Home Made Pickles & Snacks: Taste the Best

Project 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."

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

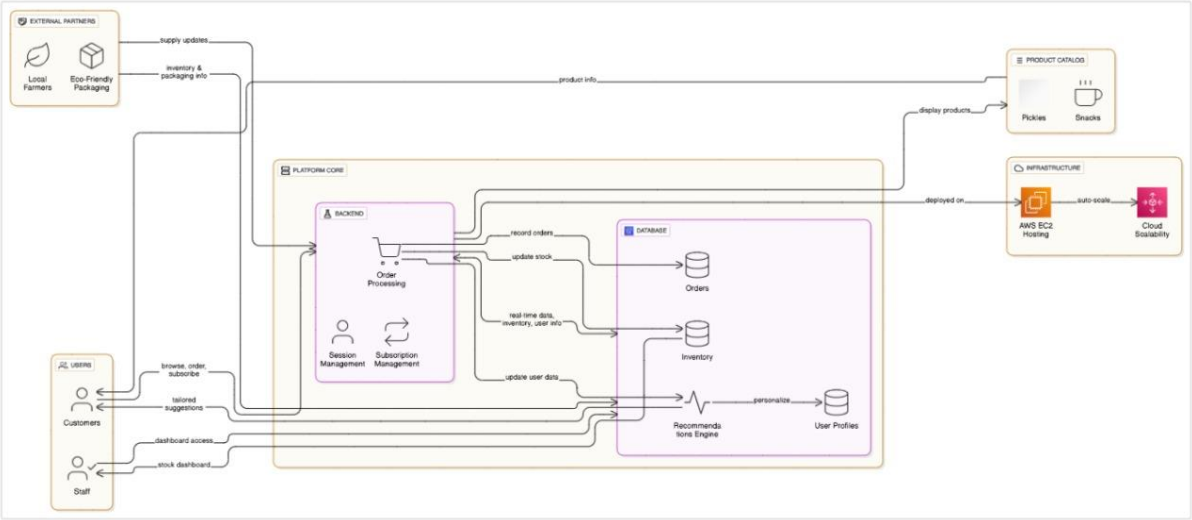
When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated

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

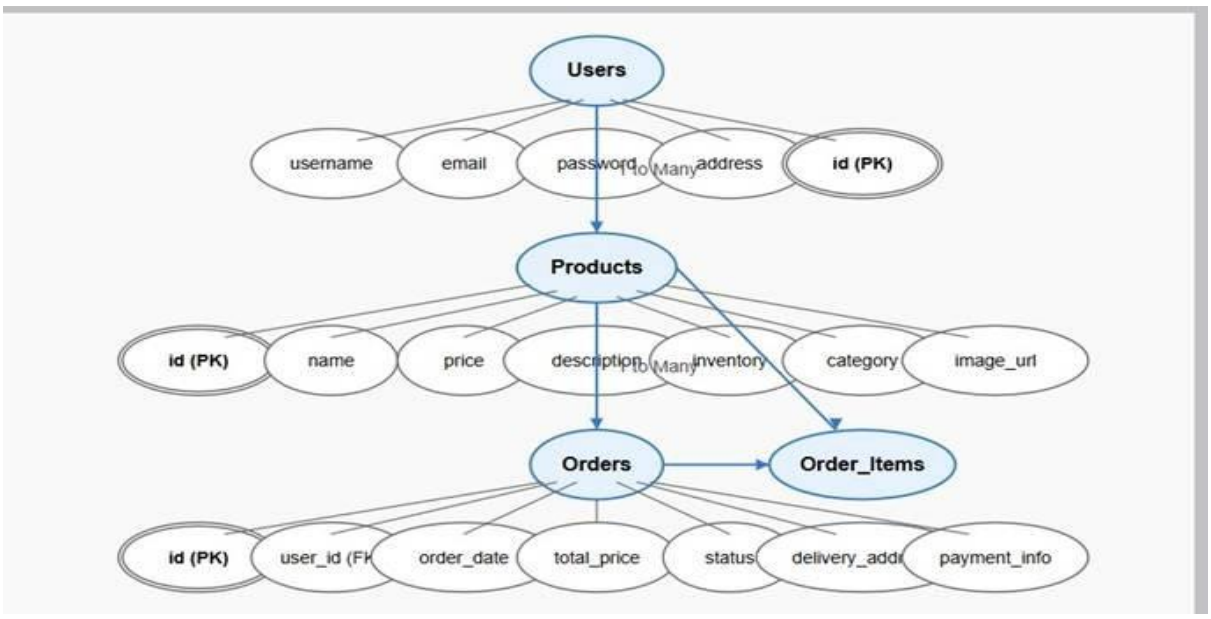
Scenario 3: Personalized User Experience and Recommendations

The platform leverages user behaviour 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.



Entity Relationship (ER)Diagram



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 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.

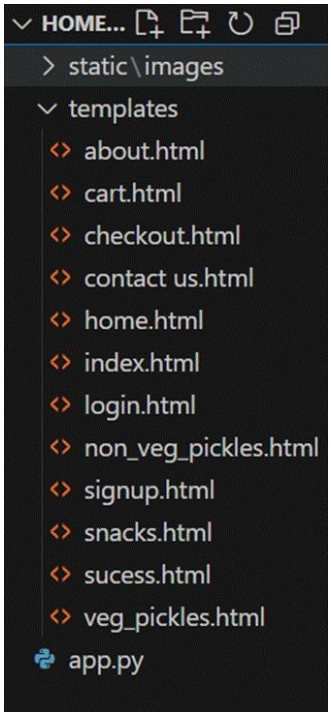
1. Web Application Development and Setup

Milestone 1: Web Application Development and Setup

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

  - Begin by building essential HTML pages and Flask routes using local Python dictionaries or lists for data storage. This allows testing and validation of core functionality before integrating cloud services.

- **Activity 1.2: Core Functionalities and User Interaction.**
  - Implement core features like user registration, login, and data submission using local storage. Ensure smooth navigation between pages and basic input validation on both frontend and backend.



Description: set up the Home-Made Pickles project with an app.py file, a static/folder for assets, and a templates/ directory containing all required HTML pages like home, login, register. products page etc.

Description of code:

- ## Flask app installation:

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

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__)
app.secret_key = os.urandom(24)
```

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

- ## Dynamodb Setup:

```
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

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

- **SNS connection:**

```
SMTP_SERVER = os.environ.get('SMTP_SERVER', 'smtp.gmail.com')
SMTP_PORT = int(os.environ.get('SMTP_PORT', 587))

SENDER_EMAIL = os.environ.get('SENDER_EMAIL')
SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD')

ENABLE_EMAIL = os.environ.get('ENABLE_EMAIL', 'False').lower() == 'true'

#SNS Configuration

SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
ENABLE_SNS = os.environ.get('ENABLE_SNS', 'False').lower() == 'true'

# Initialize SNS client
sns = boto3.client('sns', region_name='us-east-1')

logging.basicConfig(
 level=logging.INFO,
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
 handlers=[
 logging.FileHandler("fleetsync.log"),
 logging.StreamHandler()
 ]
)
logger = logging.getLogger(__name__)
```

Description: Configure SNS to send notifications when a book 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.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section.

- **Products:**

```
# Product Data - Unchanged
veg_items = [
    {"name": "Mango Pickles", "price": 150, "img": "Mango-Pickles.jpg"},
    {"name": "Lemon Pickles", "price": 120, "img": "Lemon Pickles.jpg"},
    {"name": "Tomato Pickles", "price": 180, "img": "Tomato Pickles.jpg"},
    {"name": "Spicy Pandu Mirchi Pickles", "price": 160, "img": "Spicy Pandu Mirchi Pickles.jpg"},
    {"name": "Kakarakaya Pickles", "price": 170, "img": "Kakarakaya Pickles.jpg"}
]

non_veg_items = [
    {"name": "Gongura Chicken Pickles", "price": 300, "img": "Gongura Chicken Pickles.jpg"},
    {"name": "Mutton Pickles", "price": 350, "img": "Mutton Pickles.jpg"},
    {"name": "Fish Pickles", "price": 320, "img": "Fish Pickles.jpg"},
    {"name": "Gongura Prawns Pickles", "price": 340, "img": "Gongura Prawns Pickles.jpg"},
    {"name": "Gongura Mutton Pickles", "price": 360, "img": "Gongura Mutton Pickles.jpg"}
]

snack_items = [
    {"name": "Crispy Aam Papad", "price": 200, "img": "Crispy Aam Papad.jpg"},
    {"name": "Crispy Chekka Pakodi", "price": 100, "img": "Crispy Chekka Pakodi.jpg"},
    {"name": "Dryfruitladdu", "price": 120, "img": "Dryfruitladdu.jpg"},
    {"name": "Chekkalu", "price": 130, "img": "Chekkalu.jpg"},
    {"name": "Banana Chips", "price": 140, "img": "Banana chips.jpg"},
    {"name": "Boondhi Acchu", "price": 110, "img": "Boondhi acchu.jpg"},
    {"name": "Gavvalu", "price": 115, "img": "Gavvalu.jpg"},
    {"name": "Kaju Chikki", "price": 160, "img": "Kaju Chikki.jpg"},
    {"name": "Ragi Laddu", "price": 125, "img": "Ragi Laddu.jpg"}
]

# Routes
```

- **Routes of Web Pages:**

Home Route:

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

- **Login Route:**

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = user_table.get_item(Key={'email': email}).get('Item')
        if user and user['password'] == password:
            session['user'] = email
            flash("Login successful!", "success")
            return redirect(url_for('index'))
        flash("Invalid credentials", "danger")
    return render_template('login.html')
```

- **index Route:**

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

- **Contact Route:**

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

- **Signup Route:**

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user_table.put_item(Item={'email': email, 'password': password})
        send_email(email, "Welcome to Pickle Paradise", "Thank you for signing up!")
        flash("Signup successful! Please login.", "success")
        return redirect(url_for('login'))
    return render_template('signup.html')
```

- **Logout Route:**

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

- **Non-Veg Pickles Route:**

```
@app.route('/non_vegpickles')
def non_vegpickles():
    return render_template('non_vegpickles.html', products=products ['non_vegpickles'])
```

- **Veg Pickles Route:**

```
@app.route('/veg_pickles')
def veg_pickles():
    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products ['veg_pickles'
```

- **Snacks Route:**

```python
@app.route('/snacks')
def snacks():
    return render_template('snacks.html', products=products['snacks'])
```

- **Checkout Route:**

```python
@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if request.method == 'POST':
        name = request.form['fullname']
        email = request.form['email']
        address = request.form['address']
        phone = request.form['phone']
        payment = request.form['payment']
        cart_items = session.get('cart', [])
        total = sum(int(i['price']) for i in cart_items)
        order_id = str(uuid.uuid4())

        orders_table.put_item(Item={
            'order_id': order_id,
            'name': name,
            'email': email,
            'address': address,
            'phone': phone,
            'payment': payment,
            'total': total,
            'items': cart_items
        })

        send_email(email, "Order Confirmation", f"Thank you {name} for your order! Total: ₹{total}")

        session.pop('cart', None)
        return render_template('success.html', name=name, order_id=order_id)
```

- **Cart Route:**

```
@app.route('/cart')
def cart():
    cart_items = session.get('cart', [])
    total = sum(int(i['price']) for i in cart_items)
    return render_template('cart.html', cart=cart_items, total=total)
```

- **Success Route:**

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

- **About Route:**

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

- **Deployment:**

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

Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 5000 with debug mode enabled for development and testing.

- **Env file:**

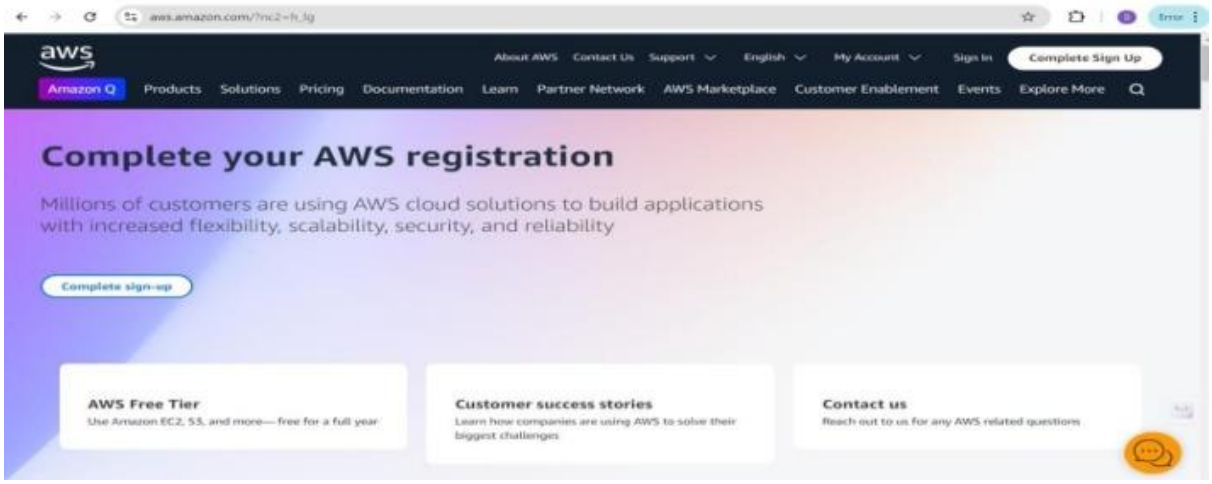
```
msg = MIMEMultipart()
msg['From'] = EMAIL_ADDRESS
msg['To'] = to_email
msg['Subject'] = subject
msg.attach(MIMEText(body, 'plain'))

server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
server.send_message(msg)
server.quit()
```

2. AWS Account Setup:

Milestone 2: AWS Account Setup

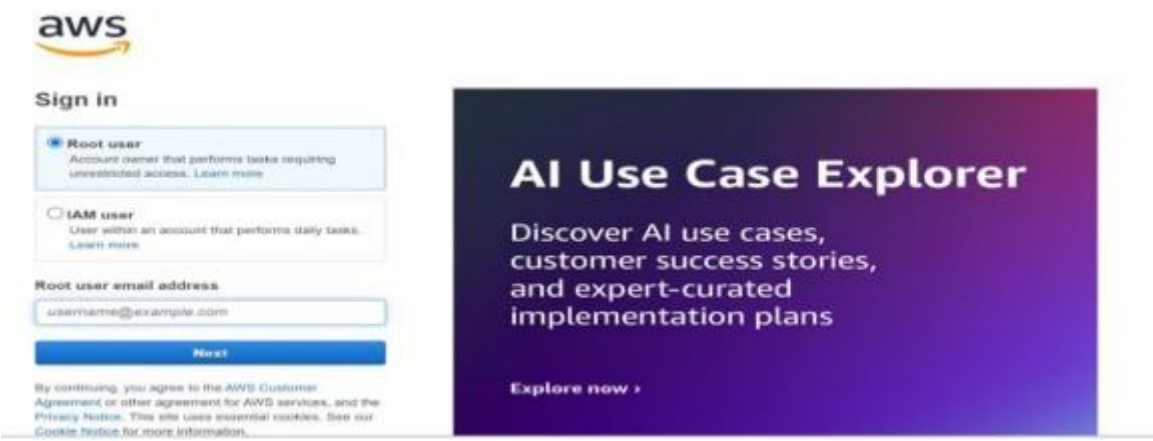
- **Activity 2.1: Set up an AWS account if not already done.**

  Begin by building essential HTML pages and Flask routes using local Python dictionaries or lists for data storage. This allows testing and validation of core functionality before integrating cloud services.



- **Activity 2.2: Log in to the AWS Management Console.**

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

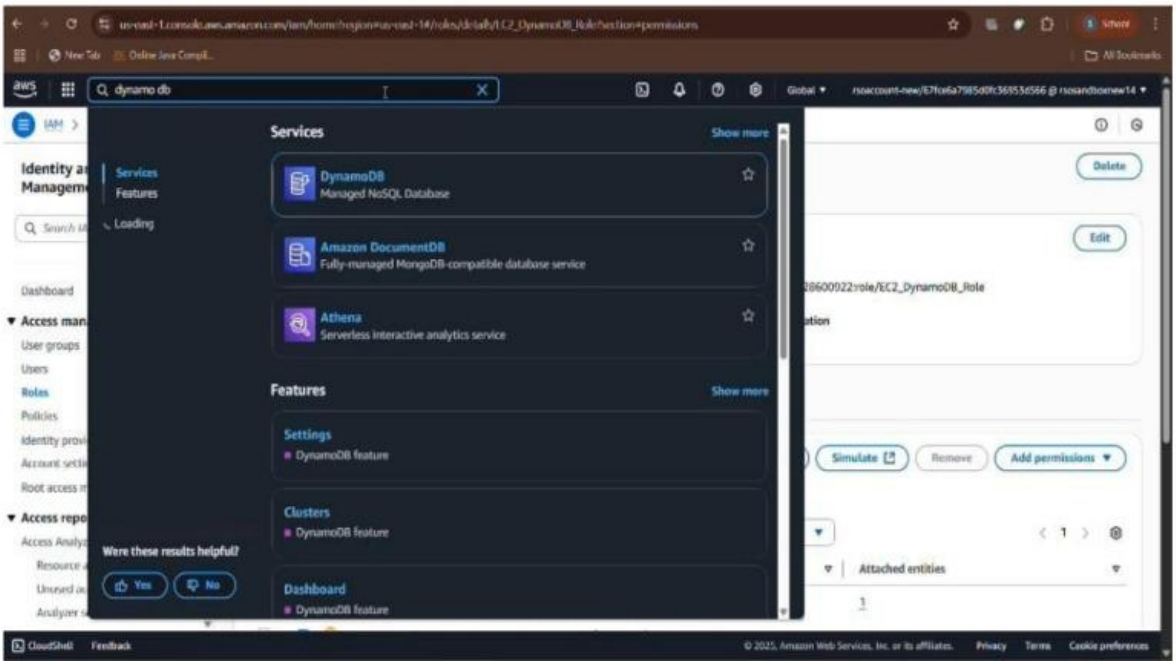


3. DynamoDB Database Creation and Setup

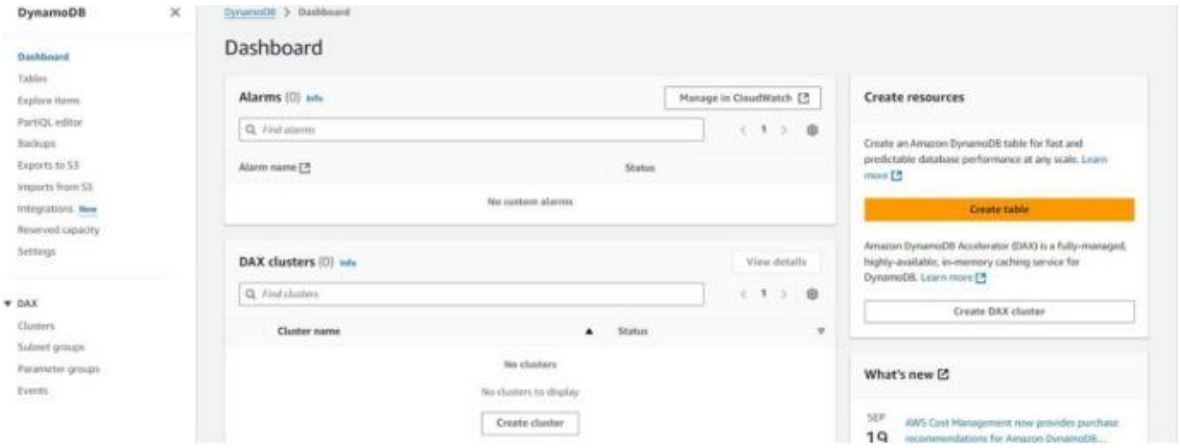
Milestone 3: DynamoDB Database Creation and Setup

○ Activity 3.1: Navigate to the DynamoDB

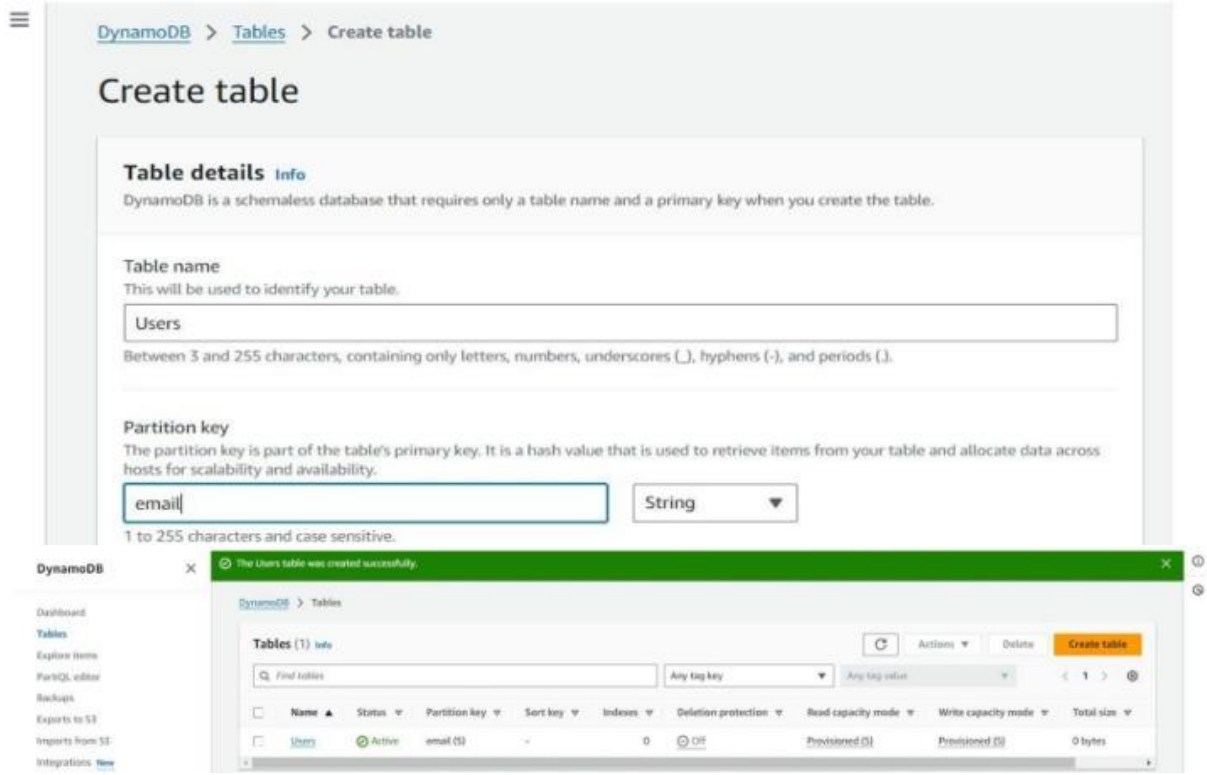
In the AWS Console, navigate to DynamoDB and click on create tables



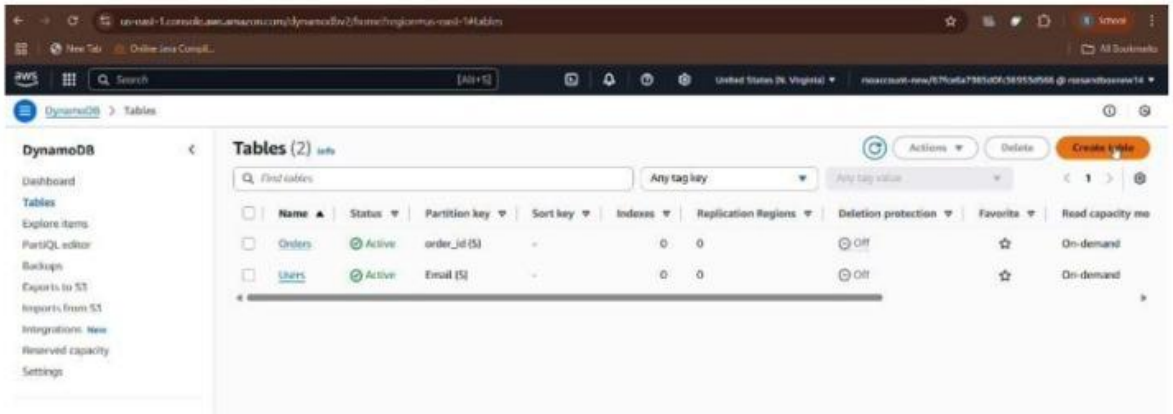
- **Activity 3.2: Create a DynamoDB table for storing registration details and book requests.**



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



- Follow the same steps to create a requests table with E-mail as the primary key for book requests data.

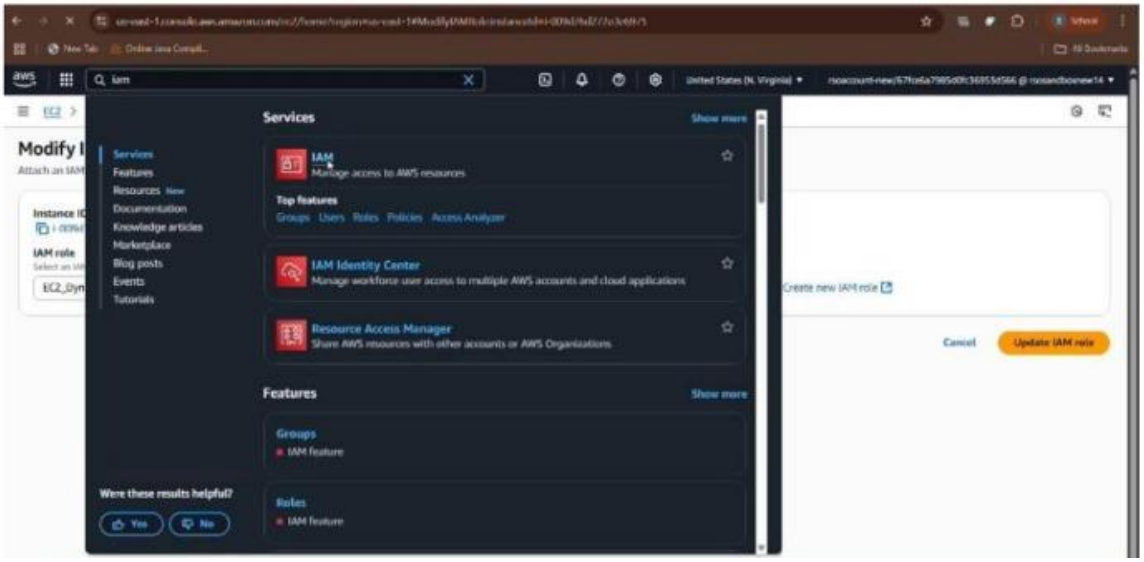


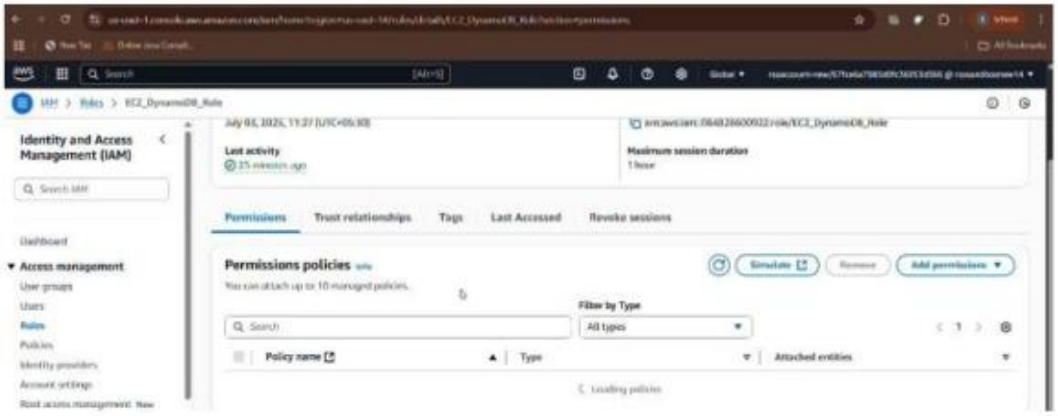
4: IAM Role Setup

Milestone 4: IAM Role Setup

- **Activity 4.1: Create IAM Role**

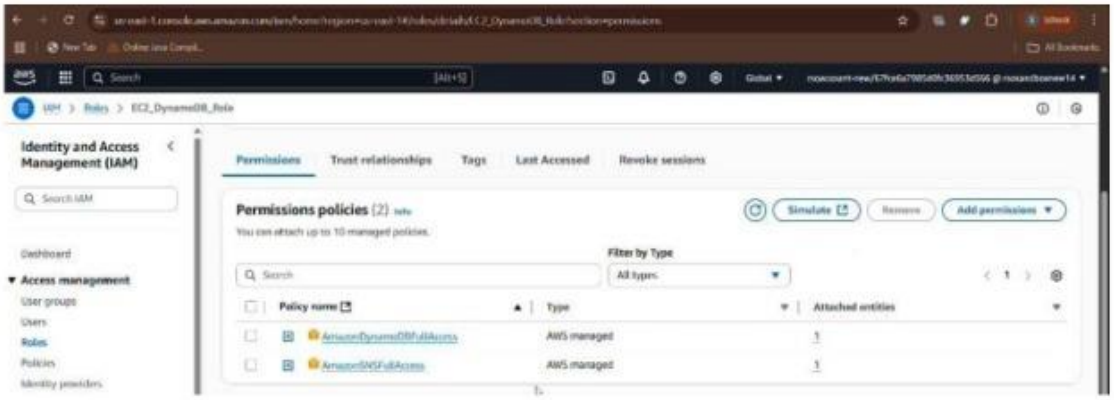
In the AWS Console, navigate to IAM and create a new IAM Role for EC2 to allow interaction with DynamoDB.





- **Activity 4.2: Attach Policies**

Attach the Amazon DynamoDBFullAccess and AmazonSNSFullAccess policy to the role. This grants EC2 instances permission to perform read and write operations on DynamoDB.

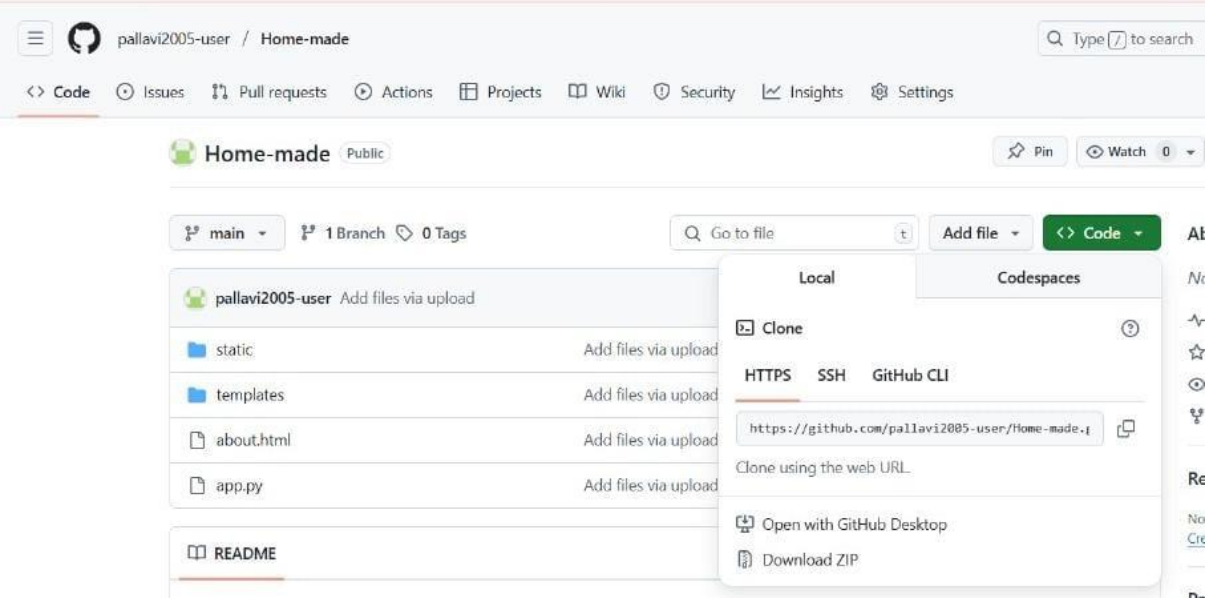


5: EC2 Instance Setup

Milestone 5: EC2 Instance Setup

- **Activity 5.1: Load Project Files to GitHub**

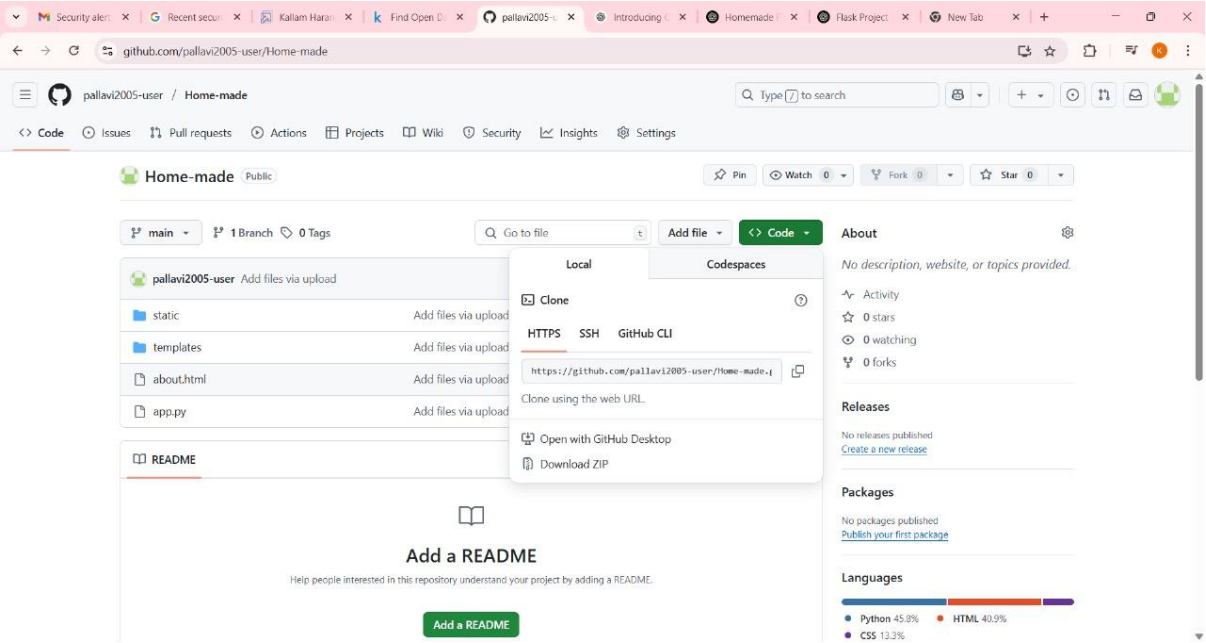
Upload your Flask application and HTML files to a GitHub reposit Note: This will allow easy access and deployment to the EC2 instance.

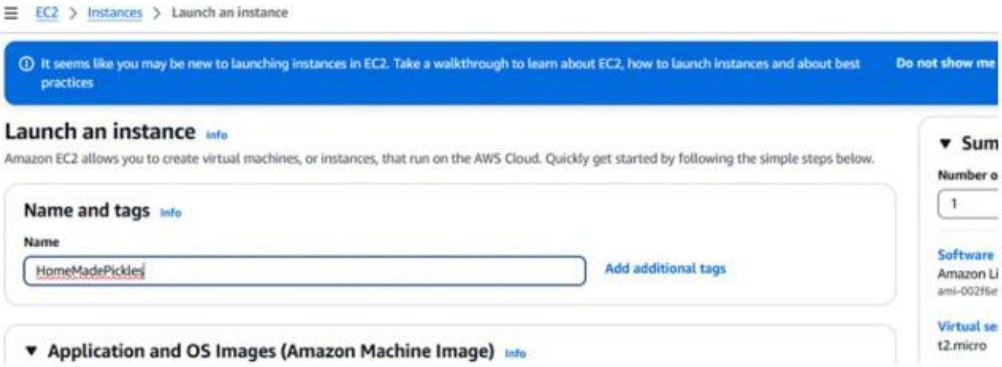


- **Activity 5.2: Launch an EC2 Instance**

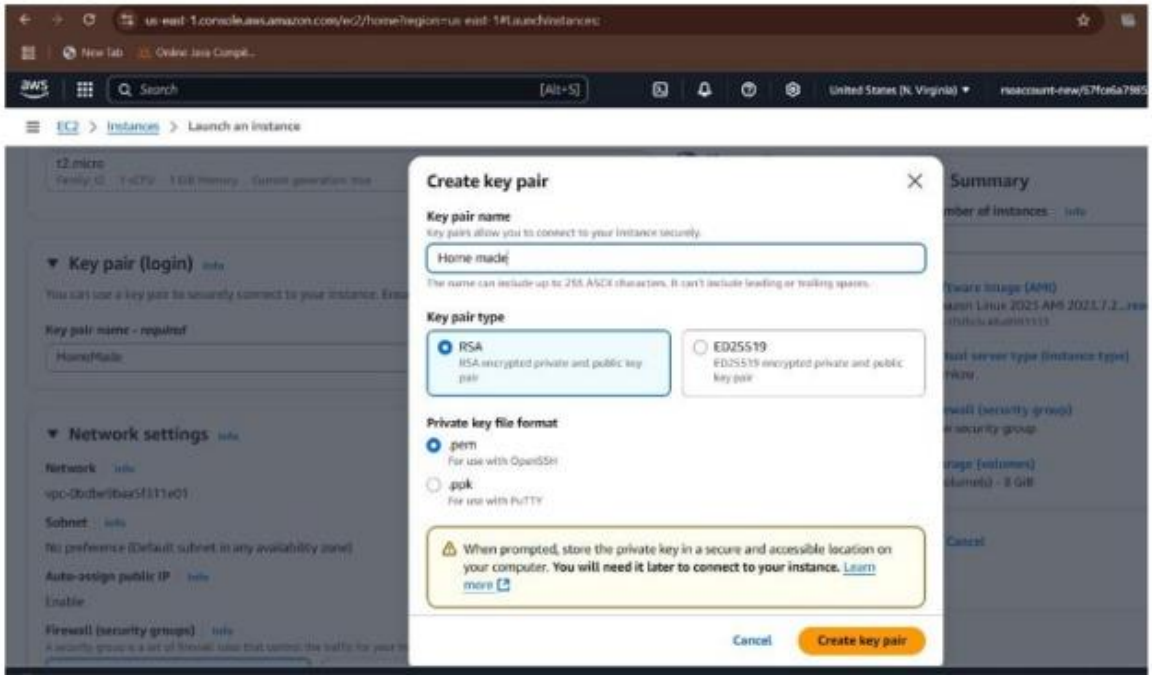
In the AWS Console, go to EC2 and click "Launch Instance".

Choose Amazon Linux 2 or Ubuntu as the AMI and select t2. micro [Free-tier eligible)



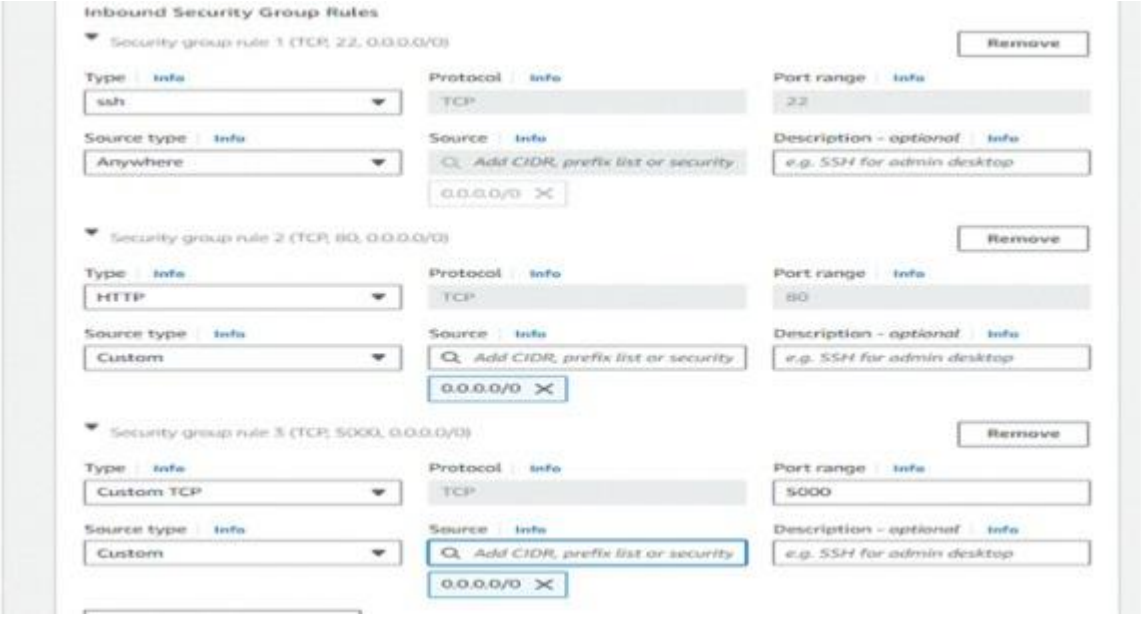


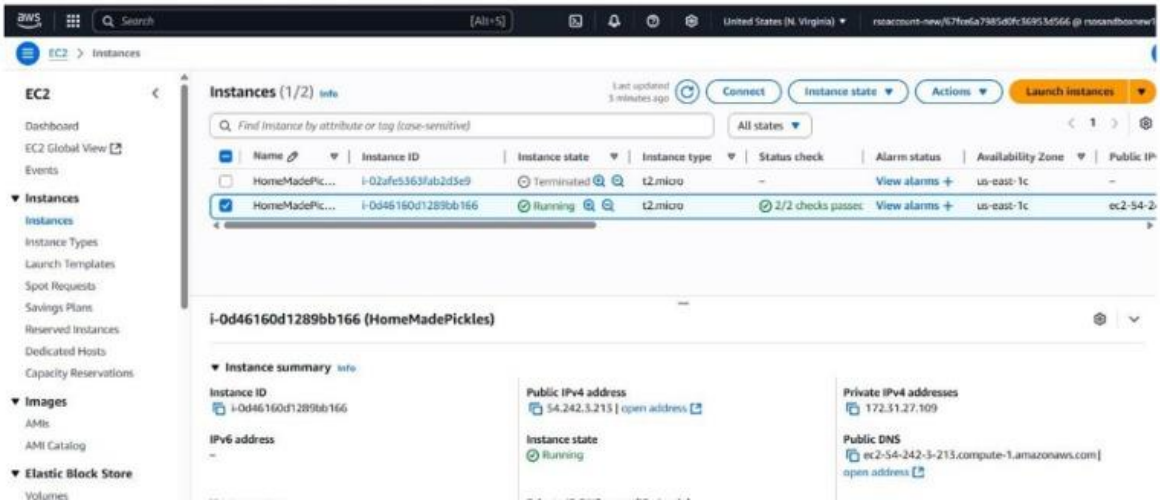
create and download a key pair for secure SSH access.



- **Activity 5.3: Configure Security Groups**

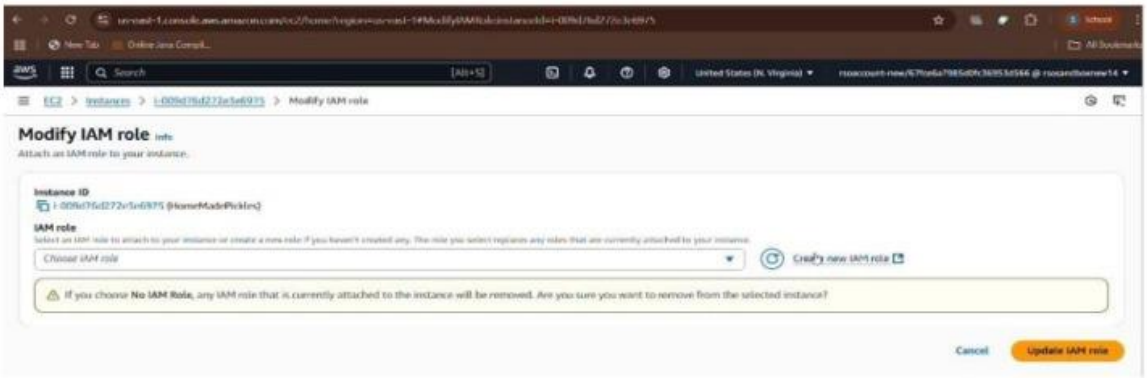
Allow HTTP (port 80) and SSH (port 22) inbound traffic.





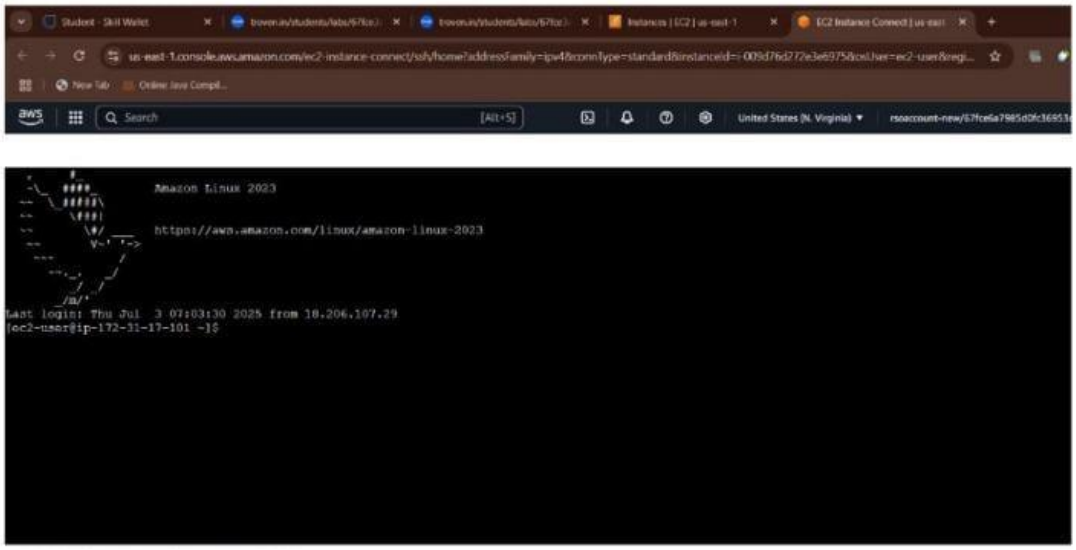
- **Activity 5.4: Attach IAM Role**

Attach the IAM Role created earlier to your EC2 instance by selecting your instance → Actions → Security → Modify IAM Role



- **Activity 5.5: Connect to EC2 Instance**

Use EC2 Instance Connect via AWS Console to open a terminal session.



6. Deployment on EC2

Milestone 6: Deployment on EC2

- **Activity 6.1: Install Required Software**

Run the following commands to install necessary packages:

sudo yum update-y

sudo yum install python3 git

sudo pip3 inst flask boto3

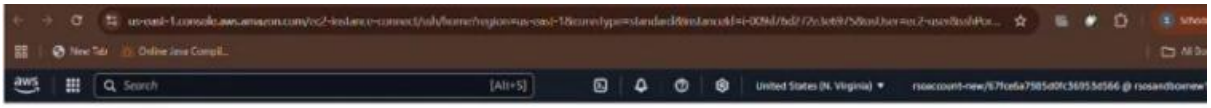
Verify installations:

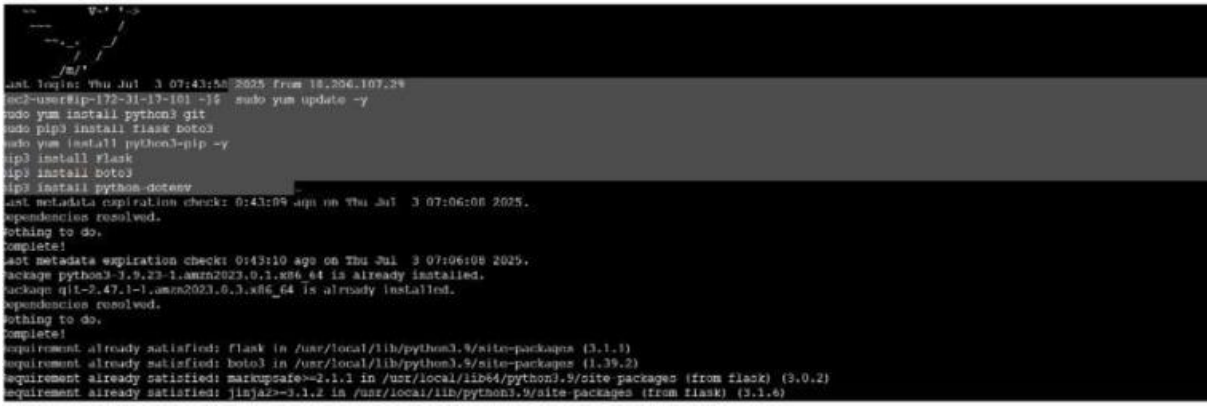
bash

Copy code

flask-version

git-version





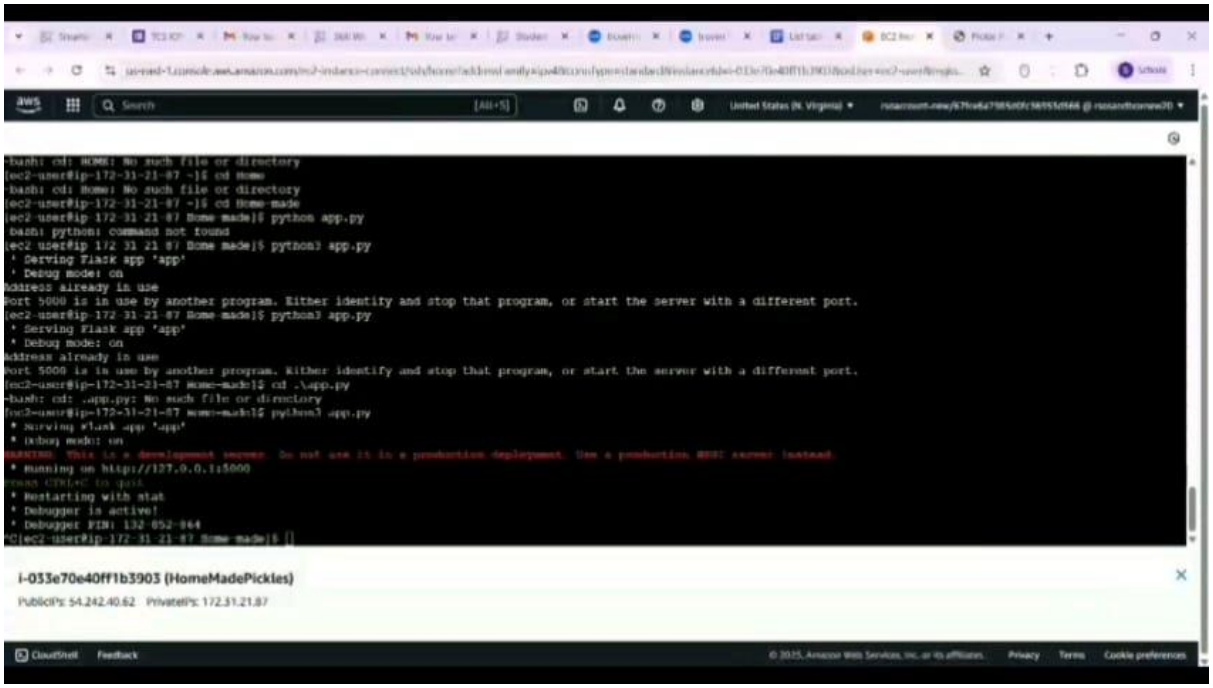
- **Activity 6.2: Clone Flask Project from GitHub**

  Run: git clone https://github.com/pallavi2005-user/Home-made.git

  Navigate to the project folder: cd Home-Made.

- **Activity 6.3: Run the Flask Application**

  Run: python3 app.py



- **Activity 6.4: Access the Website**

  Open your browser and go to: http://127.0.0.1:5000

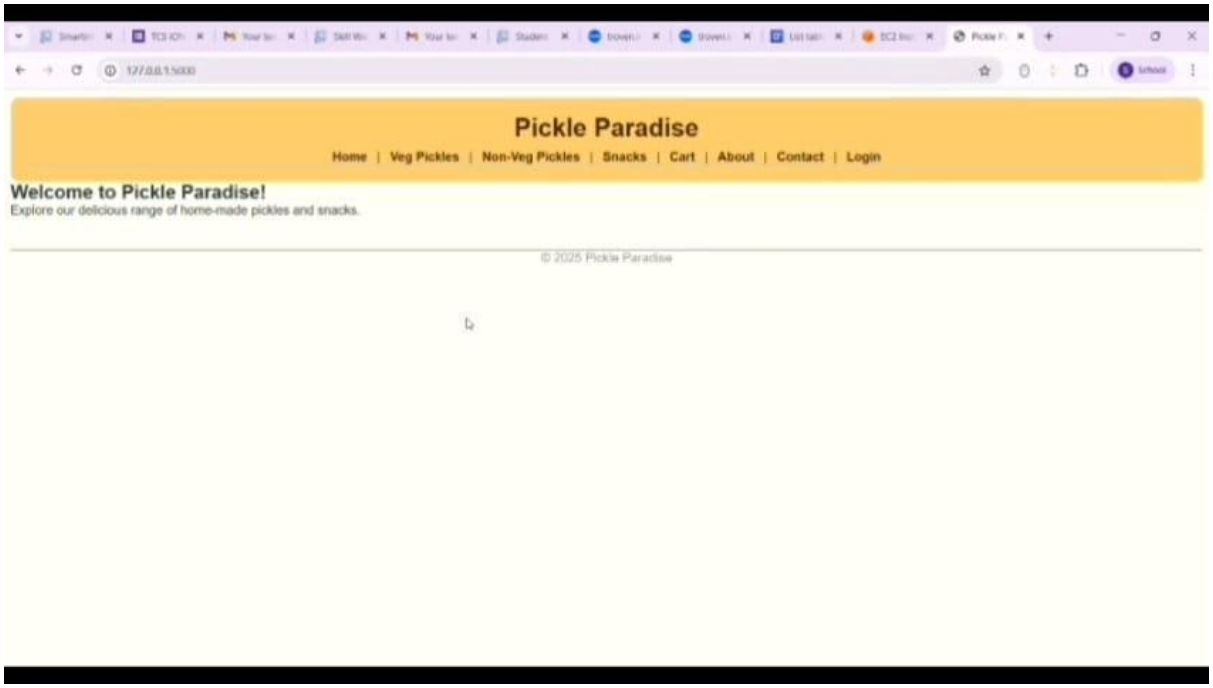
7. Testing and Deployment

Milestone 7: Testing and Deployment

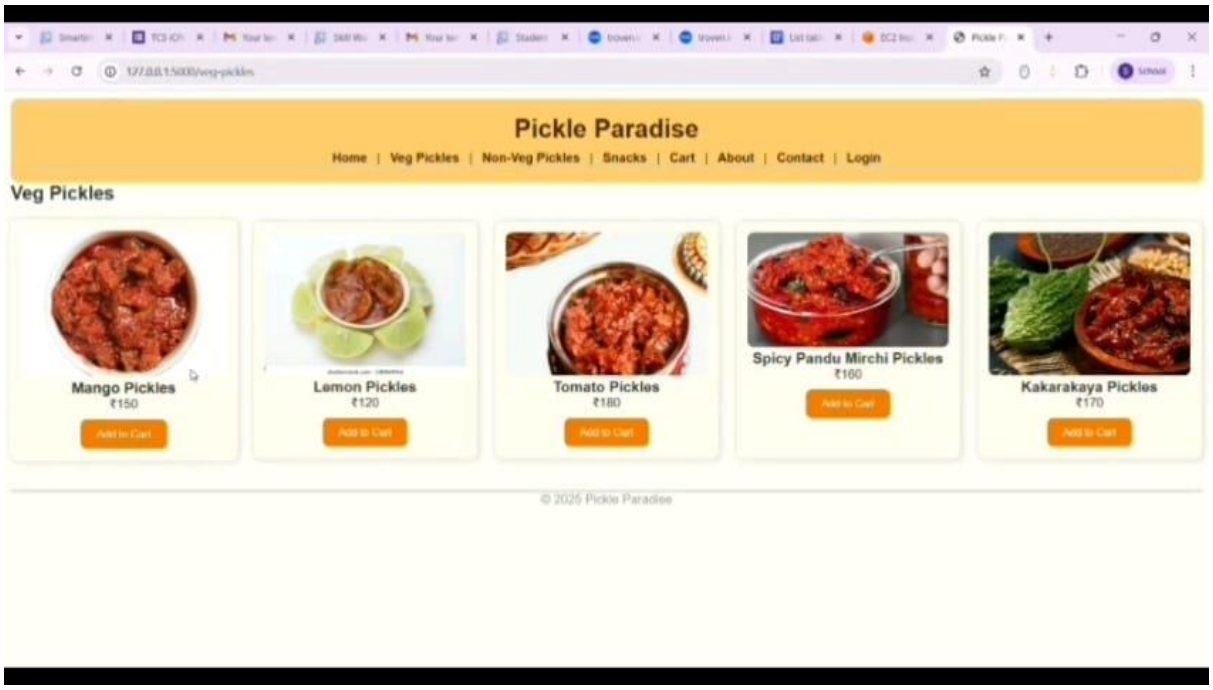
- **Activity 7.1: Functional Testing to Verify the Project**

Test each of the following pages for proper functionality, navigation, and flow:

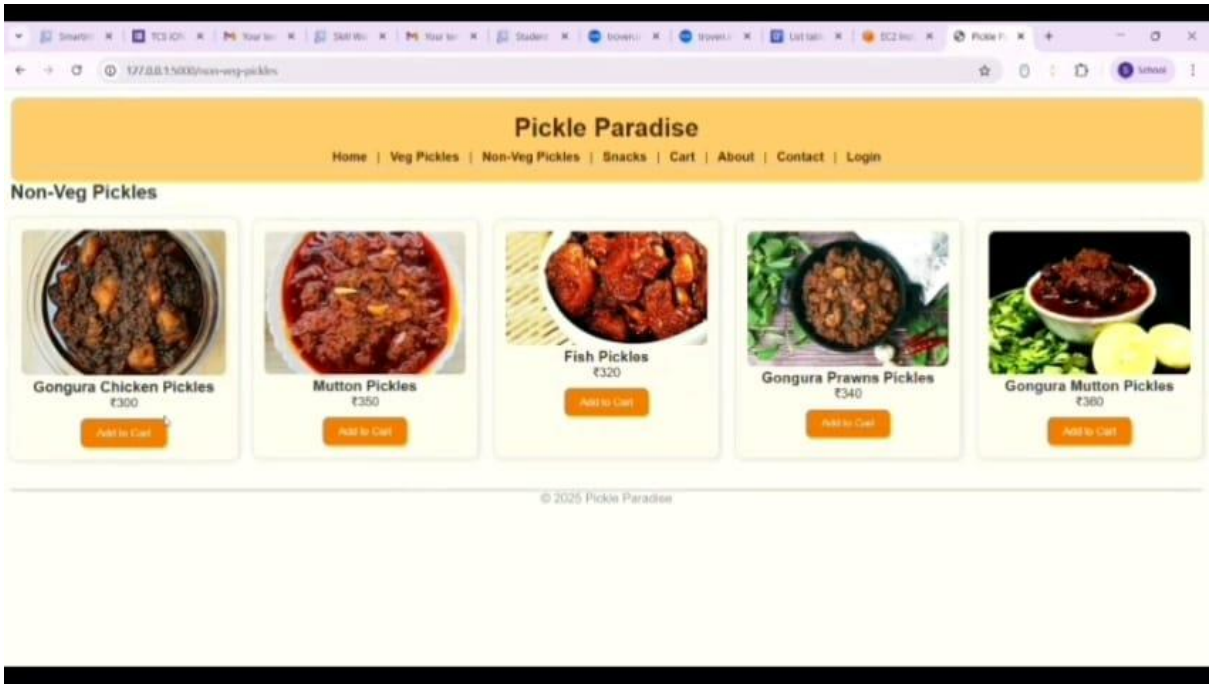
Home Page:



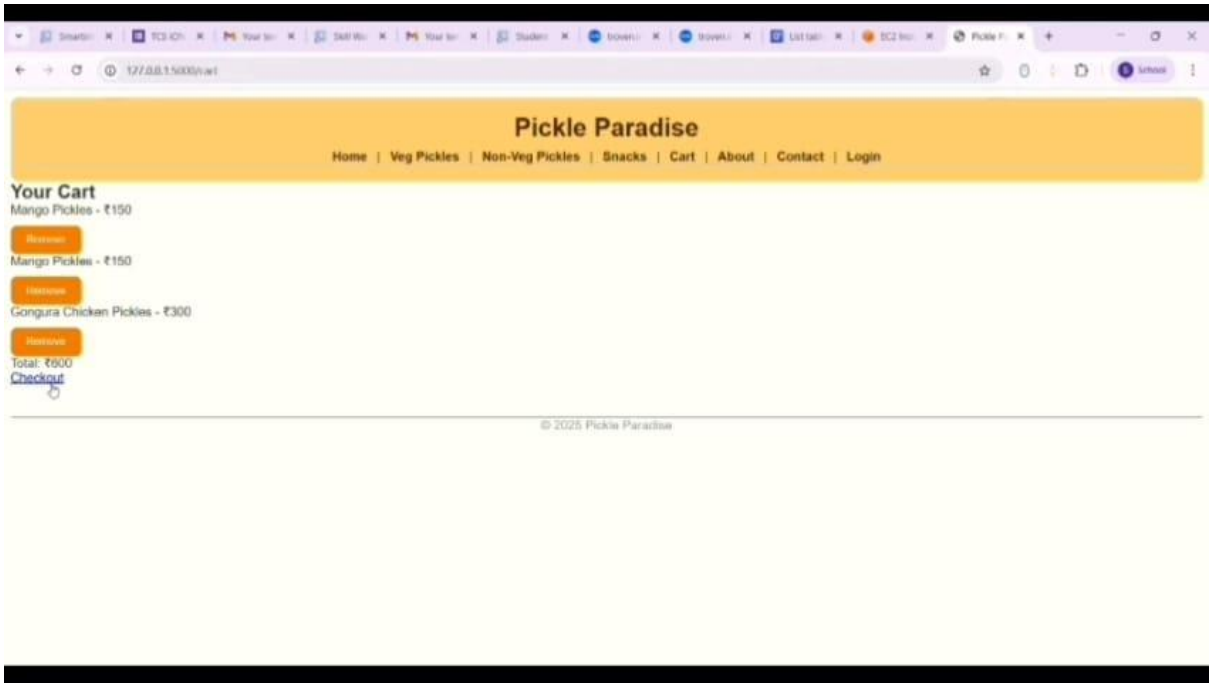
Veg Pickles Page:



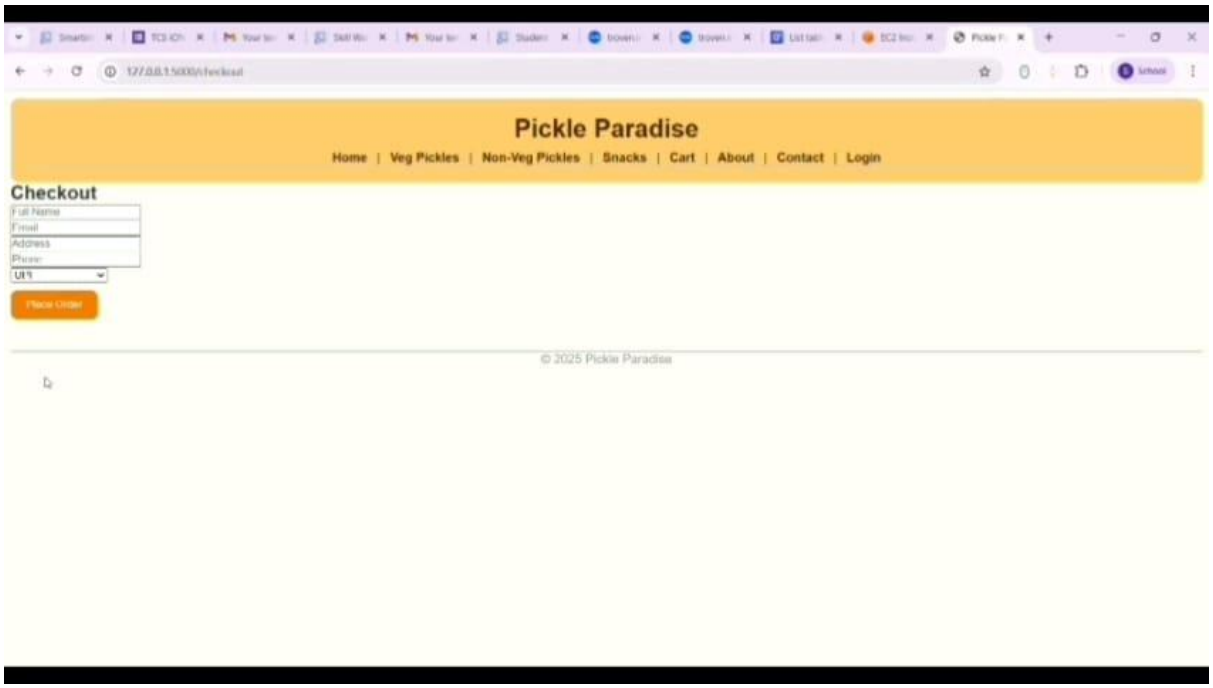
Non-Veg Pickles Page:



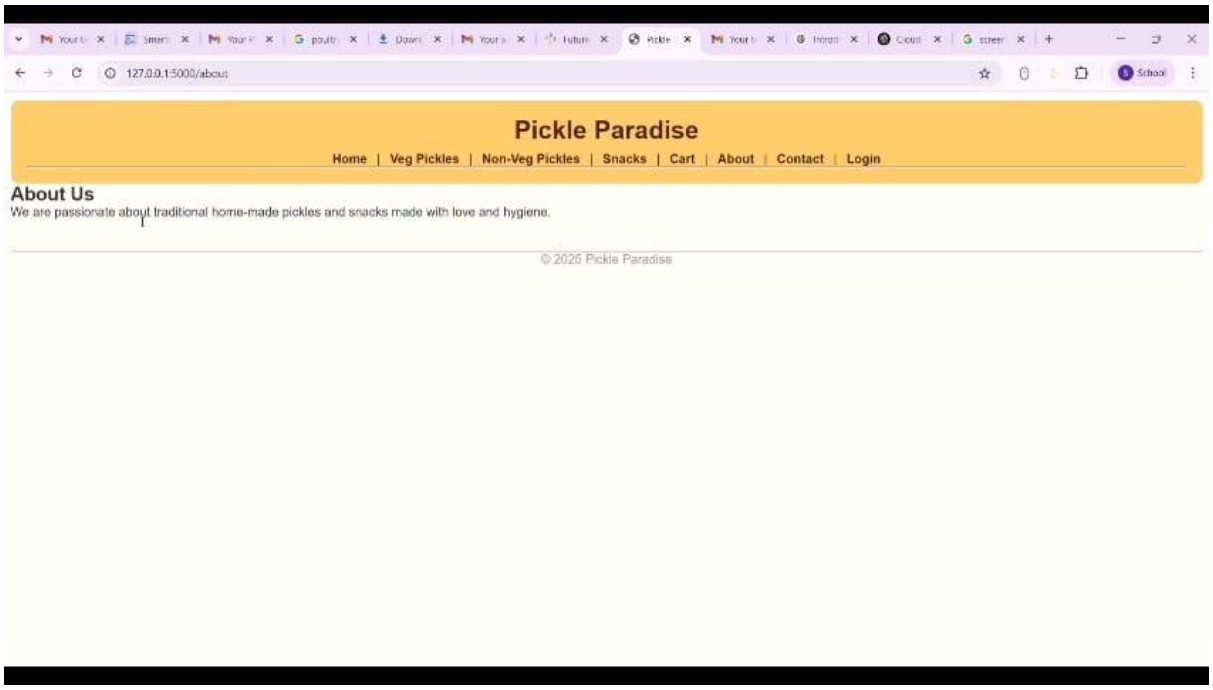
Cart Page:



Checkout Page:



About us Page:



Success Page: