintakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 12, 'name': 'Spicy Pandu Mirchi', 'weights': {'250': 130, '500': 240, '1000': 450}}
    ],  # Add your veg pickle products here
    'snacks': [
        {'id': 7, 'name': 'Banana Chips', 'weights': {'250': 300, '500': 600, '1000': 800}},
        {'id': 8, 'name': 'Crispy Aam-Papad', 'weights': {'250': 150, '500': 300, '1000': 600}},
        {'id': 9, 'name': 'Crispy Chekka Pakodi', 'weights': {'250': 50, '500': 100, '1000': 200}},
        {'id': 10, 'name': 'Boondhi Acchu', 'weights': {'250': 300, '500': 600, '1000': 900}},
        {'id': 11, 'name': 'Chekkalu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 12, 'name': 'Ragi Laddu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 13, 'name': 'Dry Fruit Laddu', 'weights': {'250': 500, '500': 1000, '1000': 1500}},
        {'id': 14, 'name': 'Kara Boondi', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 15, 'name': 'Gavvalu', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 16, 'name': 'Kaju Chikki', '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('/')
def index():
    return render_template('index.html')

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

        try:
            # Fetch user from DynamoDB
            response = users_table.get_item(Key={'username': username})

            if 'Item' not in response:
                return render_template('login.html', error='User not found')

            user = response['Item']

            # Ensure password field exists in the DB
            if 'password' not in user:
                return render_template('login.html', error='Password not found in database')

            # Verify password
            if check_password_hash(user['password'], password):
                session['logged_in'] = True
                session['username'] = username
                session.setdefault('cart', [])  # Initialize cart if not set
```

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

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

        try:
            # Check if username exists
            response = users_table.get_item(Key={'username': username})
            if 'Item' in response:
                return render_template('signup.html', error='Username already exists')

            # Hash password before storing
            hashed_password = generate_password_hash(password)

            # Store new user in DynamoDB
            users_table.put_item(
                Item={
                    'username': username,
                    'email': email,
                    'password': hashed_password  # Store hashed password
                }
            )

            return redirect(url_for('login'))

        except Exception as e:
            app.logger.error(f"Signup error: {str(e)}")
```

```
        except Exception as e:
            app.logger.error(f"Signup error: {str(e)}")
            return render_template('signup.html', error='Registration failed. Please try again.')

    return render_template('signup.html')

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

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

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

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

```
@app.route('/home')
def home():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('home.html')

@app.route('/non_veg_pickles')
def non_veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('non_veg_pickles.html', products=products['non_veg_pickles'])

@app.route('/veg_pickles')
def veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products['veg_pickles'])

@app.route('/snacks')
def snacks():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('snacks.html', products=products['snacks'])
```

- **Check out Route**:

```python
@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    error_message = None  # Variable to hold error messages

    if request.method == 'POST':
        try:
            # Extract form data safely
            name = request.form.get('name', '').strip()
            address = request.form.get('address', '').strip()
            phone = request.form.get('phone', '').strip()
            payment_method = request.form.get('payment', '').strip()

            # Validate inputs
            if not all([name, address, phone, payment_method]):
                return render_template('checkout.html', error="All fields are required.")

            if not phone.isdigit() or len(phone) != 10:
                return render_template('checkout.html', error="Phone number must be exactly 10 digits.")

            # Get cart data from hidden inputs
            cart_data = request.form.get('cart_data', '[]')
            total_amount = request.form.get('total_amount', '0')
```

Restart Visual Studio Code to apply the late

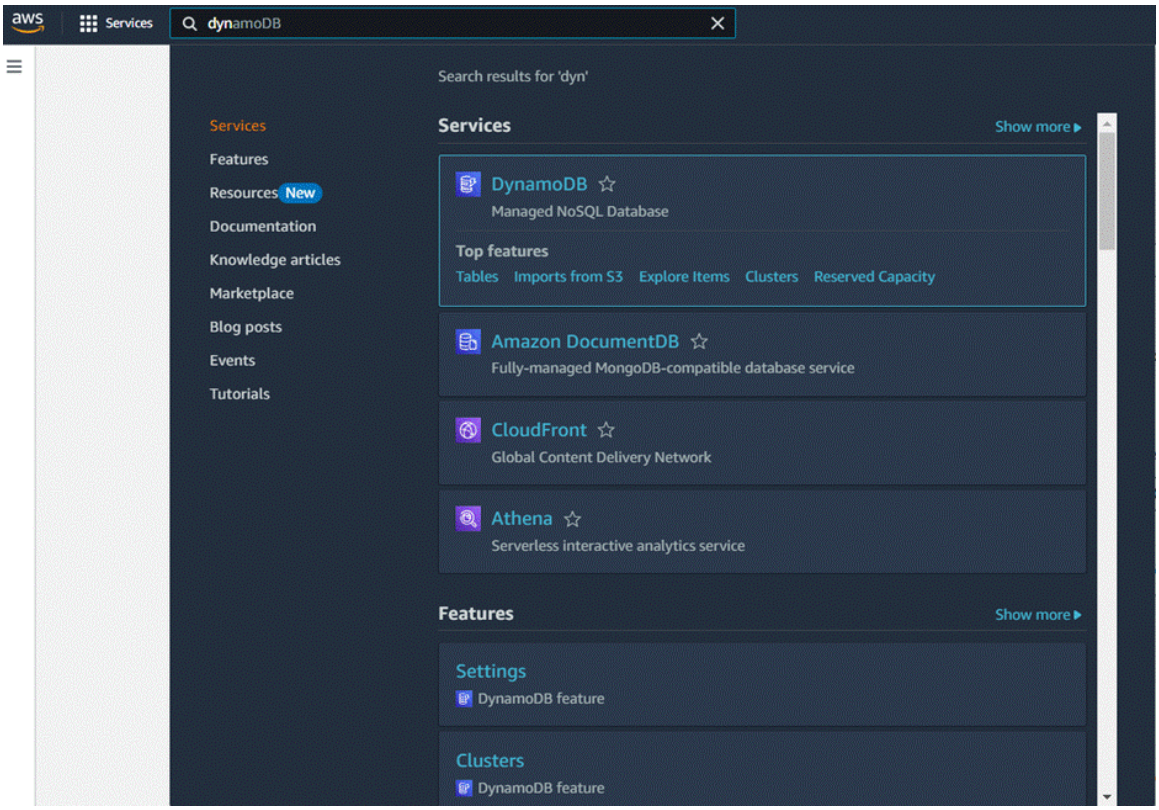
```python
            try:
                cart_items = json.loads(cart_data)
                total_amount = float(total_amount)
            except (json.JSONDecodeError, ValueError):
                return render_template('checkout.html', error="Invalid cart data format.")

            # Ensure cart is not empty
            if not cart_items:
                return render_template('checkout.html', error="Your cart is empty.")

            # Store order in DynamoDB
            try:
                orders_table.put_item(
                    Item={
                        'order_id': str(uuid.uuid4()),
                        'username': session.get('username', 'Guest'),
                        'name': name,
                        'address': address,
                        'phone': phone,
                        'items': cart_items,
                        'total_amount': total_amount,
                        'payment_method': payment_method,
                        'timestamp': datetime.now().isoformat()
                    }
                )
            except Exception as db_error:
                print(f"DynamoDB Error: {db_error}")
                return render_template('checkout.html', error="Failed to save order. Please try again later.")
```

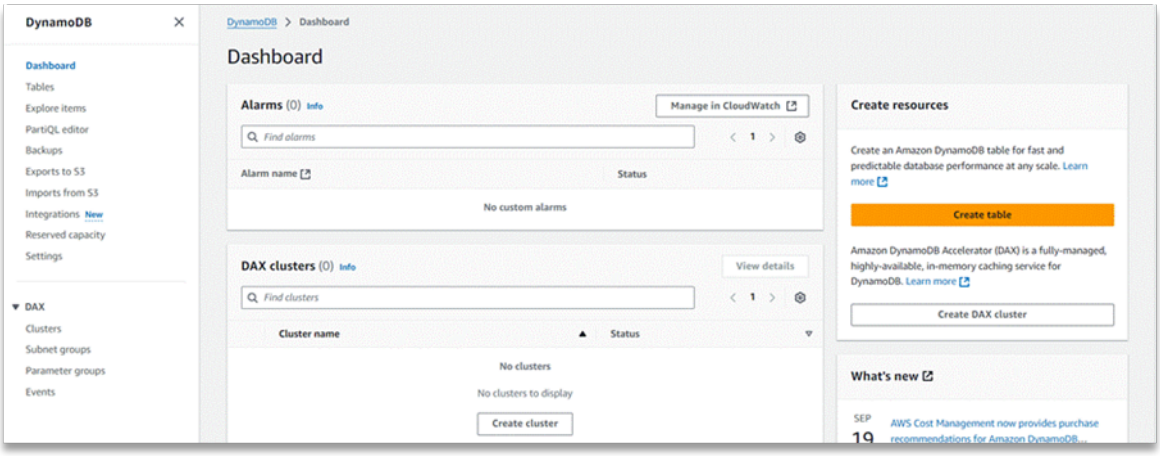
```python
            # Redirect to success page with success message
            return redirect(url_for('sucess', message="Your order has been placed successfully!"))

        except Exception as e:
            print(f"Checkout error: {str(e)}")
            return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

    return render_template('checkout.html')  # Render checkout page for GET request


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

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

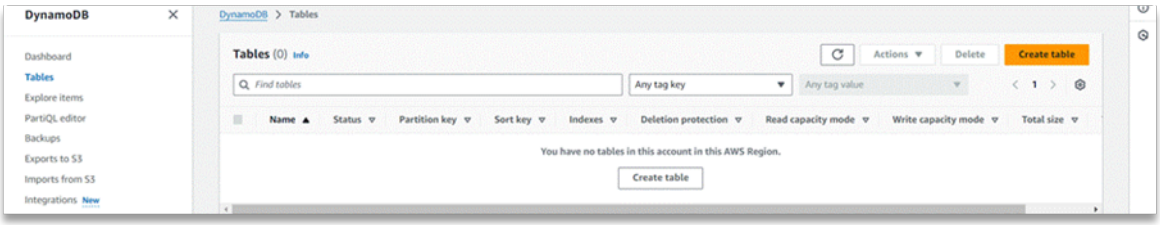
Milestone 2. AWS Account Setup and Login

Milestone 3 : DynamoDB Database Creation and Setup
Navigate to the DynamoDB

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

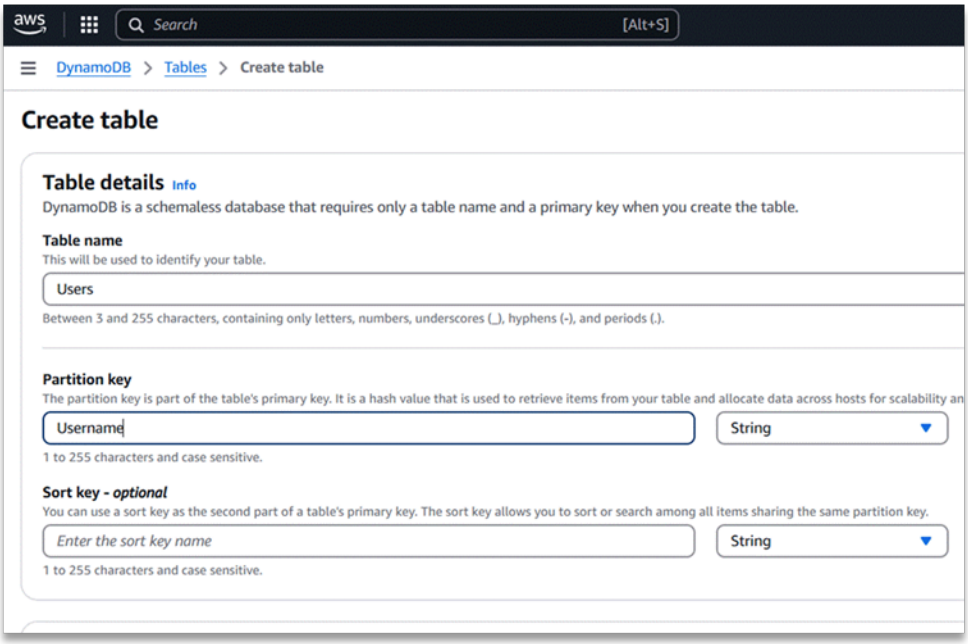


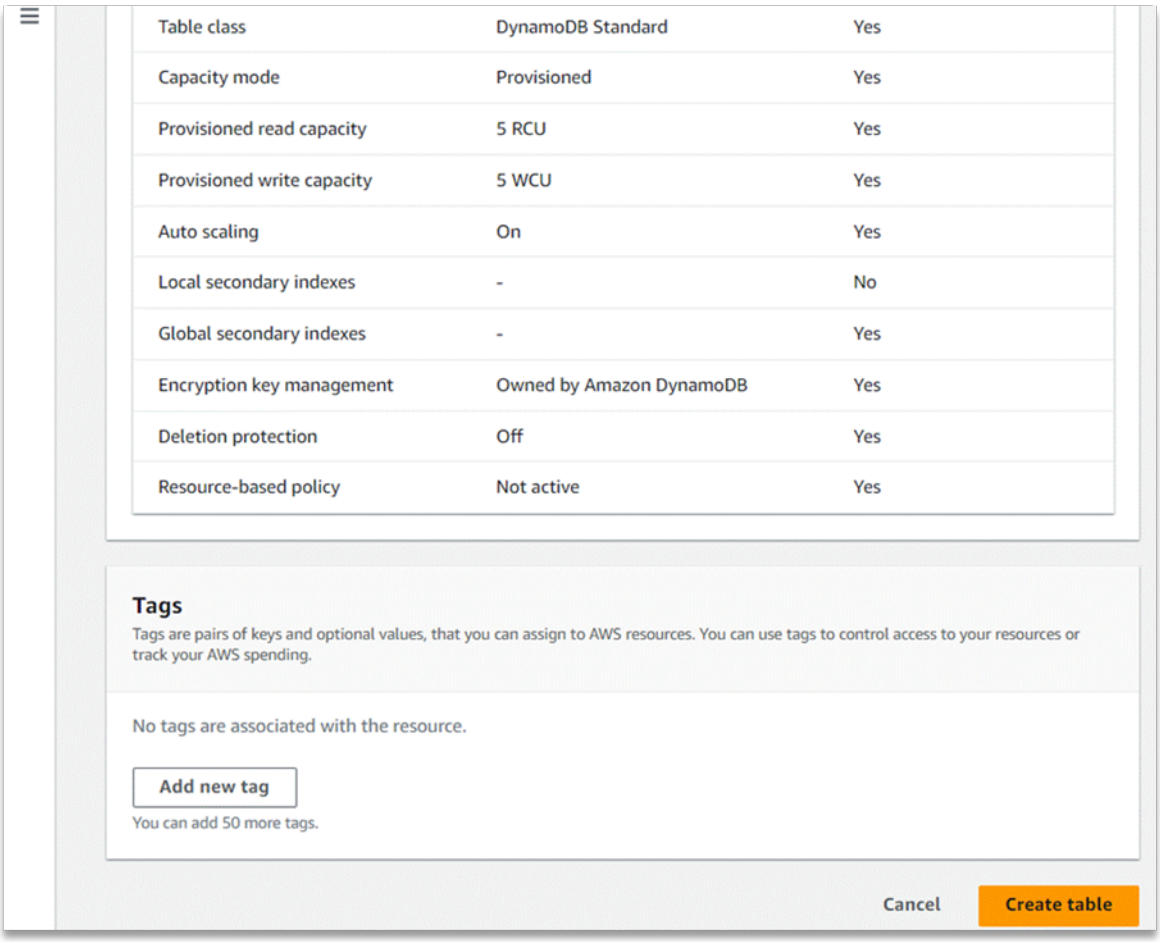


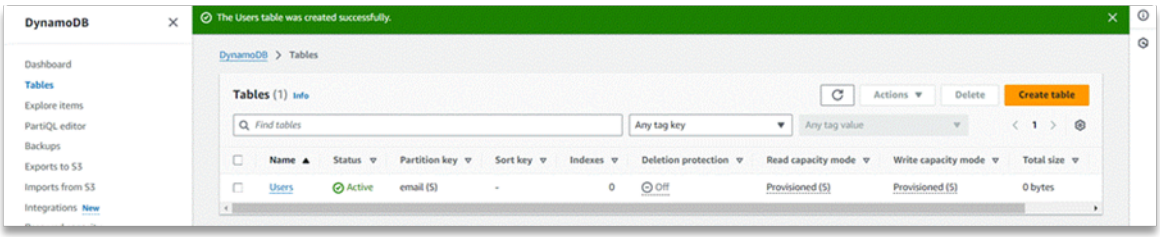


Create a DynamoDB table for storing data

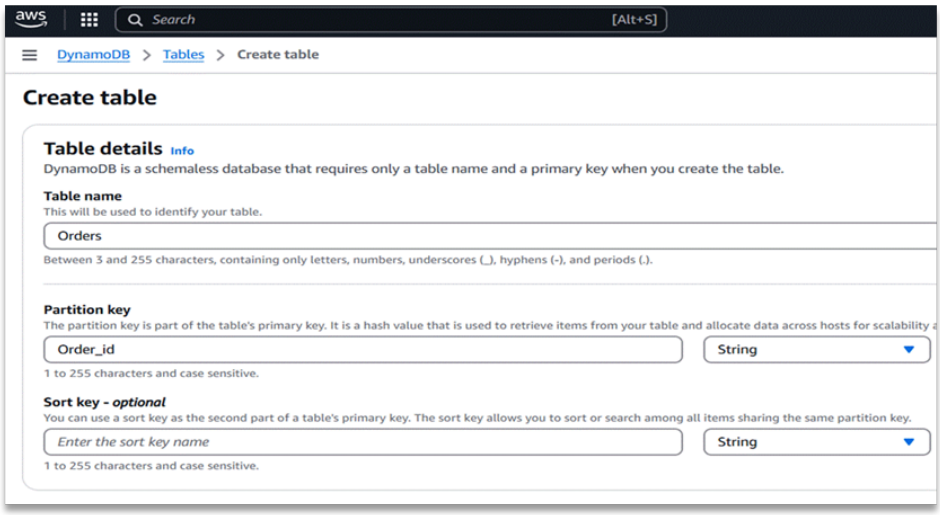
- Create Users table with partition key “Username” with type String and click on create tables.

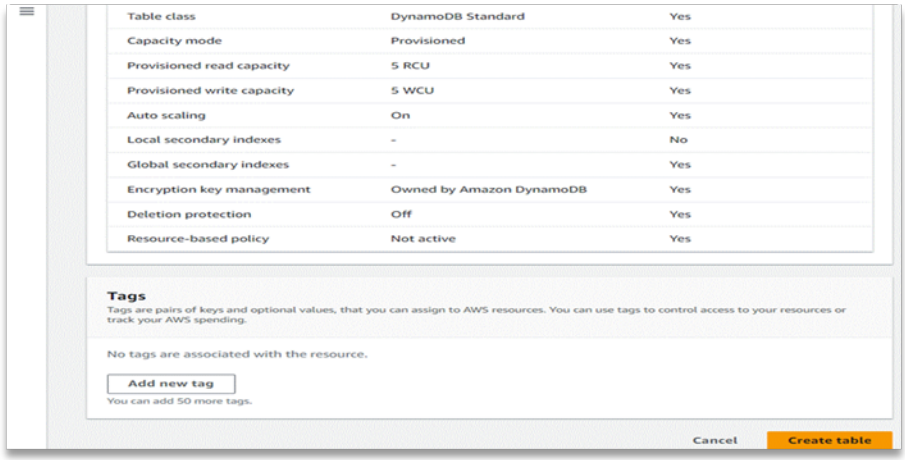


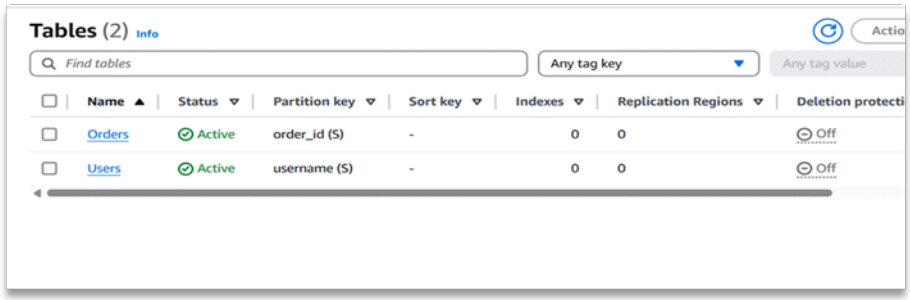




Follow the same steps to create an Orders table with Order_id as the primary key to store Order details.



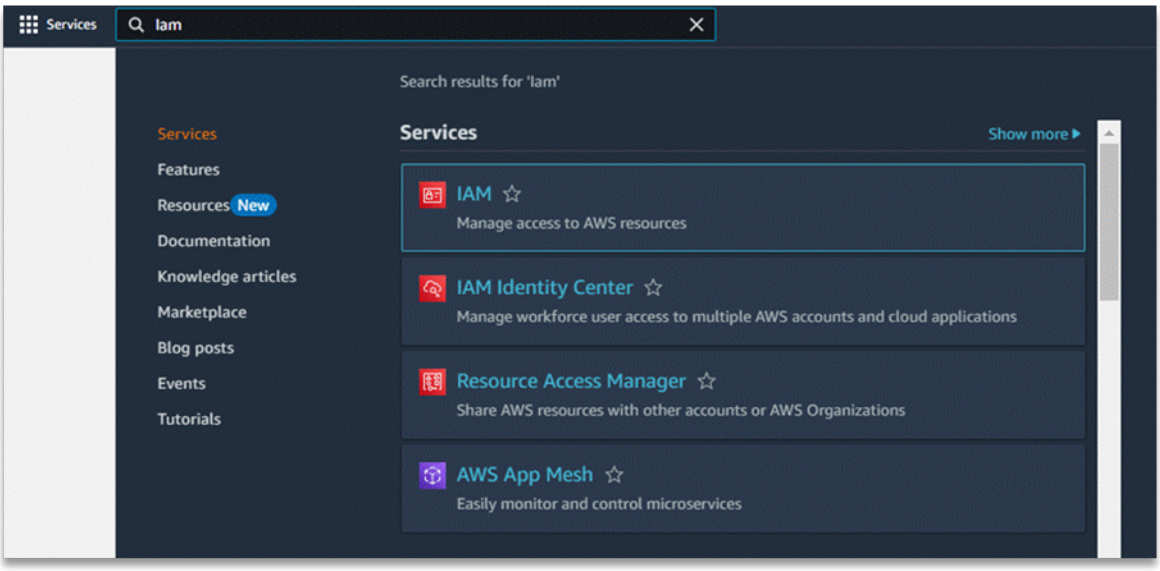


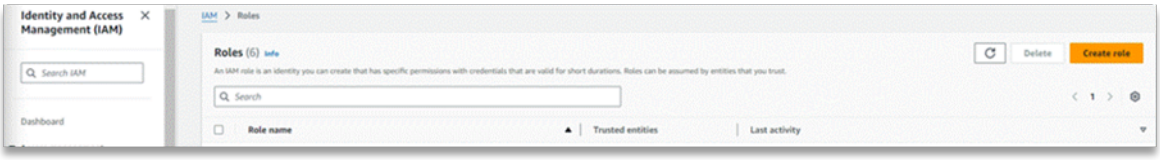


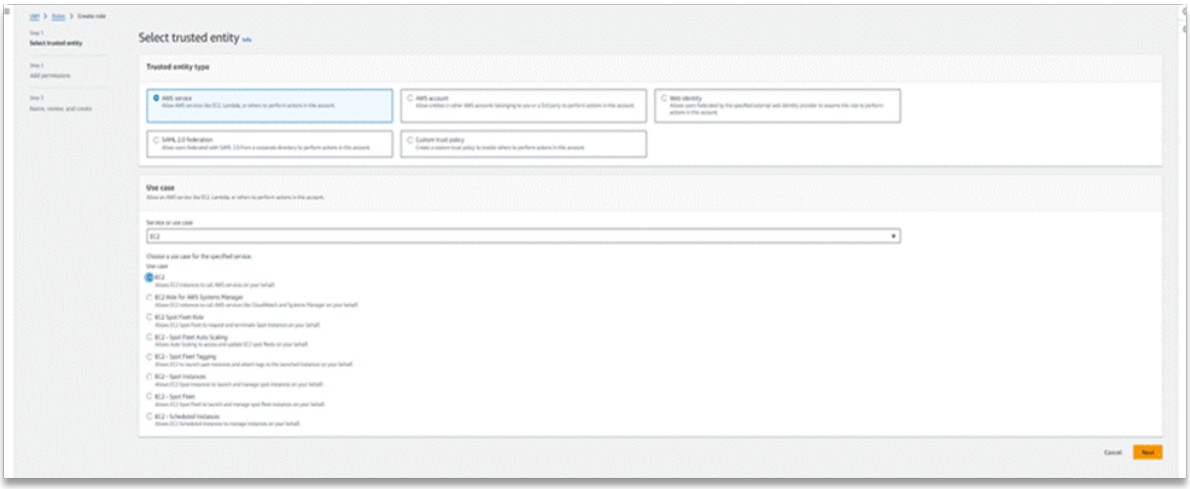
Milestone 4 : IAM Role Setup

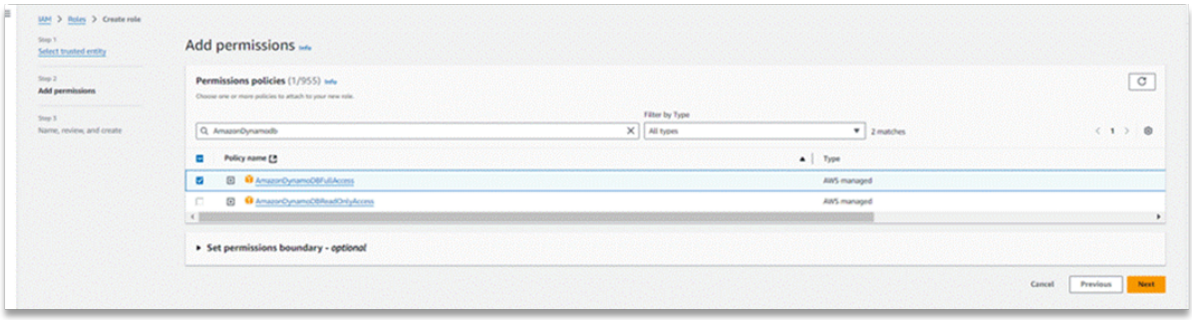
Create IAM Role.

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





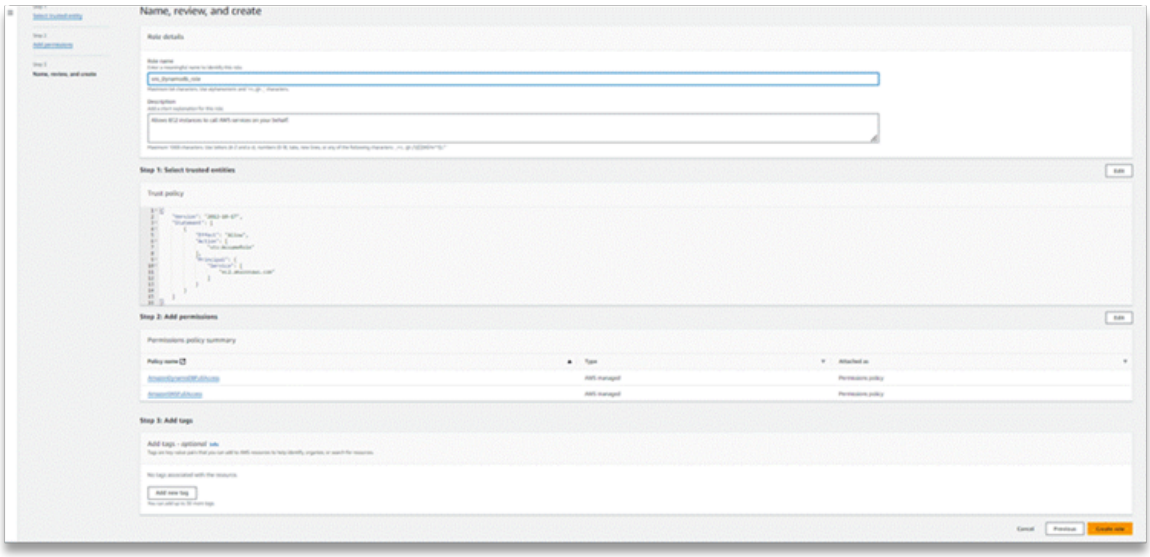


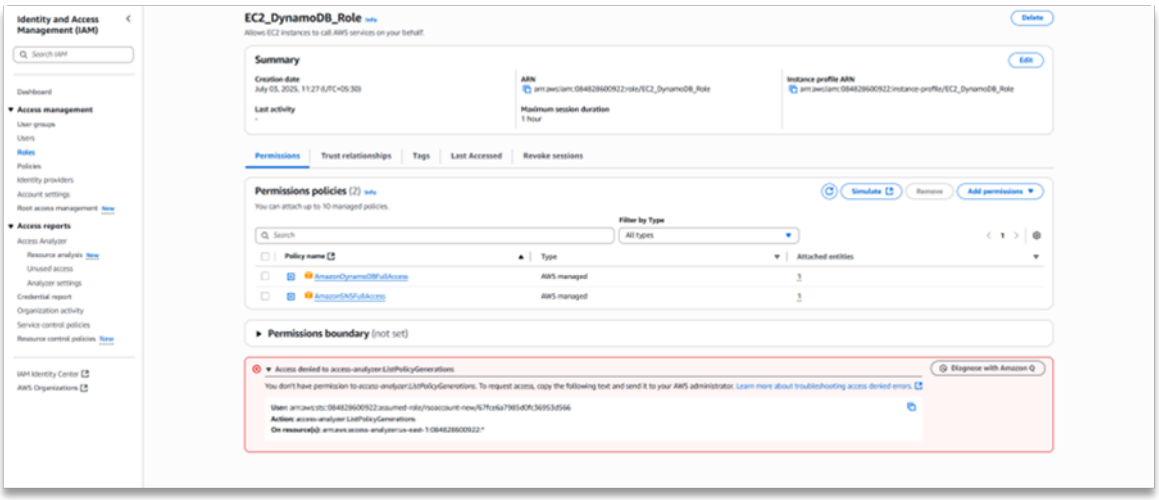


Attach Policies

Attach the following policies to the role:

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



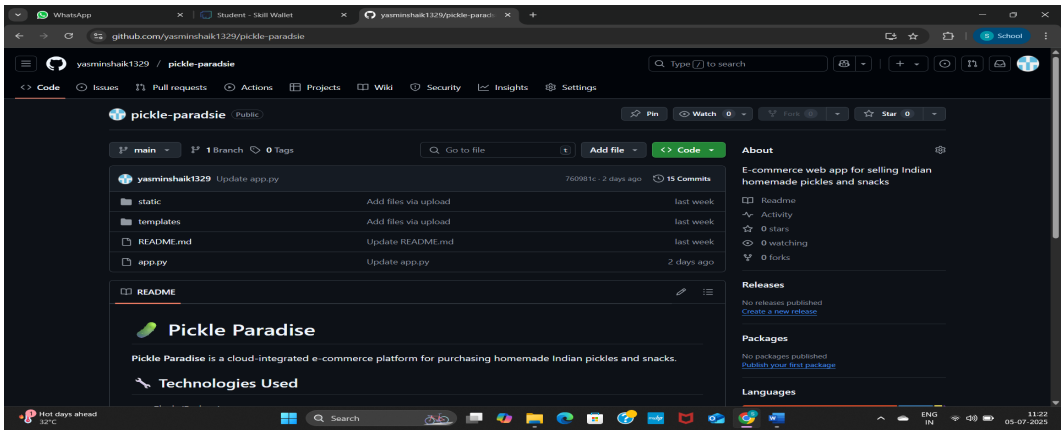


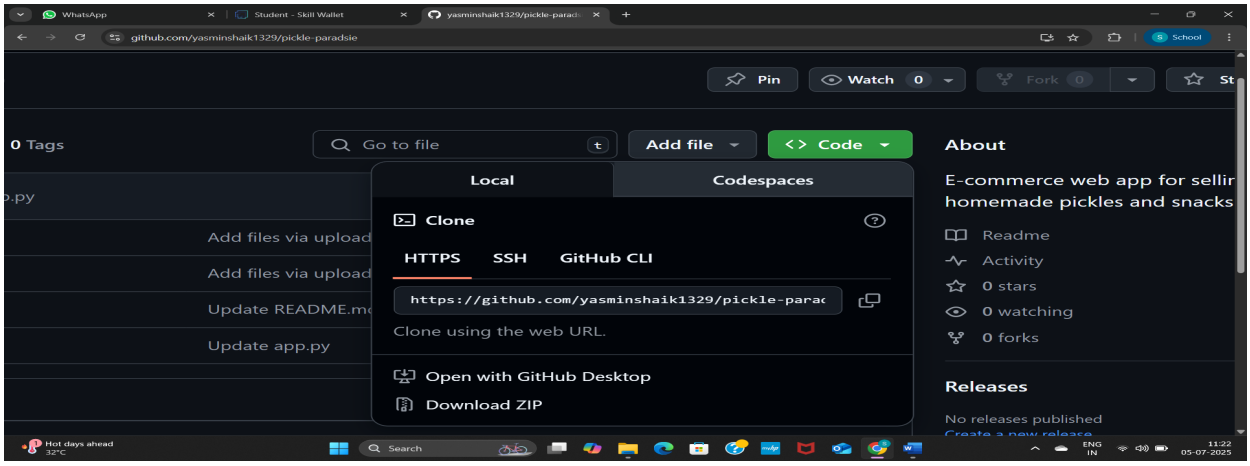
Milestone 5 : EC2 Instance Setup

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.

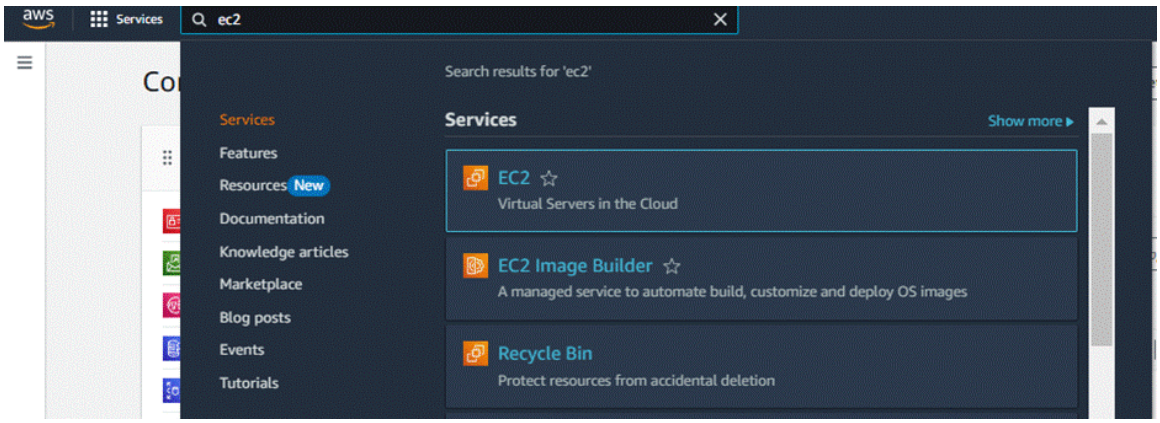
Load your Project Files to GitHub

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

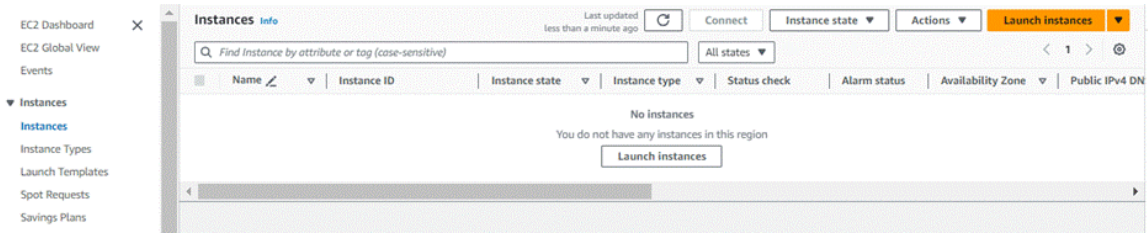


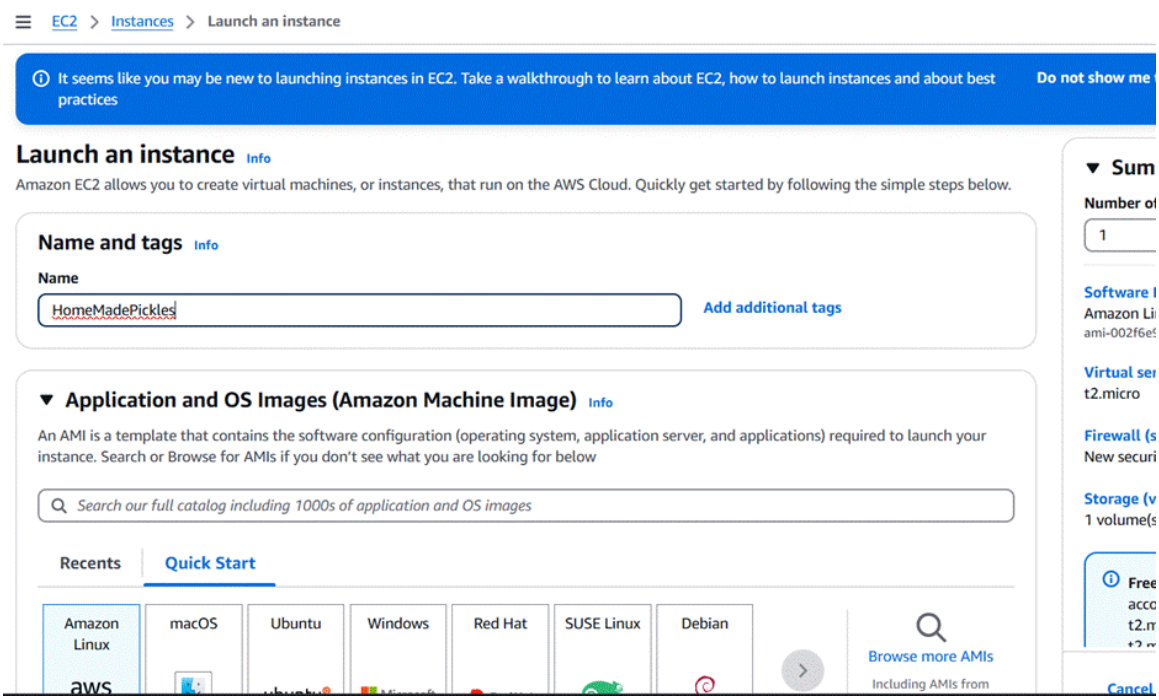


Launch an EC2 instance to host the Flask.

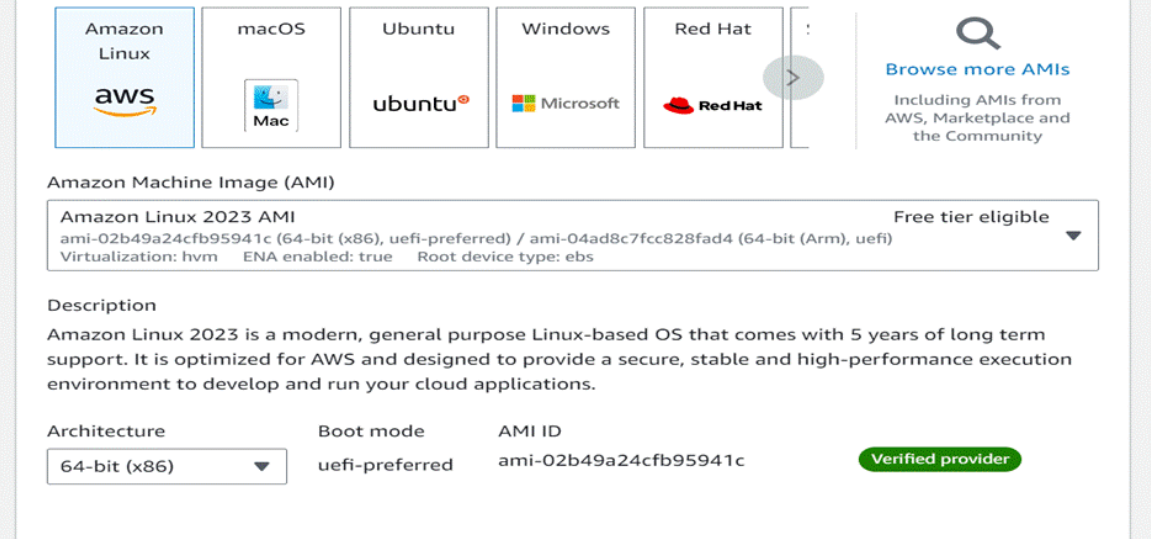


- **Click on Launch instance to launch EC2 instance**

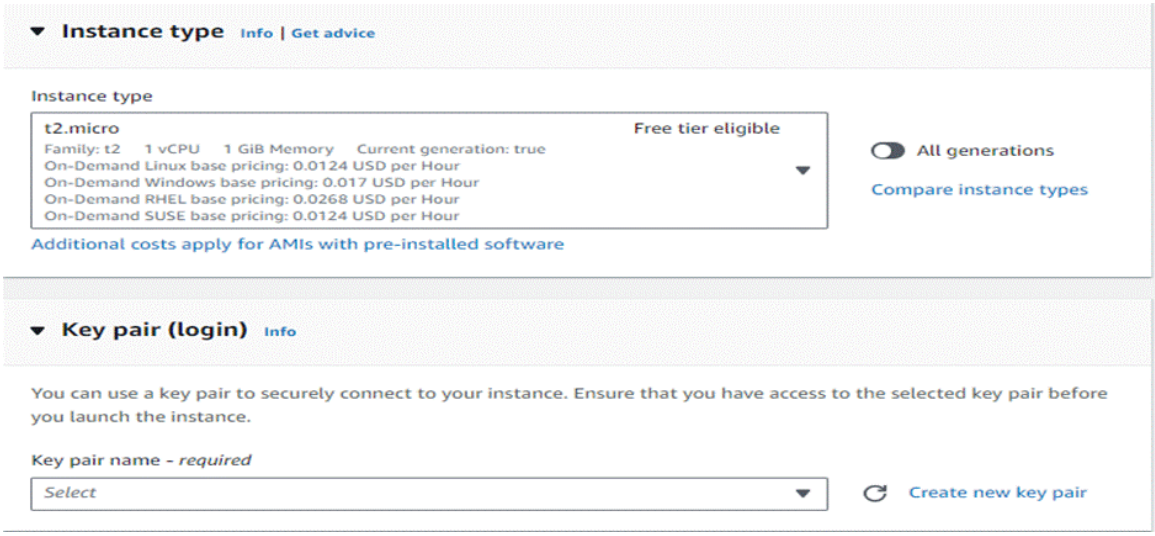


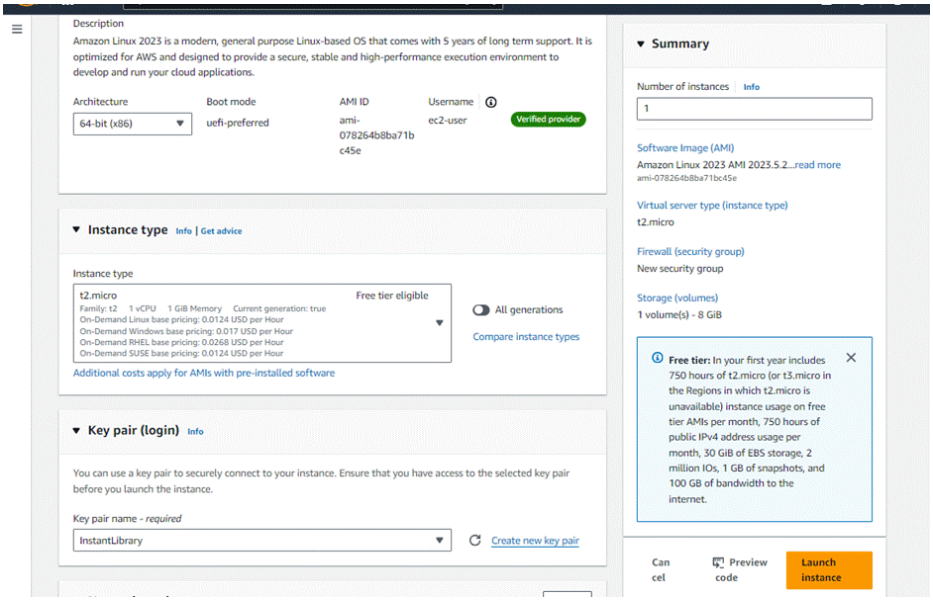


? Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

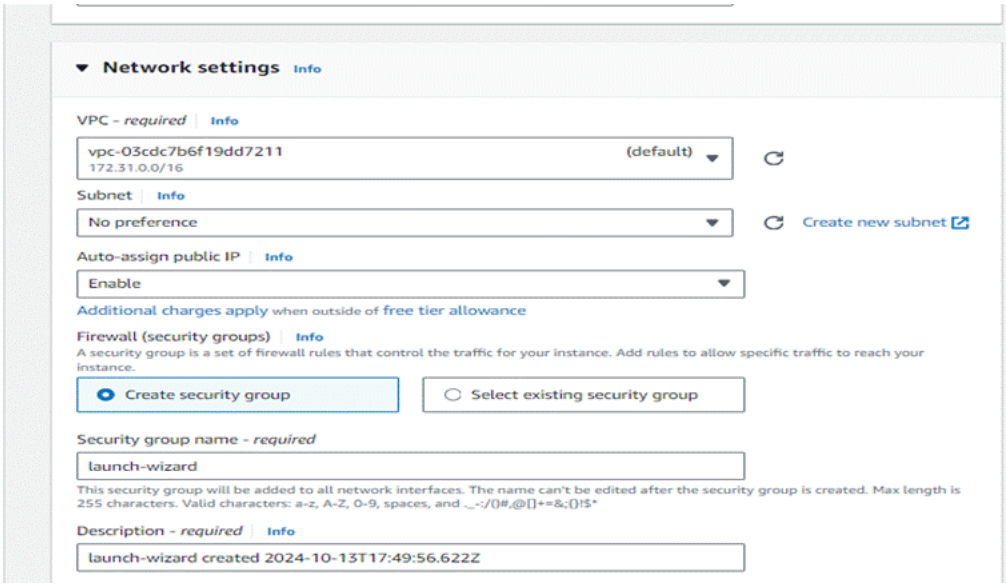


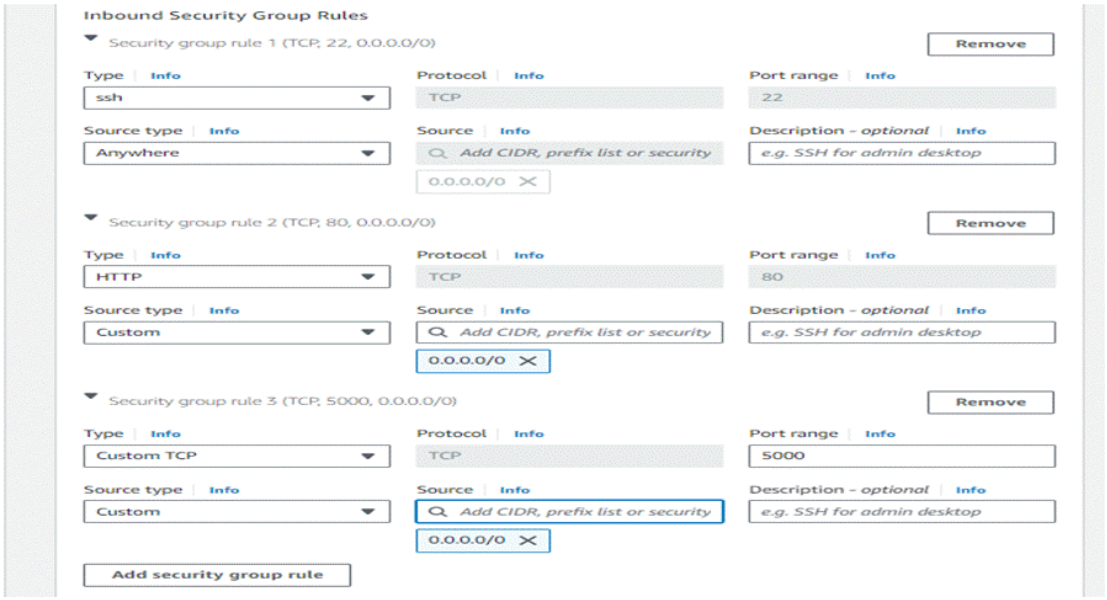
- **Create and download the key pair for Server access.**

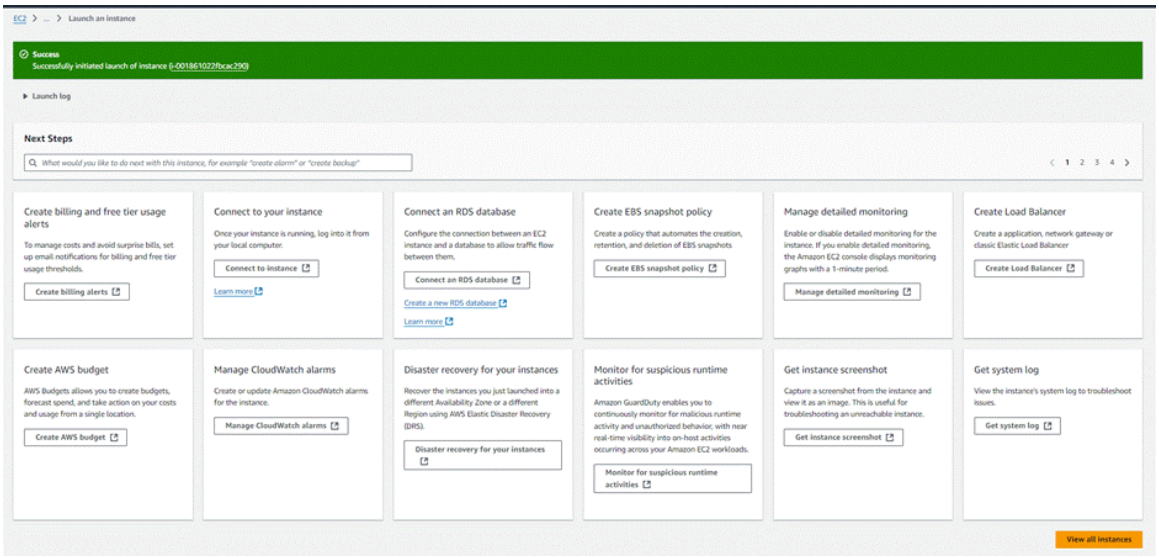




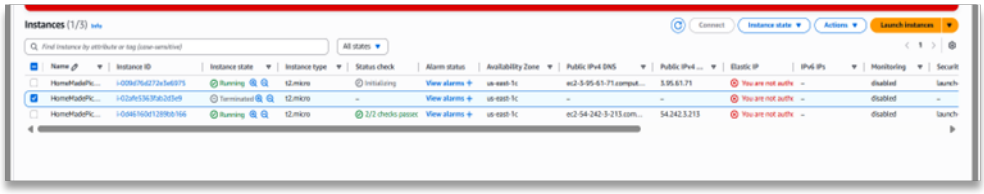
Configure security groups for HTTP, and SSH access.

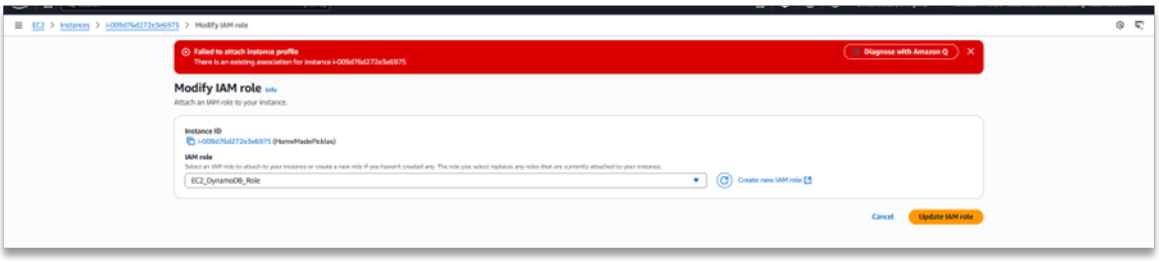


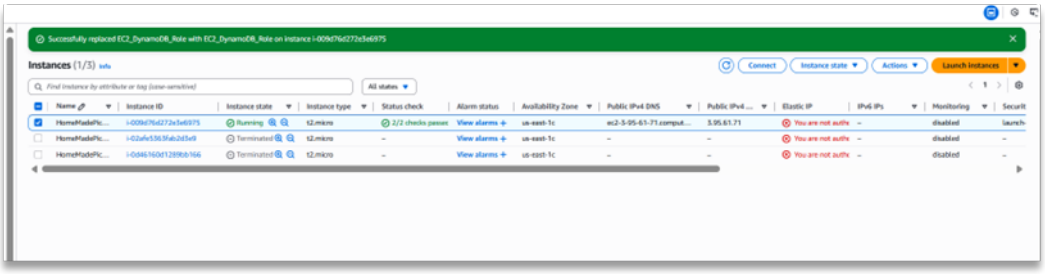




To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected,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







Now connect the EC2 with the files

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

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

- Run: 'https://github.com/yasminshaik1329/pickle-paradsie.git'
- Note: change your-github-username and your-repository-name with your credentials here:https://github.com/yasminshaik1329/pickle-paradsie.git
- This will download your project to the EC2 instance.

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

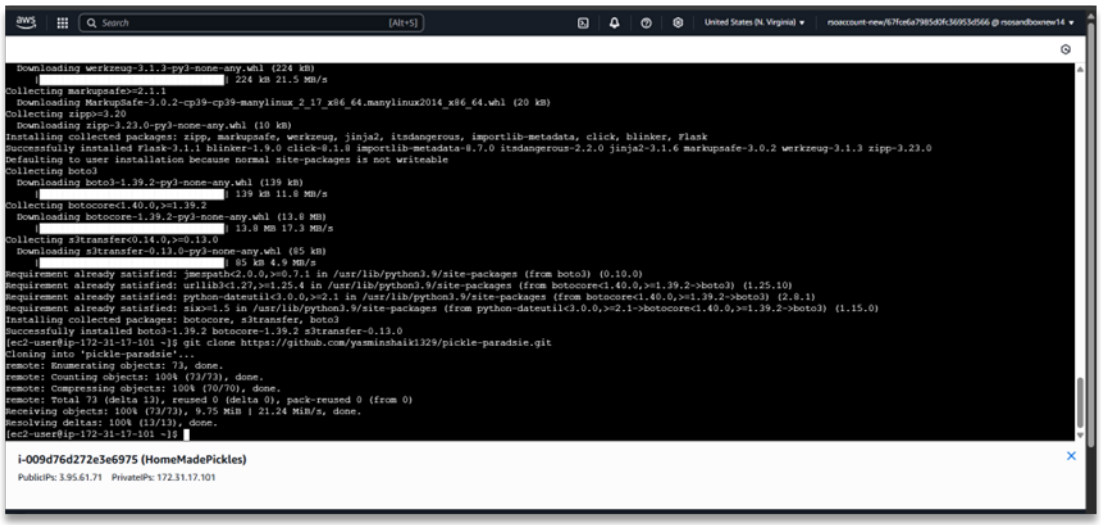
- cd Homemadepicklesandsnacks
- cd “Home Made Pickles1”

Create a Virtual Environment:

- python3 -m venv 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:

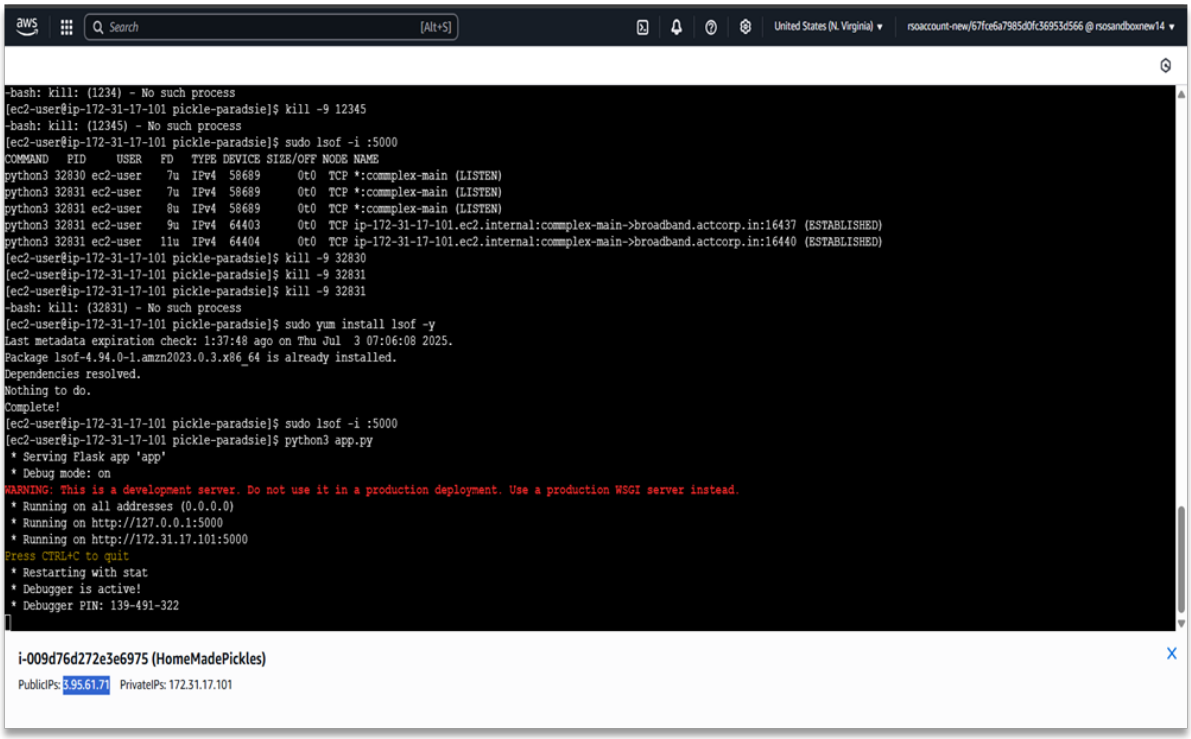
- Run the Flask Application
- sudo flask run --host=0.0.0.0 --port=5000



Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



Milestone 7 : Testing and Deployment

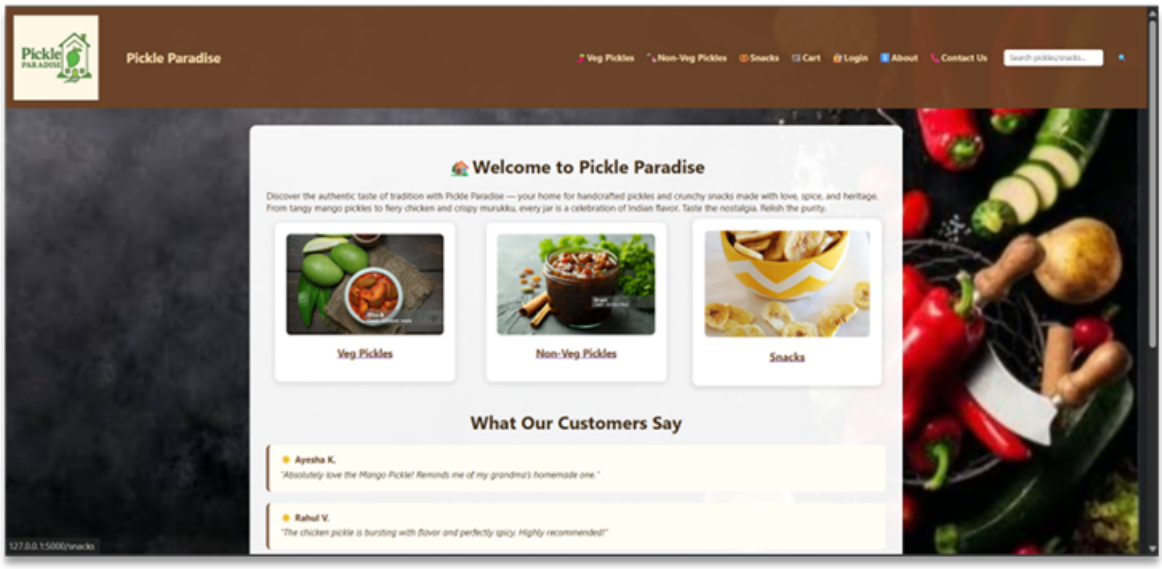
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

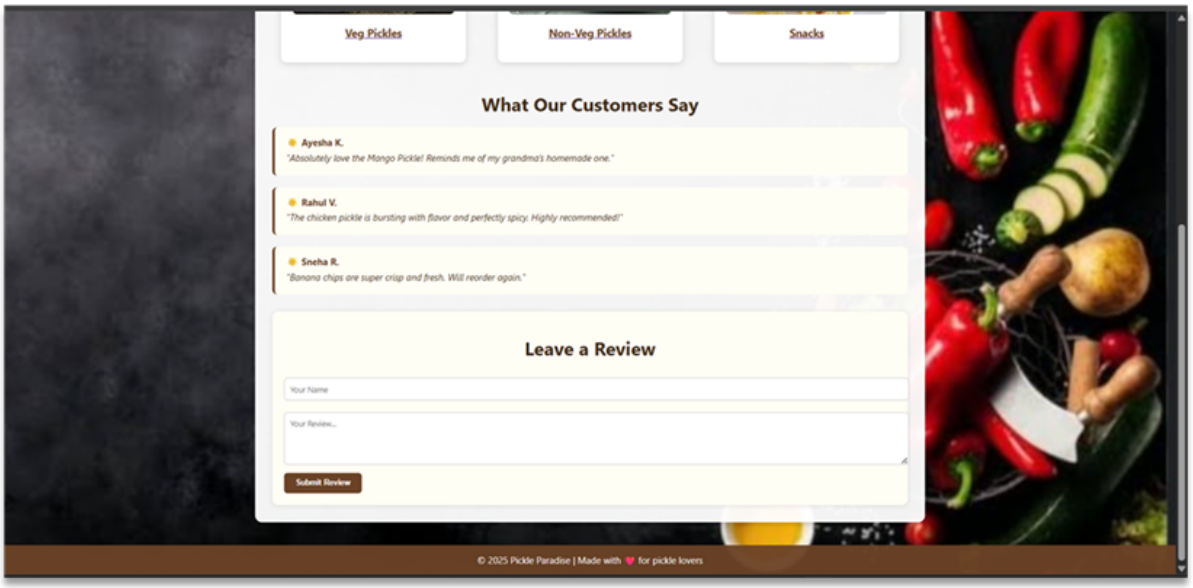
smoothly in the production environment.

Functional testing to verify the Project

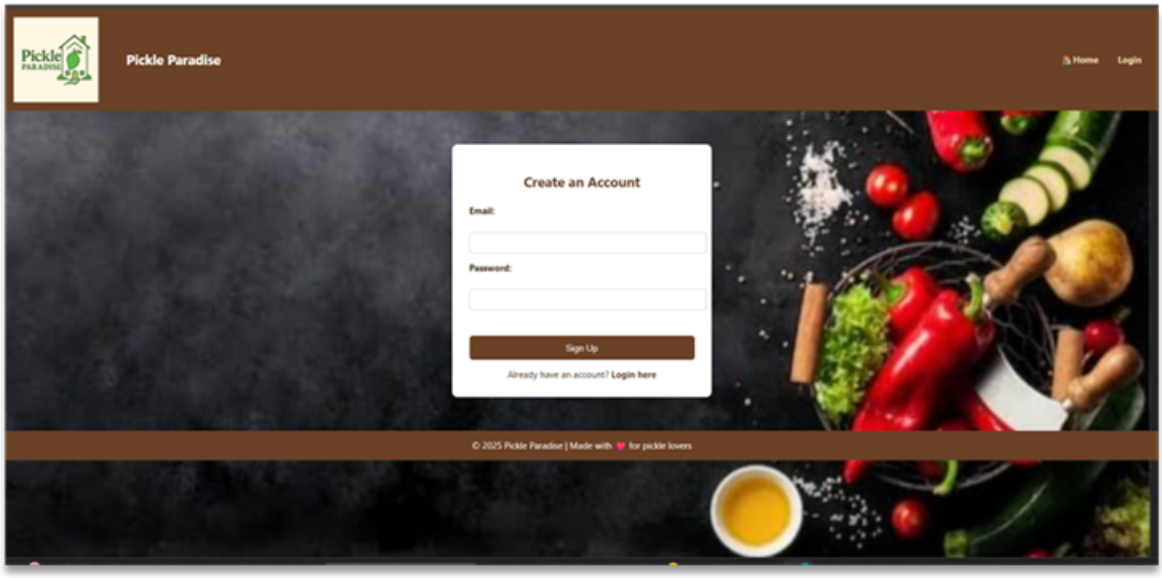
Welcome page:

Home page:

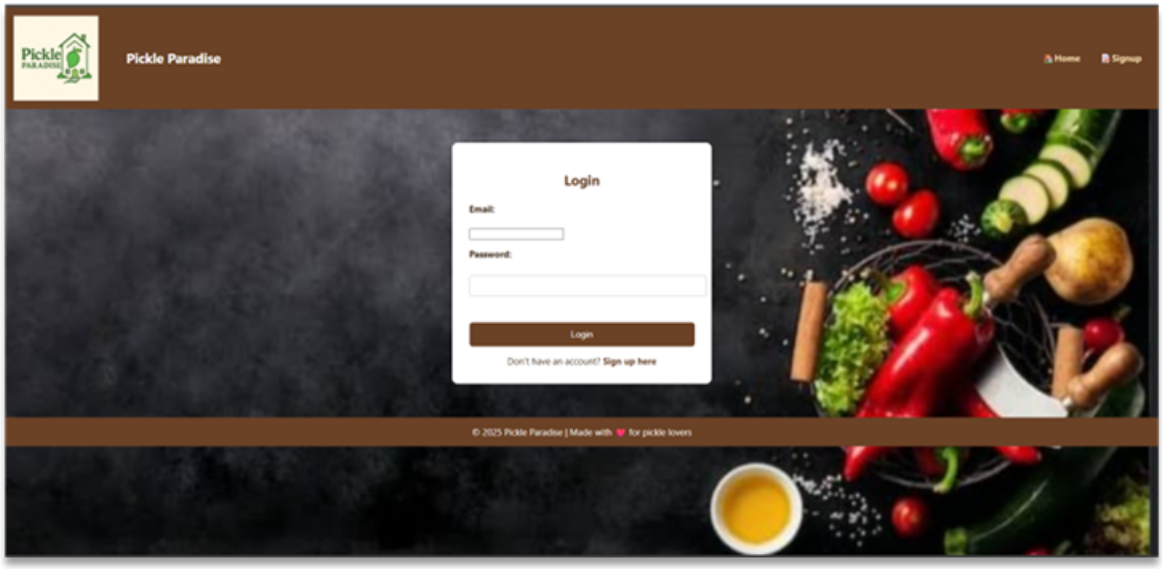




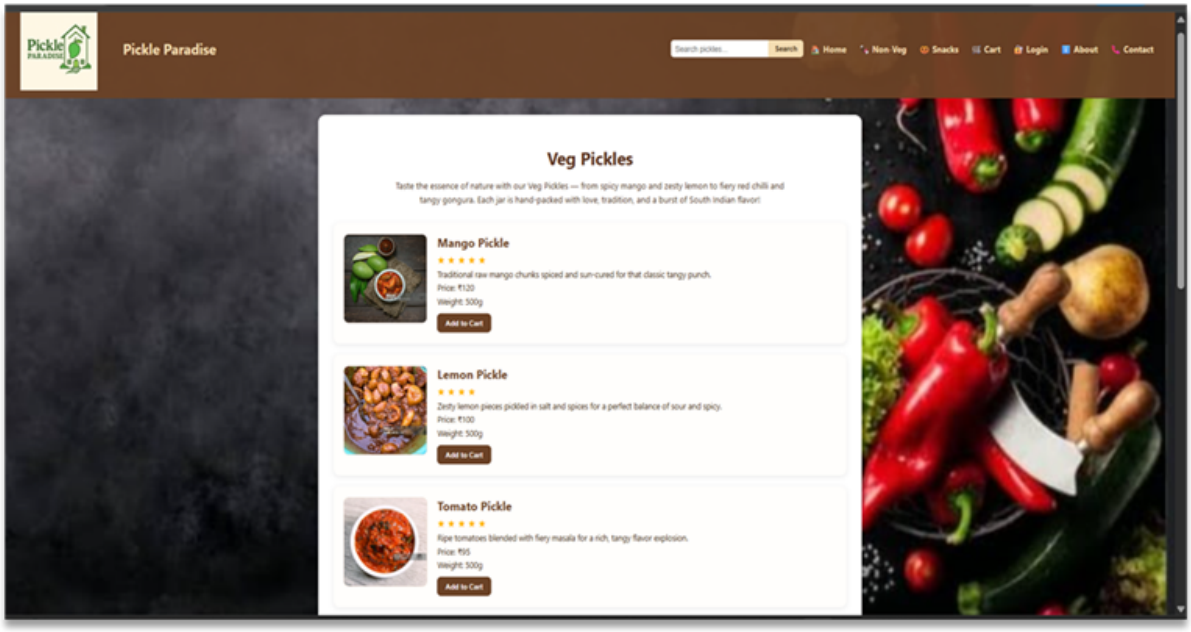
Signup page:

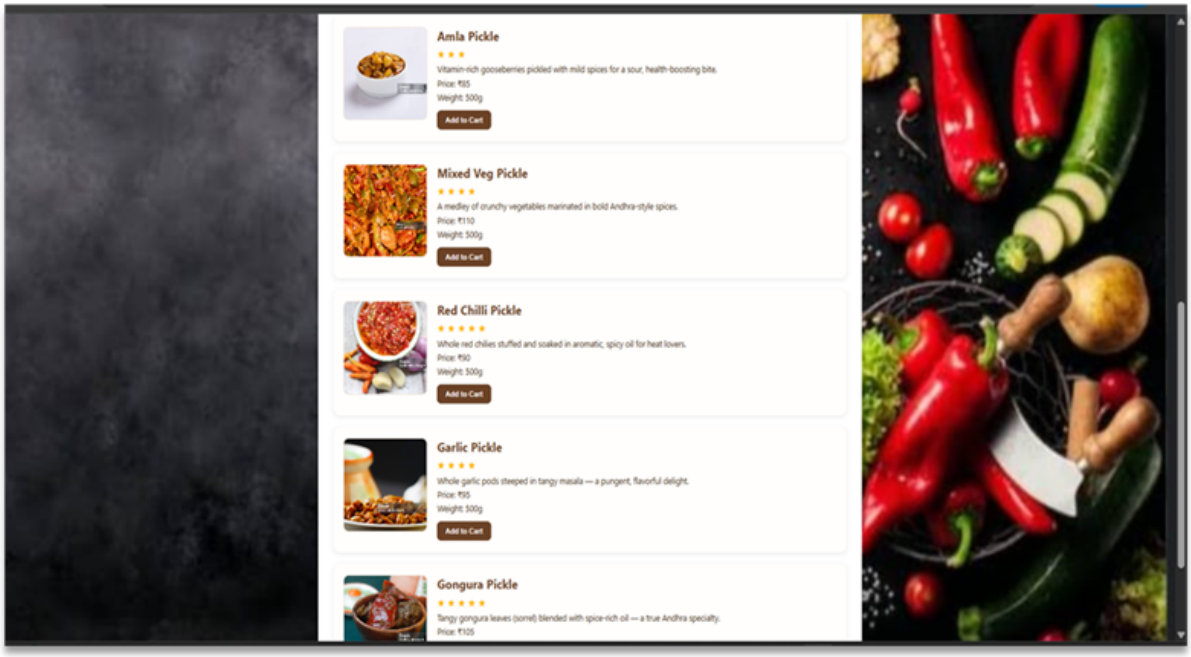


Login page:

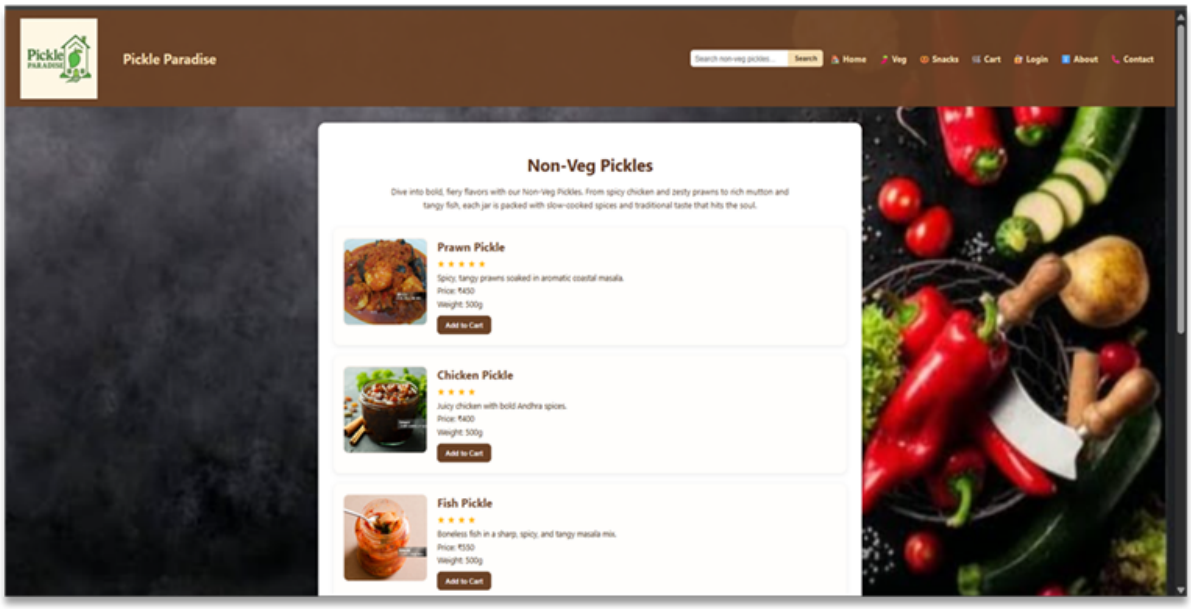


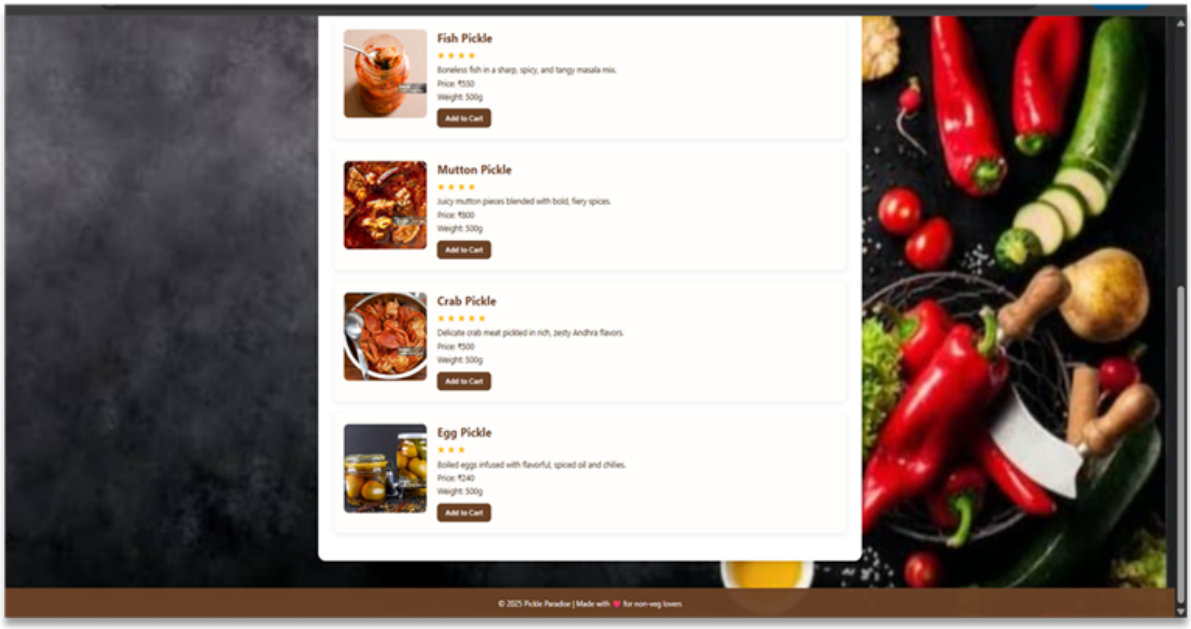
Veg pickles:


Pickle Paradise
Search
Home
Non-Veg
Snacks
Cart
Login
About
Contact
Veg Pickles
Taste the essence of nature with our Veg Pickles — from spicy mango and zesty lemon to fiery red chilli and tangy gongura. Each jar is hand-packed with love, tradition, and a burst of South Indian flavor!
Mango Pickle
Traditional raw mango chunks spiced and sun-cured for that classic tangy punch.
Price: ₹120
Weight: 500g
Add to Cart
Lemon Pickle
Zesty lemon pieces pickled in salt and spices for a perfect balance of sour and spicy.
Price: ₹100
Weight: 500g
Add to Cart
Tomato Pickle
Ripe tomatoes blended with fiery masala for a rich, tangy flavor explosion.
Price: ₹95
Weight: 500g
Add to Cart

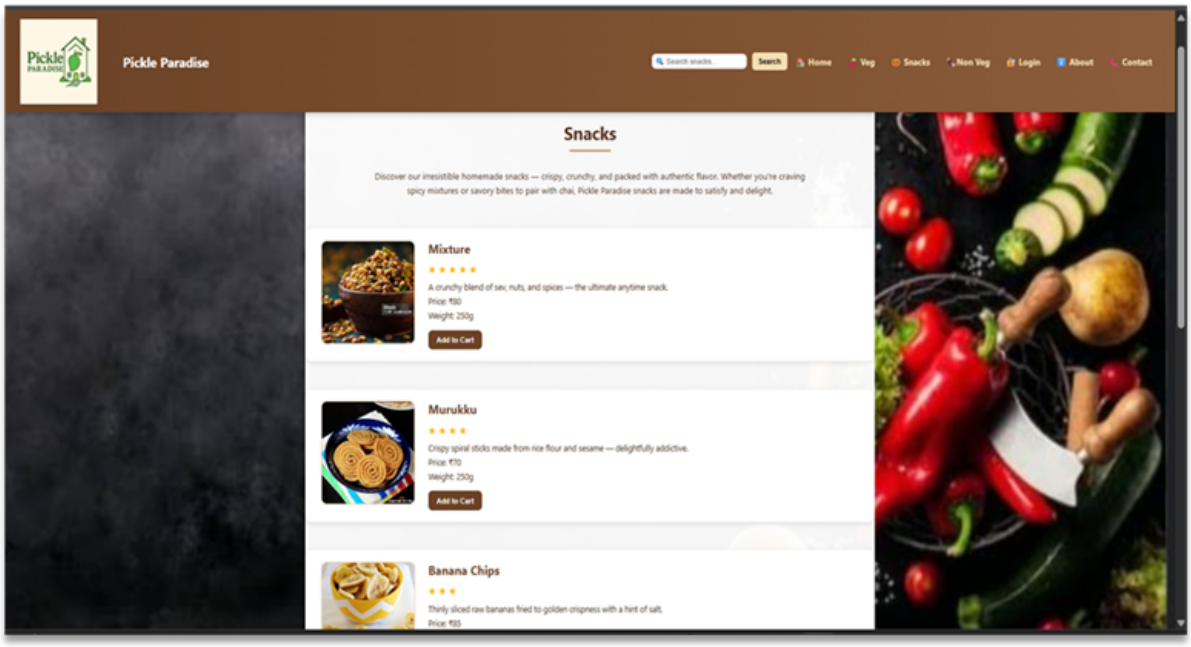

Amla Pickle
Vitamin-rich gooseberries pickled with mild spices for a sour, health-boosting bite.
Price: ₹85
Weight: 500g
Add to Cart
Mixed Veg Pickle
A medley of crunchy vegetables marinated in bold Andhra-style spices.
Price: ₹110
Weight: 500g
Add to Cart
Red Chilli Pickle
Whole red chilies stuffed and soaked in aromatic, spicy oil for heat lovers.
Price: ₹90
Weight: 500g
Add to Cart
Garlic Pickle
Whole garlic pods steeped in tangy masala — a pungent, flavorful delight.
Price: ₹95
Weight: 500g
Add to Cart
Gongura Pickle
Tangy gongura leaves (sorrel) blended with spice-rich oil — a true Andhra specialty.
Price: ₹105

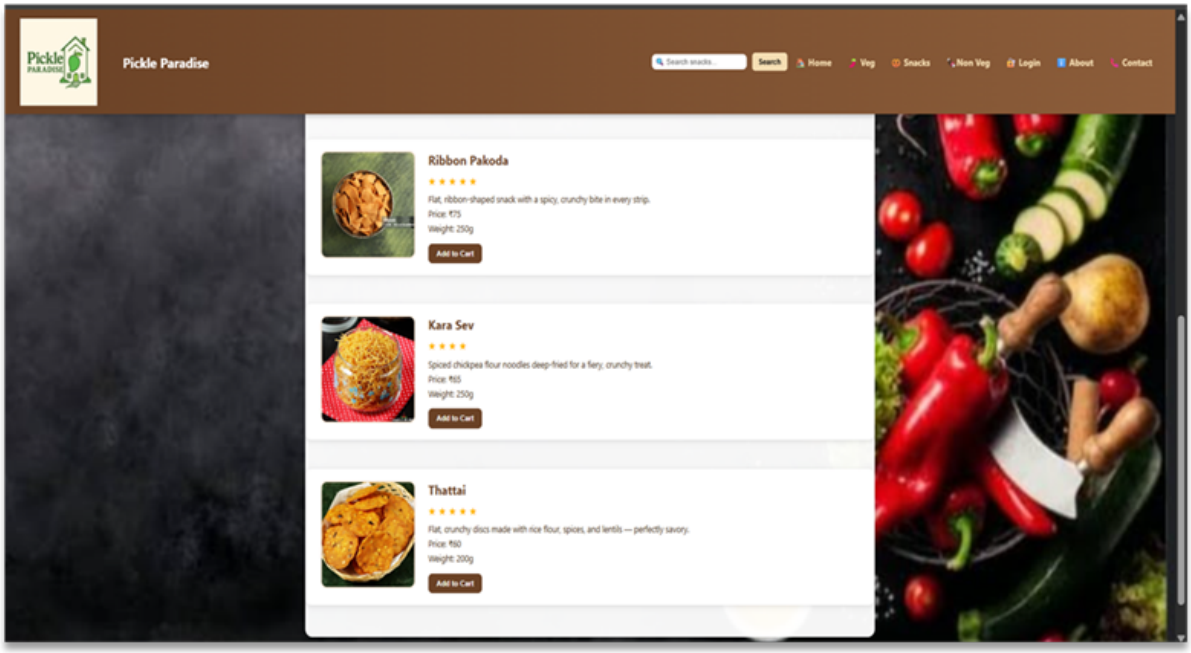
Non-Veg pickles:



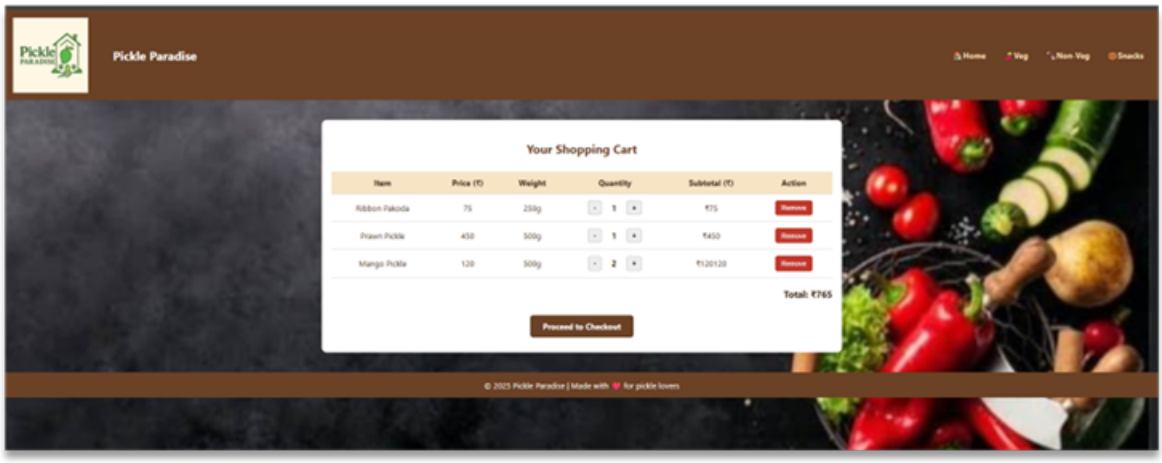


Snacks:

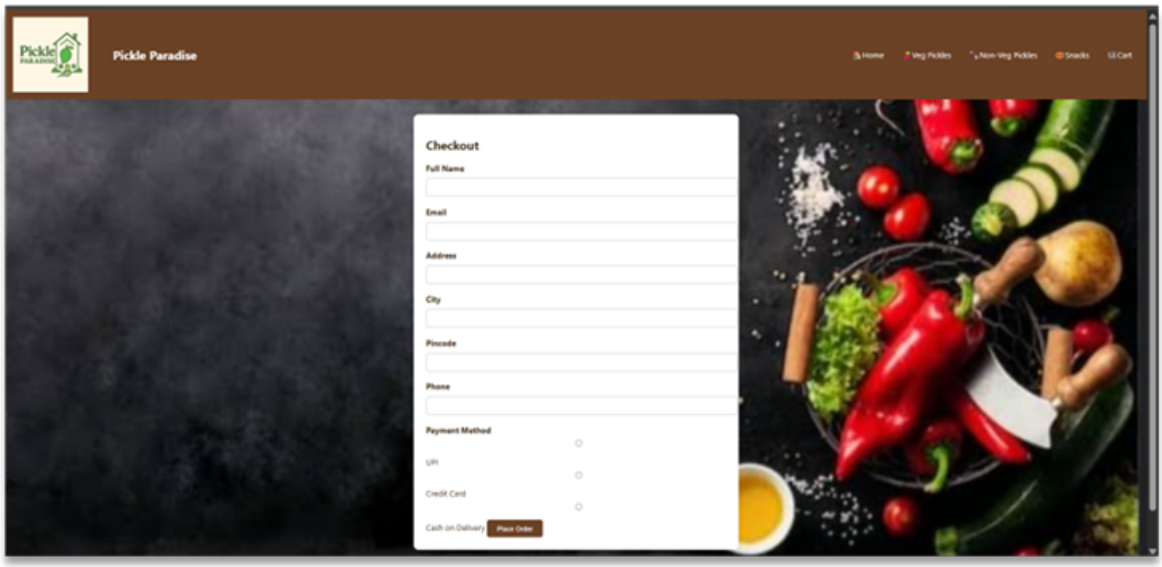

Pickle Paradise
Search
Home
Veg
Snacks
Non Veg
Login
About
Contact
Snacks
Discover our irresistible homemade snacks — crispy, crunchy, and packed with authentic flavor. Whether you're craving spicy mixtures or savory bites to pair with chai, Pickle Paradise snacks are made to satisfy and delight.
Mixture
A crunchy blend of sev, nuts, and spices — the ultimate anytime snack.
Price: ₹80
Weight: 250g
Add to Cart
Murukku
Crispy spiral sticks made from rice flour and sesame — delightfully addictive.
Price: ₹70
Weight: 250g
Add to Cart
Banana Chips
Thinly sliced raw bananas fried to golden crispness with a hint of salt.
Price: ₹65


Pickle Paradise
Search
Home
Veg
Snacks
Non Veg
Login
About
Contact
Ribbon Pakoda
Flat, ribbon-shaped snack with a spicy, crunchy bite in every strip.
Price: ₹75
Weight: 250g
Add to Cart
Kara Sev
Spiced chickpea flour noodles deep-fried for a fiery, crunchy treat.
Price: ₹65
Weight: 250g
Add to Cart
Thattai
Flat, crunchy discs made with rice flour, spices, and lentils — perfectly savory.
Price: ₹60
Weight: 200g
Add to Cart

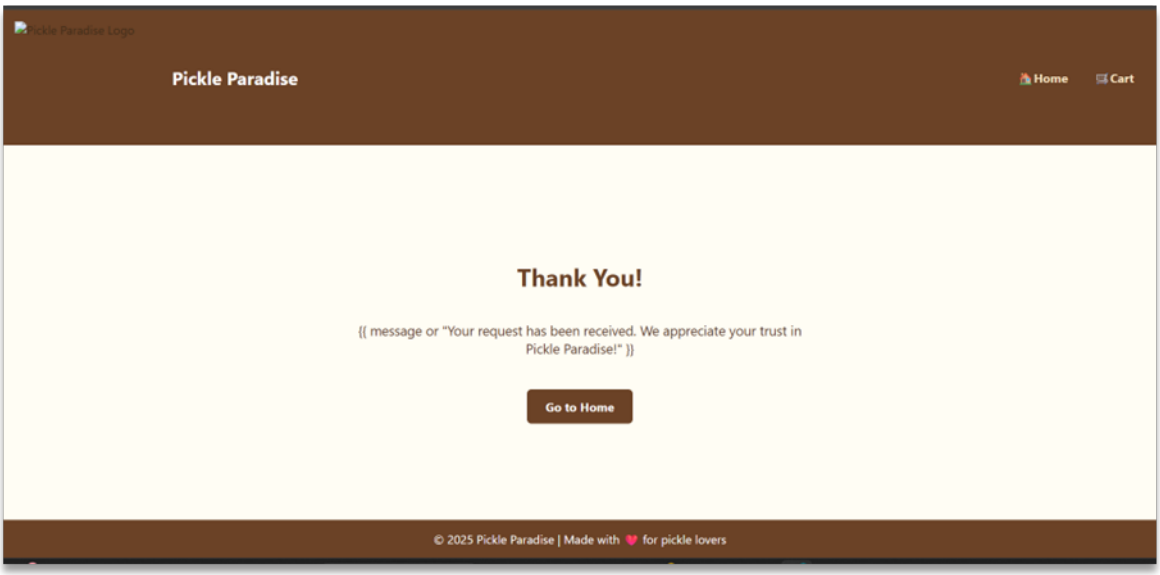
Cart Page:



Checkout Page:



Success Page:



Conclusion

Pickle Paradise demonstrates how cloud-native architecture can empower small businesses with reliable, scalable, and fast digital platforms. Leveraging AWS and Python's Flask framework, the platform ensures excellent user experience, automated notifications, and simple backend management.

The integration of cloud-ready architecture (e.g., AWS for future scalability) and robust session management allows the platform to handle high traffic efficiently while maintaining real-time updates for orders and inventory. Features like weight-based pricing, category-specific searches, and instant checkout streamline the shopping process, empowering customers to explore a diverse range of traditional and innovative recipes with ease.

This project addresses the growing demand for homemade, preservative-free food products by bridging the gap between small-scale producers and discerning customers. The platform's intuitive design and secure payment workflows enhance trust and convenience, while backend tools enable effortless inventory tracking and order fulfillment for administrators.

In essence, this project redefines the way homemade delicacies are shared and enjoyed, offering a flavorful bridge between tradition and technology.