

Homemade Pickles & Snacks: Taste the Best

Hardware Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

Software Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

System Required:

Projector and Audio System for presentations in all labs/classrooms Classrooms/Labs are equipped with systems or provisions for students to join sessions with their own laptops.

Description:

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

Scenarios:

Scenario 1: Scalable Order Management for High Demand

A cloud-based system ensures seamless order processing during peak user activity. For instance, during a promotional event, hundreds of users simultaneously access the platform to place orders. The backend efficiently processes requests, updates

inventory in real-time, and manages user sessions. The cloud infrastructure handles traffic spikes without performance degradation, ensuring smooth transactions and minimizing wait times.

Scenario 2: Real-Time Inventory Tracking and Updates

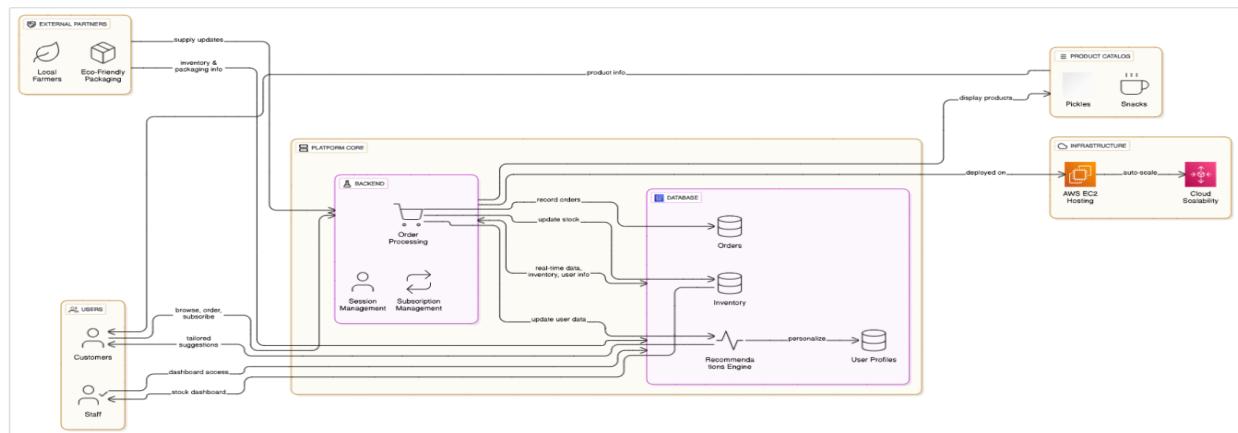
When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and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

Scenario 3: Personalized User Experience and Recommendations

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

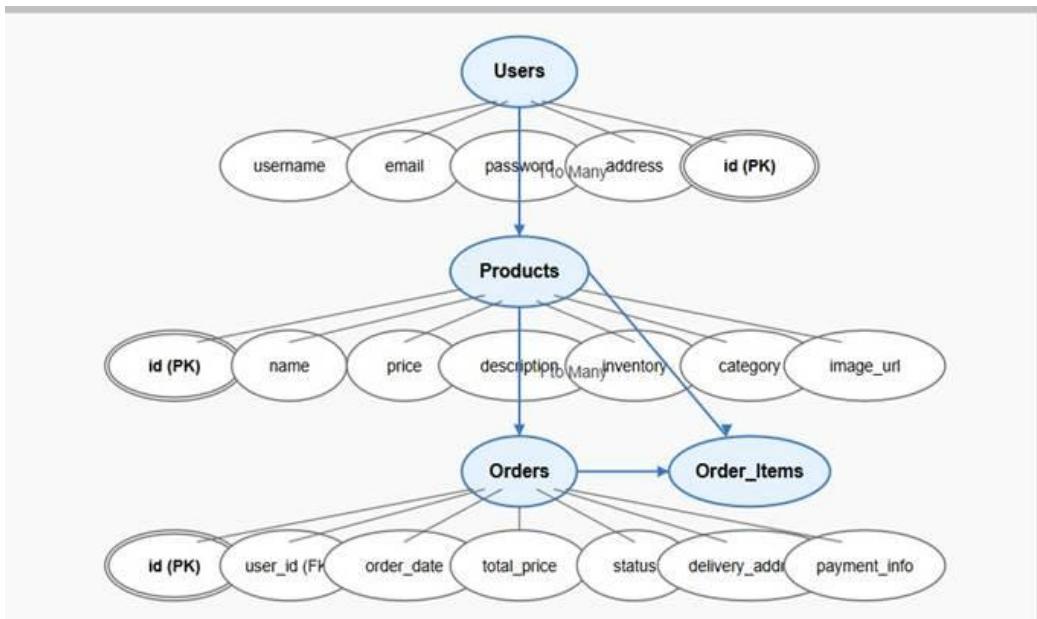
Architecture

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



Entity Relationship (ER)Diagram

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



Pre-requisites

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

Project Work Flow

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: Web Application Development and Setup

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

Important Instructions:

- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

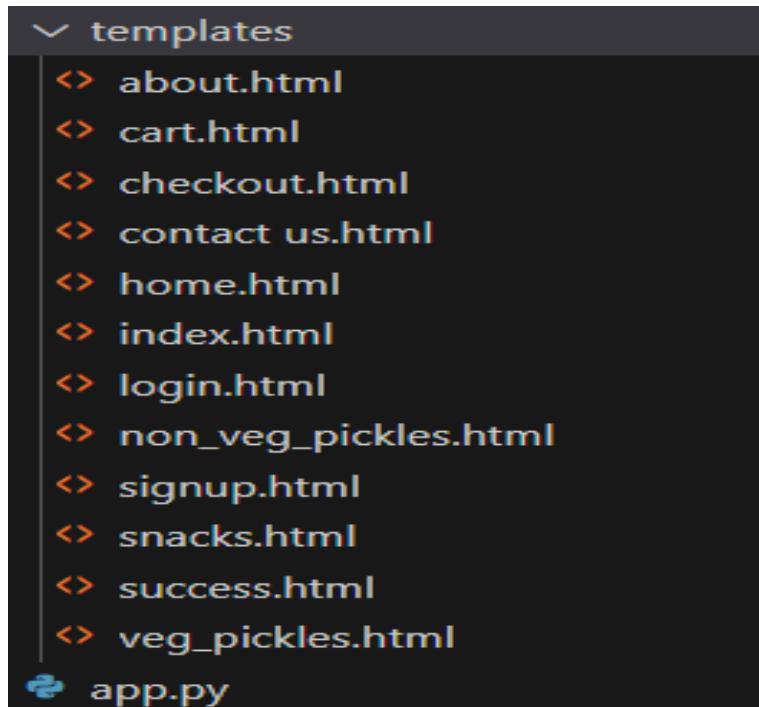
Post Troven Access Activation:

- Once troven labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
- Using the temporary credentials provided by troven Labs, securely connect your application to AWS resources.

- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

LOCAL DEPLOYMENT

- File Explorer Structure



Description of the code:

- Flask App Initialization

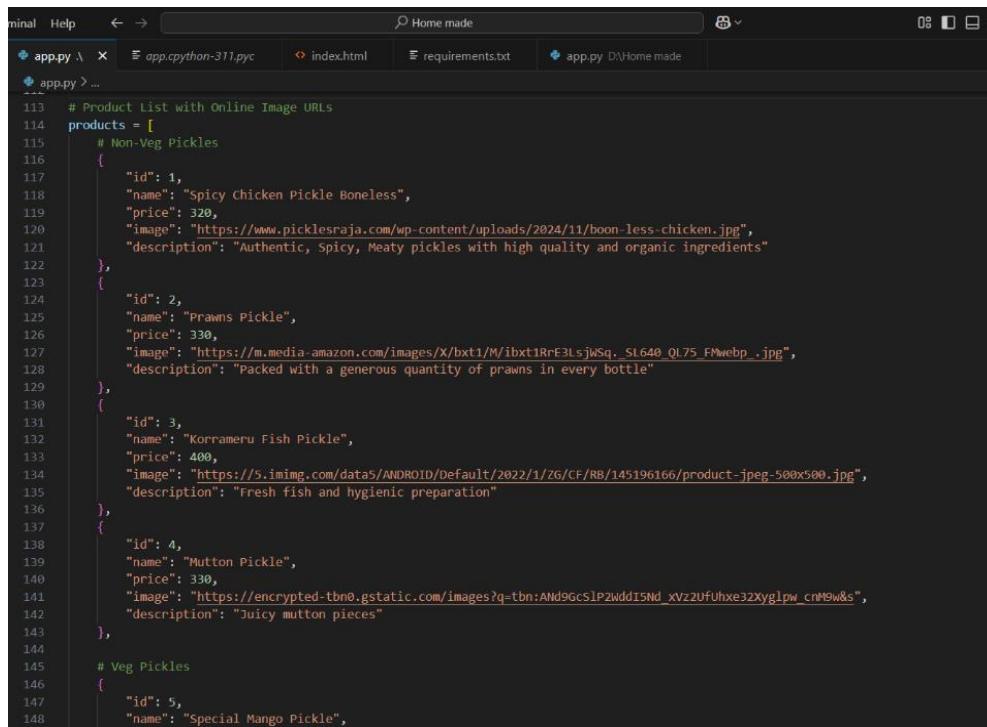
```

Terminal Help ← → ⌂ Home made
  ↗ app.py X ↗ app.cpython-311.pyc ↗ index.html ↗ requirements.txt ↗ app.py D:\Home made
  ↗ app.py > ...
  1 from flask import Flask, render_template_string, render_template , redirect, url_for, session, request, flash, jsonify
  2 from functools import wraps
  3 import smtplib
  4 import logging
  5 from email.mime.text import MIMEText
  6 from datetime import datetime
  7 from werkzeug.exceptions import BadRequest
  8 from werkzeug.security import generate_password_hash, check_password_hash
  9 import os
 10 import boto3
 11 import uuid
 12
 13
 14 if __name__ == '__main__':
 15     app.run(debug=True, host='0.0.0.0', port=int(os.environ.get("PORT", 5000)))
  
```

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



```
# DynamoDB tables
dynamodb = boto3.resource('dynamodb', region_name=region)
orders_table = dynamodb.Table('orders')
users_table = dynamodb.Table('users')
contacts_table = dynamodb.Table('contacts')
reviews_table = dynamodb.Table('reviews')
```



```
# Product List with Online Image URLs
products = [
    # Non-Veg Pickles
    {
        "id": 1,
        "name": "Spicy Chicken Pickle Boneless",
        "price": 320,
        "image": "https://www.picklesraja.com/wp-content/uploads/2024/11/boon-less-chicken.jpg",
        "description": "Authentic, Spicy, Meaty pickles with high quality and organic ingredients"
    },
    {
        "id": 2,
        "name": "Prawns Pickle",
        "price": 330,
        "image": "https://m.media-amazon.com/images/X/bxt1/M/ibxt1RrE3LsjWSq_S1640_QL75_FMwebp_.jpg",
        "description": "Packed with a generous quantity of prawns in every bottle"
    },
    {
        "id": 3,
        "name": "Korrameru Fish Pickle",
        "price": 400,
        "image": "https://5.imimg.com/data5/ANDROID/Default/2022/1/ZG/CF/RB/145196166/product-jpeg-500x500.jpg",
        "description": "Fresh fish and hygienic preparation"
    },
    {
        "id": 4,
        "name": "Mutton Pickle",
        "price": 330,
        "image": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSlP2Wdd15Nd_xVz2Ufuhxel2Xyglpw_cnM9w&s",
        "description": "Juicy mutton pieces"
    },
    # Veg Pickles
    {
        "id": 5,
        "name": "Special Mango Pickle",
    }
]
```

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

```

    /
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username'].strip()
        password = request.form['password'].strip()
        if users.get(username) == password:
            session['username'] = username
            flash("Logged in successfully!", "success")
            return redirect(url_for('products_page'))
        else:
            flash("Invalid username or password.", "error")
    return render_template_string("""
        <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}')>
            background-size: cover;
            background-position: center;
            font-family: sans-serif;
            padding: 20px;>
        <h2 style="color:#2c3e50;">Login</h2>
        <form method="POST">
            Username: <input type="text" name="username"><br><br>
            Password: <input type="password" name="password"><br><br>
            <button type="submit">Login</button>
        </form>
        <a href="{{ url_for('register') }}">Don't have an account? Register</a>
    """)

```

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

```

331
332     @app.route('/register', methods=['GET', 'POST'])
333     def register():
334         if request.method == 'POST':
335             username = request.form['username'].strip()
336             password = request.form['password'].strip()
337             if username in users:
338                 flash("Username already exists.", "error")
339             elif not username or not password:
340                 flash("Please enter both username and password.", "error")
341             else:
342                 users[username] = password
343                 flash("Registered successfully. Please login.", "success")
344                 return redirect(url_for('login'))
345         return render_template_string("""
346             <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}')>
347                 background-size: cover;
348                 background-position: center;
349                 font-family: sans-serif;
350                 padding: 20px;>
351             <h2 style="color:#2c3e50;">Register</h2>
352             <form method="POST">
353                 Username: <input type="text" name="username"><br><br>
354                 Password: <input type="password" name="password"><br><br>
355                 <button type="submit">Register</button>
356             </form>
357             <a href="{{ url_for('login') }}">Already registered? Login</a>
358         """)

```

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

```

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

```

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

```

@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'])

```

Restart Visual Studio

- Check out Route:

```

@app.route('/checkout')
@login_required
def checkout():
    cart_items = []
    total = 0

    for product_id, quantity in session.get('cart', {}).items():
        product = next((p for p in products if p['id'] == int(product_id)), None)
        if product:
            cart_items.append({
                'name': product['name'],
                'quantity': quantity,
                'items': cart,
                'price': product['price']
            })
            total += product['price'] * quantity

    return render_template_string("""
<body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');
    background-size: cover;
    background-position: center;
    font-family: sans-serif;
    padding: 20px;">
<h1 style="color:#c3e3f0;">Checkout</h1>
{%
    if cart_items %}
        <ul style="list-style:none;">
            {%
                for item in cart_items %}
                    <li>{{ item.name }} - {{ item.quantity }} - {{ item.quantity * item.price }}</li>
            {%
                endfor %}
        </ul>
        <p><strong>Total: {{ total }}</strong></p>
<form method="POST" action="{{ url_for('place_order') }}">
    Name:<input name="name"><br>
    Address:<input name="address"><br>
    email:<input name="email"><br>

```

```
app.py 3...
def checkout():
    if len(cart) == 0:
        flash("Your cart is empty.", "error")
    else:
        save_order_to_dynamodb(order_data)
        send_order_email(email, summary)

        # (Optional) SNS call is defined but not triggered here
        # send sns notification("New order received.")

    return render_template('checkout.html', cart_items=cart_items, total=total)

@app.route('/place_order', methods=['POST'])
@login_required
def place_order():
    session['cart'] = []
    flash("Your order has been placed successfully!", "success")
    session['cart'] = []
    logger.info("order placed and cart cleared.")
    return redirect(url_for('order_success'))

@app.route('/order-success')
@login_required
def order_success():
    return render_template_string("""
        <body style="background-image: url('{{ url_for('static', filename='bg.jpg') }}');>
            <div style="background-size: cover; background-position: center; font-family: sans-serif; padding: 20px;>
                <h2>Order Successful!</h2>
                <p>Thank you for your order, {{ session['username'] }}!</p>
                <a href="{{ url_for('products_page') }}> Back to Products</a><br>
                <a href="{{ url_for('logout') }}> Logout</a>
            </div>
        </body>
    """)
```

Milestone 2: AWS Account Setup

Important Notice: Use troven Labs for AWS Access

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

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

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

Reminder: You must complete the Web Development task before gaining access to troven. Once accessed, the AWS Console via troven is available for only 3 hours—please plan your work accordingly.

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

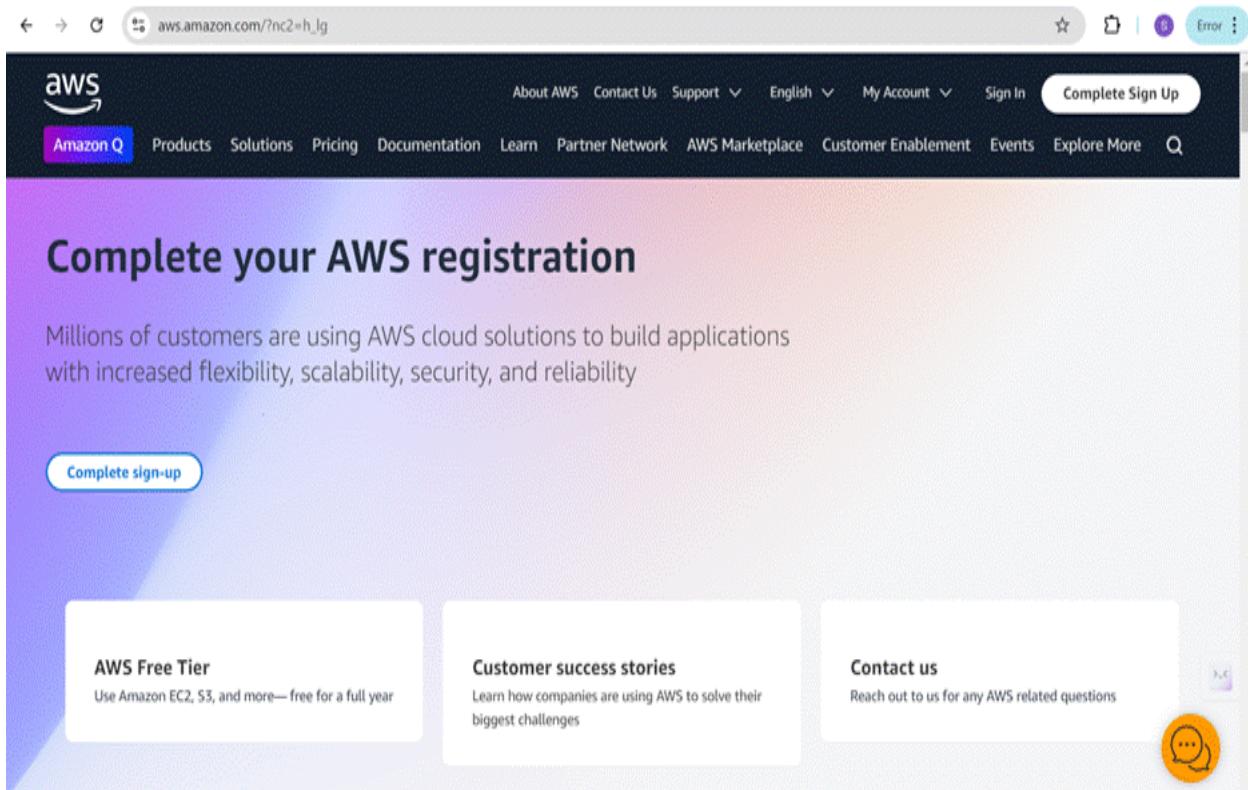
Please refer the below link -

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

AWS Account Setup and Login

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

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





Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.

Sign up for AWS

Root user email address
Used for account recovery and as described in the [AWS Privacy Notice](#)

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

[Sign in to an existing AWS account](#)

- Log in to the AWS Management Console
- After setting up your account, log in to the [AWS Management Console](#).

Sign in

Root user
Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user
User within an account that performs daily tasks. [Learn more](#)

Root user email address

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

AI Use Case Explorer

Discover AI use cases, customer success stories, and expert-curated implementation plans

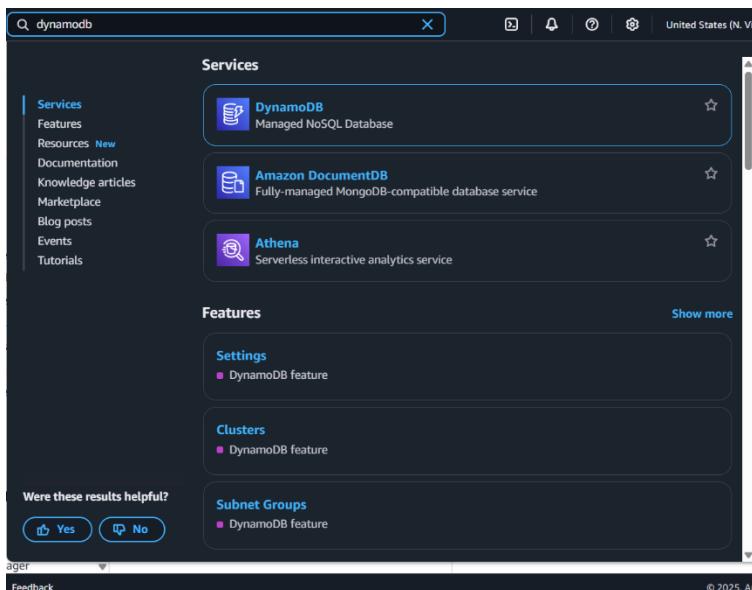
[Explore now >](#)

Milestone 3: DynamoDB Database Creation and Setup

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

Navigate to the DynamoDB

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



The screenshot shows the AWS DynamoDB dashboard. The left sidebar provides navigation for various services and features. The main dashboard displays sections for Alarms, DAX clusters, and a 'What's new' update about AWS Cost Management. A prominent 'Create resources' button is located on the right side of the dashboard.

Create a DynamoDB table for storing data

- Create Users table with partition key “Username” 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

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](#)

The Users table was created successfully.

DynamoDB > Tables

Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
Users	Active	email (\$)		0	Off	Provisioned (\$)	Provisioned (\$)	0 bytes

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



☰ [DynamoDB](#) > [Tables](#) > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Orders

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and performance.

Order_id

String



1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name

String



1 to 255 characters and case sensitive.



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

Tags

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

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create table

Tables (2) [Info](#)

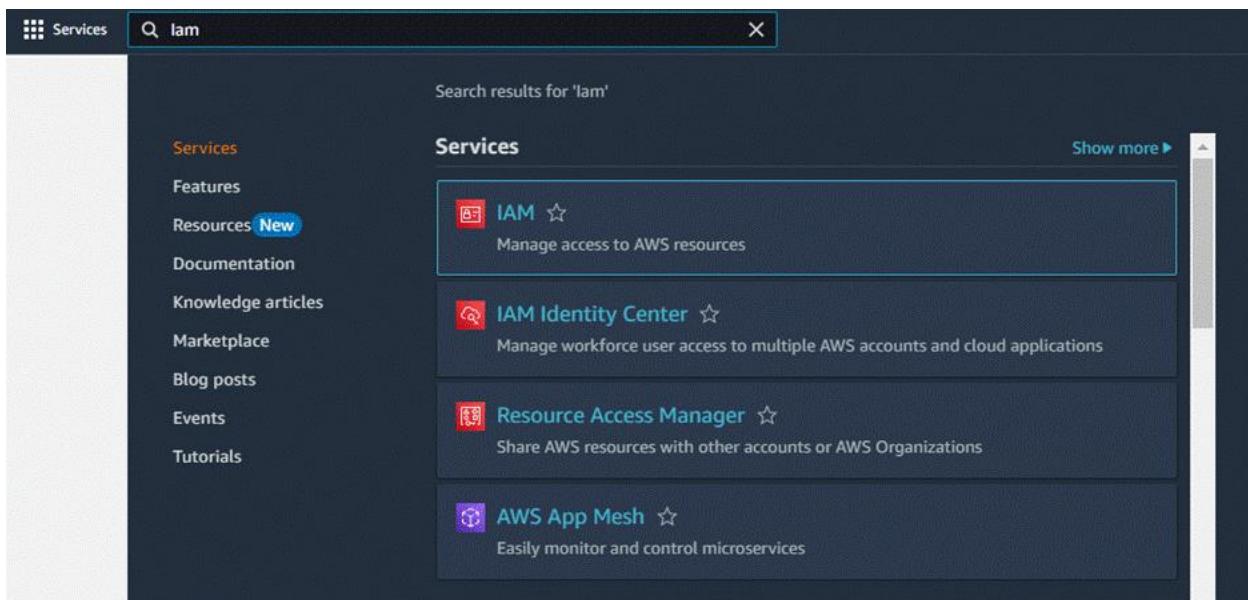
<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protecti
<input type="checkbox"/>	Orders	Active	order_id (S)	-	0	0	Off
<input type="checkbox"/>	Users	Active	username (S)	-	0	0	Off

Milestone 4: IAM Role Setup

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

Create IAM Role.

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



Screenshot of the AWS IAM Roles page. The sidebar shows 'Identity and Access Management (IAM)'. The main area shows a table of roles with columns for 'Role name', 'Trusted entities', and 'Last activity'. A search bar and buttons for 'Delete' and 'Create role' are at the top right.

Screenshot of the 'Create role' wizard Step 1: Select trusted entity. It shows options for 'AWS service', 'AWS account', 'AWS Lambda', and 'Custom trust policy'. Below is a section for 'Service or use case' with a dropdown set to 'EC2'.

Screenshot of the 'Create role' wizard Step 2: Add permissions. It shows the 'Permissions policies' section with a search bar for 'AmazonDynamoDB'. Two policies are listed: 'AmazonDynamoDBFullAccess' and 'AmazonDynamoDBReadOnlyAccess'. A 'Set permissions boundary - optional' link is at the bottom.

Screenshot of the AWS IAM Roles page after creating a new role. A green success message says 'Role EC2_DynamoDB_Role created.' The main area shows a table of roles with columns for 'Role name', 'Trusted entities', and 'Last activity'. The table includes rows for 'AWSServiceRoleForAmazonEKS' and 'AWSServiceRoleForAmazonEKSNodegroup'.

Attach Policies

Attach the following policies to the role:

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

The screenshot shows the 'Create New Role' wizard in the AWS IAM console. It consists of three main steps:

- Step 1: Select trusted entities**: Shows a JSON-based trust policy allowing EC2 instances to assume the role. The policy includes a service principal for 'ec2.amazonaws.com' and a condition that restricts the action to specific IP ranges (10.0.0.0/16 and 10.0.1.0/24).
- Step 2: Add permissions**: Displays a 'Permissions policy summary' table. It lists two managed policies:
 - AmazonDynamoDBFullAccess**: AWS managed, Attached as Permissions policy.
 - AmazonSNSFullAccess**: AWS managed, Attached as Permissions policy.
- Step 3: Add tags**: A section for optional tags, currently empty.

At the bottom right of the wizard, there are 'Cancel', 'Previous', and 'Create role' buttons.

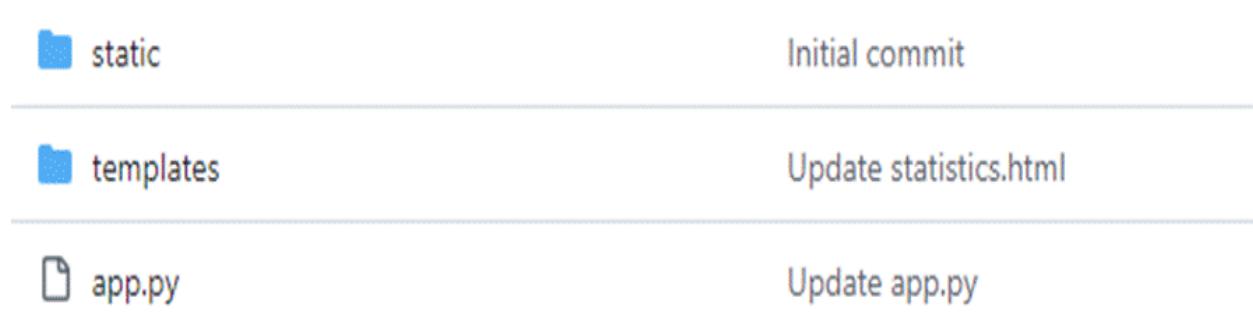
The screenshot shows the 'sns_Dynamodb_role' details page in the AWS IAM console. It includes the following sections:

- Summary**: Shows creation date (October 13, 2024), ARN (arn:aws:iam::557690616836:role/sns_Dynamodb_role), and instance profile ARN (arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role). Last activity is shown as 6 days ago.
- Permissions**: Shows two attached managed policies:
 - AmazonDynamoDBFullAccess**: AWS managed, Attached as Permissions policy.
 - AmazonSNSFullAccess**: AWS managed, Attached as Permissions policy.

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.

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



The image shows a GitHub repository page for `Homedae-pickles-and-snacks-app`. The repository is public and has 1 branch and 0 tags. The code tab is selected, showing the following files:

File	Action	Last Commit
<code>__pycache__</code>	Add files via upload	last week
<code>static</code>	Add files via upload	last week
<code>templates</code>	Add files via upload	last week
<code>about.html</code>	Add files via upload	last week
<code>app.py</code>	Add files via upload	4 days ago
<code>bg.jpg</code>	Add files via upload	last week
<code>cart.html</code>	Add files via upload	last week
<code>checkout.html</code>	Add files via upload	last week
<code>contact.us.html</code>	Add files via upload	last week

The repository has 0 stars, 0 forks, and 0 releases. The languages used are HTML (89.6%) and Python (10.4%).

Launch an EC2 instance to host the Flask

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

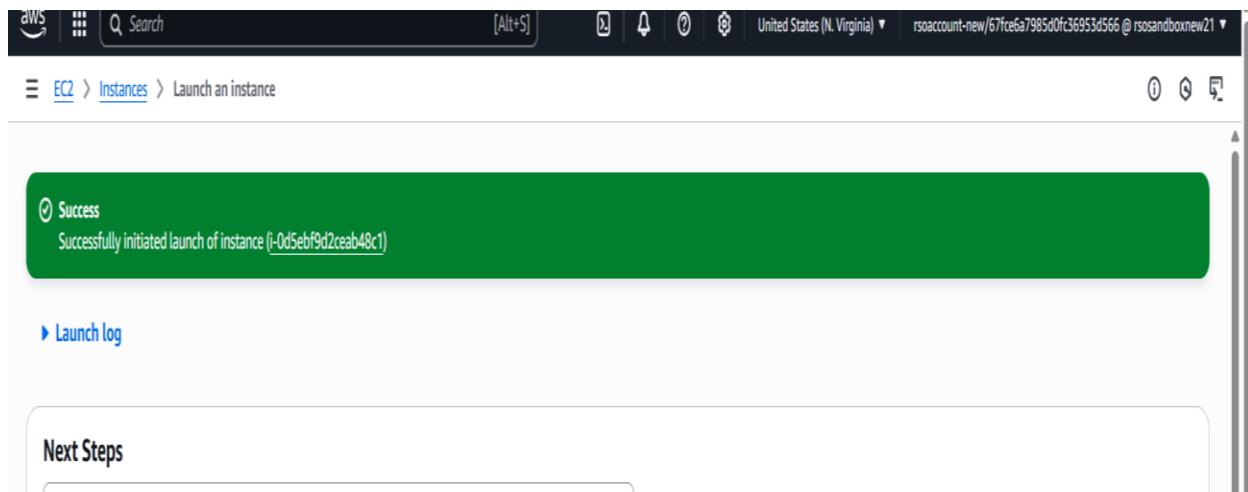
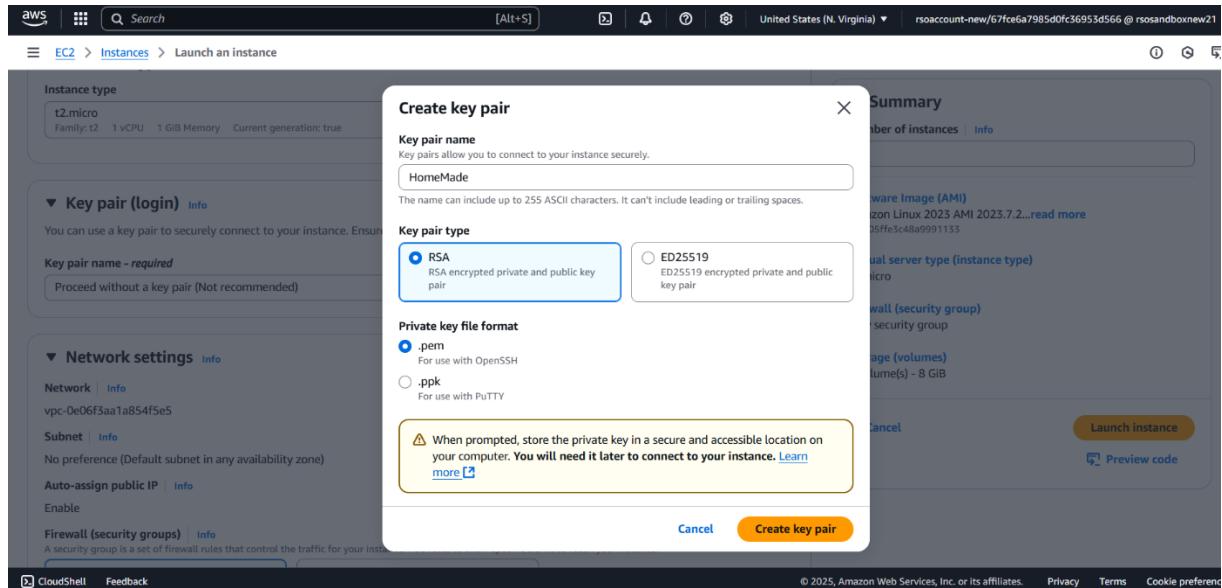
The screenshot shows the AWS search interface with the query 'ec2' entered in the search bar. The results are categorized under 'Services' and 'Features'. The 'EC2' service card is highlighted, showing its icon (a server), name, and description: 'Virtual Servers in the Cloud'. Other cards include 'EC2 Image Builder' (managed service for OS images) and 'EC2 Global View' (global dashboard and search functionality). On the left sidebar, there are links for Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. At the bottom, there's a section asking 'Were these results helpful?' with 'Yes' and 'No' buttons.

- Click on Launch instance to launch EC2 instance

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed, showing the 'EC2' navigation item. The main content area is titled 'Instances' and includes a search bar, filters for 'Name', 'Instance ID', 'Instance state', 'Instance type', 'Status check', 'Alarm status', 'Availability Zone', and 'Public IP'. A message states 'No instances' and 'You do not have any instances in this region'. Below this is a large blue 'Launch instances' button. The bottom of the page has a 'Select an instance' section and some footer links.

- 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.

▼ Network settings [Info](#)

VPC - required [Info](#)
 vpc-03cdc7b6f19dd7211 (default) ▾ [Edit](#)

Subnet [Info](#)
 No preference ▾ [Edit](#) [Create new subnet](#) [Edit](#)

Auto-assign public IP [Info](#)
 Enable ▾ [Edit](#)

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)
 A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group [Edit](#) Select existing security group [Edit](#)

Security group name - required
 launch-wizard [Edit](#)

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-:/()#,@[]+=;&;{}!\$*

Description - required [Info](#)
 launch-wizard created 2024-10-13T17:49:56.622Z [Edit](#)

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) [Remove](#)

Type Info ssh ▾ Edit	Protocol Info TCP Edit	Port range Info 22 Edit
Source type Info Anywhere ▾ Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0) [Remove](#)

Type Info HTTP ▾ Edit	Protocol Info TCP Edit	Port range Info 80 Edit
Source type Info Custom ▾ Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0) [Remove](#)

Type Info Custom TCP ▾ Edit	Protocol Info TCP Edit	Port range Info 5000 Edit
Source type Info Custom ▾ Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

[Add security group rule](#)

The screenshot shows the AWS EC2 'Launch an Instance' success page. At the top, a green banner indicates a successful launch of instance i-00186102fbac29c. Below the banner, there's a 'Launch log' link and a 'Next Steps' section. The 'Next Steps' section contains ten cards with various configuration options:

- Create billing and free tier usage alerts**: To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds. Includes a 'Create billing alerts' button.
- Connect to your instance**: Once your instance is running, log into it from your local computer. Includes a 'Connect to instance' button.
- Connect an RDS database**: Configure the connection between an EC2 instance and a database to allow traffic flow between them. Includes a 'Connect an RDS database' button.
- Create EBS snapshot policy**: Create a policy that automates the creation, retention, and deletion of EBS snapshots. Includes a 'Create EBS snapshot policy' button.
- Manage detailed monitoring**: Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 1-minute period. Includes a 'Manage detailed monitoring' button.
- Create Load Balancer**: Create an application, network gateway or classic Elastic Load Balancer. Includes a 'Create Load Balancer' button.
- Create AWS budget**: AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage from a single location. Includes a 'Create AWS budget' button.
- Manage CloudWatch alarms**: Create or update Amazon CloudWatch alarms for the instance. Includes a 'Manage CloudWatch alarms' button.
- Disaster recovery for your instances**: Recover the instances you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (EDR). Includes a 'Disaster recovery for your instances' button.
- Monitor for suspicious runtime activities**: Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and unauthorized behavior, with near real-time visibility into on-host activities occurring across your Amazon EC2 workloads. Includes a 'Monitor for suspicious runtime activities' button.
- Get instance screenshot**: Capture a screenshot from the instance and view it as an image. This is useful for troubleshooting an unreachable instance. Includes a 'Get instance screenshot' button.
- Get system log**: View the instance's system log to troubleshoot issues. Includes a 'Get system log' button.

At the bottom right of the 'Next Steps' section is a 'View all Instances' button.

- 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.

The screenshot shows the AWS EC2 Instances dashboard. At the top, a green banner indicates a successful attachment of the EC2_DynamoDB_Role to instance i-0d5ebf9d2ceab48c1. Below the banner, the dashboard displays the following information:

- Instances (1/1) Info**: Shows 1 instance (HomeMadePic...), which is running (Status: Running), t2.micro (Instance type), us-east-1c (Availability Zone), and has a Public IP of ec2-13-2...
- Actions**: Buttons for Connect, Instance state, Actions, and Launch instances.
- Filtering and Sorting**: A search bar to find instances by attribute or tag (case-sensitive) and a dropdown menu for 'All states'.
- Table Headers**: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP.
- Table Data**: One row for the instance HomeMadePic..., showing its details: Name (HomeMadePic...), Instance ID (i-0d5ebf9d2ceab48c1), Status (Running), Instance type (t2.micro), Status check (2/2 checks passed), Alarm status (View alarms +), Availability Zone (us-east-1c), and Public IP (ec2-13-2...).

[EC2](#) > [Instances](#) > i-001861022fbcac290 [Info](#)

Updated less than a minute ago

Instance ID i-001861022fbcac290	Public IPv4 address -	Private IPv4 addresses 172.31.3.5
IPv6 address -	Instance state <input checked="" type="radio"/> Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Auto-assigned IP address -	VPC ID vpc-05cdcb6f19dd7211	Auto Scaling Group name -
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9a9	
IMDSv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	

[Details](#) [Status and alarms](#) [Monitoring](#) [Security](#) [Networking](#) [Storage](#) [Tags](#)

[EC2](#) > [Instances](#) > i-001861022fbcac290 [Info](#)

Updated less than a minute ago

Instance ID i-001861022fbcac290	Public IPv4 address -	Private IPv4 addresses 172.31.3.5
IPv6 address -	Instance state <input checked="" type="radio"/> Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Auto-assigned IP address -	VPC ID vpc-05cdcb6f19dd7211	Auto Scaling Group name -
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9a9	
IMDSv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	

[C](#) [Connect](#) [Instance state](#) [Actions](#)

- [Connect](#)
- [Manage instance state](#)
- [Instance settings](#)
- [Networking](#)
- [Security](#)
- [Image and templates](#)
- [Modify IAM role](#)
- [Monitor and troubleshoot](#)

[EC2](#) > [Instances](#) > i-001861022fbcac290 > [Modify IAM role](#)

Modify IAM role [Info](#)

Attach an IAM role to your instance.

Instance ID
[i-001861022fbcac290 \(InstantLibraryApp\)](#)

IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

[▼](#)
[C](#)
[Create new IAM role](#)

Update IAM role
[Cancel](#)

Connect to instance Info

Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

EC2 Instance Connect **Session Manager** **SSH client** **EC2 serial console**

⚠ Port 22 (SSH) is open to all IPv4 addresses

Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: **13.233.177.0/29**. [Learn more](#).

Instance ID
 i-001861022fbcac290 (InstantLibraryApp)

Connection Type

Connect using EC2 Instance Connect
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

Connect using EC2 Instance Connect Endpoint
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
 13.200.229.59

IPv6 address

—

Username
 Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

X

Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel
Connect

```
A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
      #
      #####
      Amazon Linux 2023
      ###\#
      # \#/
      https://aws.amazon.com/linux/amazon-linux-2023
      V ,-->
      ~~~
      ~~~
      ~~~
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
fec2-user@ip-172-31-3-5 ~%
```

Milestone 6: Deployment on EC2

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

Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

- sudo yum update -y
- sudo yum install python3 git
- sudo pip3 install flask boto3

Verify Installations:

- flask --version
- git --version

Clone Your Flask Project from GitHub

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

- Run: '[git clone:https://github.com/kilarukusuma-12/Homemadepicklesandsnacks.git](https://github.com/kilarukusuma-12/Homemadepicklesandsnacks.git)'
- Note: change your-github-username and your-repository-name with your credentials
- here:[here:https://github.com/kilarukusuma-12/Homemadepicklesandsnacks.git](https://github.com/kilarukusuma-12/Homemadepicklesandsnacks.git)
- This will download your project to the EC2 instance.

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

- cd Homemadepickles and snacks
- 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

```
ModuleNotFoundError: No module named 'boto3'

[ec2-user@ip-172-31-15-149 Home made pickles1]$ sudo pip3 install boto3
Collecting boto3
  Downloading boto3-1.37.23-py3-none-any.whl (139 kB)
    ██████████ | 139 kB 8.1 MB/s
Collecting s3transfer<0.12.0,>=0.11.0
  Downloading s3transfer-0.11.4-py3-none-any.whl (84 kB)
    ██████████ | 84 kB 6.7 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<1.38.0,>=1.37.23
  Downloading botocore-1.37.23-py3-none-any.whl (13.4 MB)
    ██████████ | 13.4 MB 38.3 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->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<1.38.0,>=1.37.23->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.37.23 botocore-1.37.23 s3transfer-0.11.4
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-15-149 Home made pickles1]$ sudo flask run --host=0.0.0.0 --port=5000
 * 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://172.31.15.149:5000
Press CTRL+C to quit
```

Access the website through:

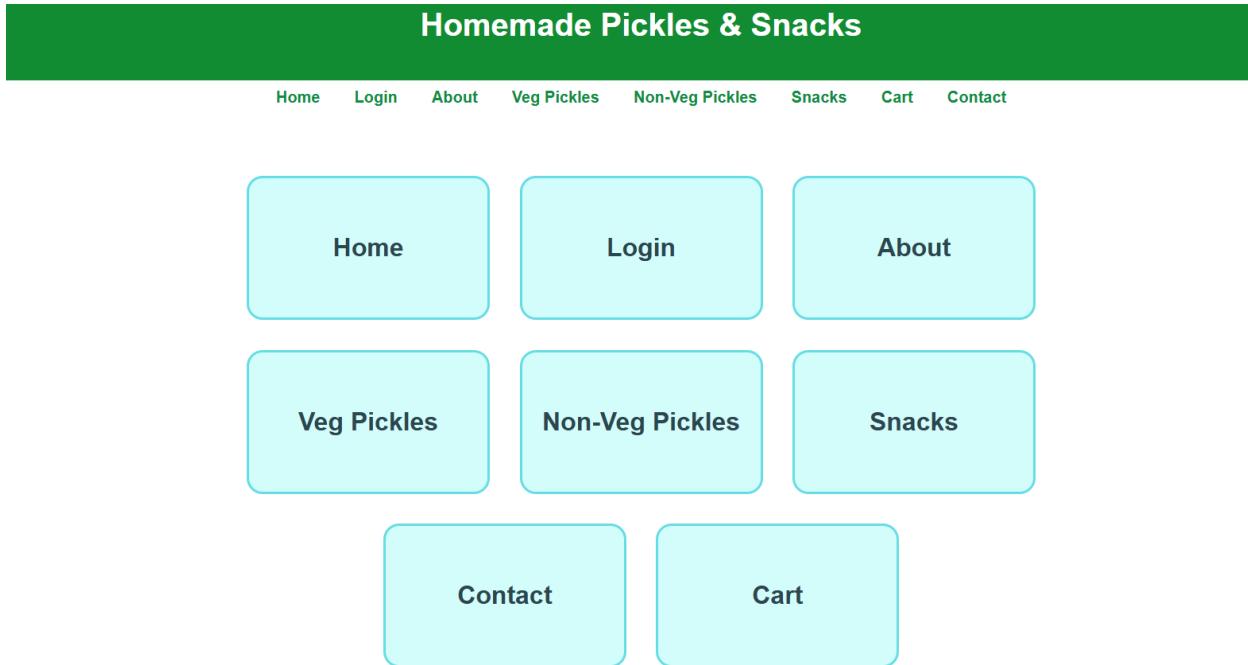
Public IPs: <http://3.88.171.131:5001>

Milestone 7: Testing and Deployment

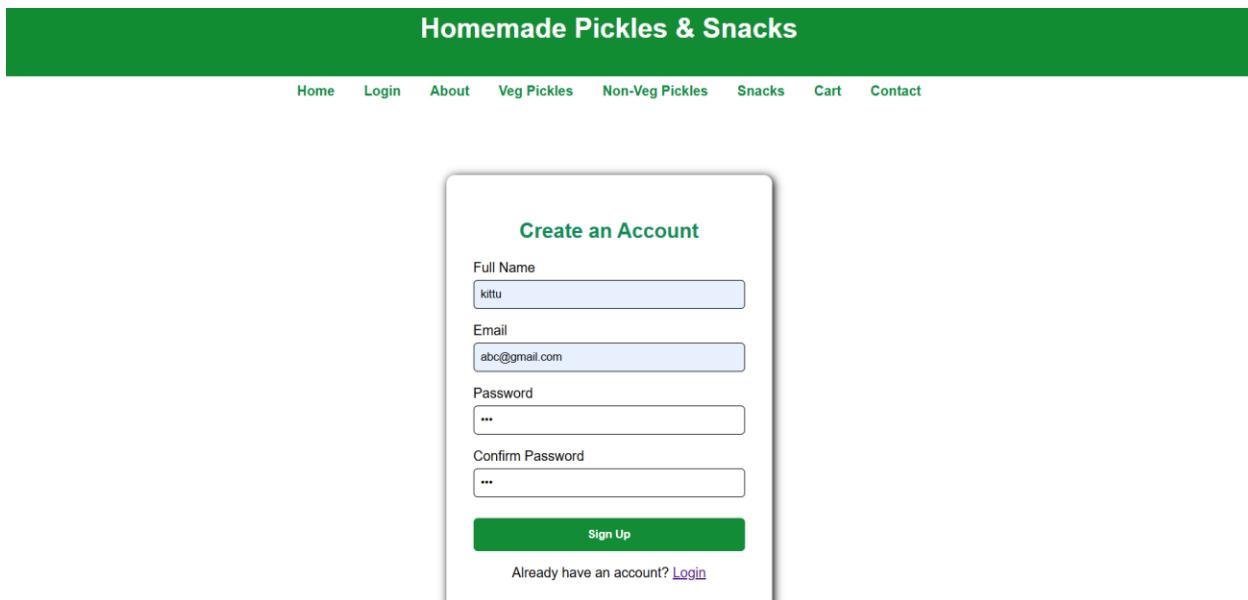
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

Functional testing to verify the Project

Index page:



Signup page:



Login page:

Homemade Pickles & Snacks

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Login

Email

Password

[Login](#)

Don't have an account? [Sign up](#)

Home page:

Homemade Pickles & Snacks

Authentic tastes from our kitchen to yours!

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Featured Products



Mango Pickle

Raw mango chunks in fiery red chili-garlic masala.



Chicken Pickle

Juicy, tender chicken cooked to perfection.



Masala chat

Masala chaat is a spicy and tangy Indian street food bursting with bold flavors.

Veg-pickles:

Homemade Pickles & Snacks

Purely Plant-Based, Bursting with Flavor

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Our Vegetarian Pickles



Mango Pickle

Raw mango chunks in fiery red chili-garlic masala.

₹150



Gongura Pickle

Tangy gongura leaves pickled with spices & tradition.

₹140



Lemon Pickle

Zesty lemons steeped in mustard, fenugreek & oil.

₹130



Tomato Pickle

Rich, spicy, and bursting with flavor.

₹120

non-veg pickles:

Homemade Pickles & Snacks

Bold, Spicy, and Packed with Flavor!

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Non-Vegetarian Pickles



Chicken Pickle

Juicy, tender chicken cooked to perfection.

₹200



Mutton Pickle

Mutton pickle with bold and spicy flavor.

₹300



Fish Pickle

A coastal delicacy made with premium fish.

₹250



Prawn Pickle

Succulent prawns infused with mustard and chili.

₹250

Snacks:

Homemade Pickles & Snacks

South Indian Crunch & Munch Specials

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Our Handmade Snacks



Masala chat

Masala chaat is a spicy and tangy Indian street food bursting with bold flavors.

₹80

[Add to Cart](#)



Samosa

Golden crispy gram flour pearls seasoned with bold flavors.

₹70

[Add to Cart](#)



Popcorn

Melt-in-your-mouth buttery goodness with every pop!

₹30

[Add to Cart](#)



Chips

Your favorite snack, now with extra crunch and flavor!

₹60

[Add to Cart](#)

Cart page:

Homemade Pickles & Snacks

Your Cart

[Home](#) [Login](#) [About](#) [Veg Pickles](#) [Non-Veg Pickles](#) [Snacks](#) [Cart](#) [Contact](#)

Product	Quantity	Unit Price	Subtotal	Action
Tomato Pickle	1	₹120	₹	Remove
Fish Pickle	1	₹250	₹	Remove
Mutton Pickle	1	₹300	₹	Remove
Prawn Pickle	1	₹250	₹	Remove
Samosa	1	₹70	₹	Remove
Masala chat	1	₹80	₹	Remove
Popcorn	1	₹30	₹	Remove
Chips	1	₹60	₹	Remove

Total: ₹1160

[Proceed to Checkout](#)

Checkout page:

Home Login About Veg Pickles Non-Veg Pickles Snacks Cart Contact

Billing & Shipping Details

Full Name
kittu

Email
abc@gmail.com

Phone Number
12121221

Shipping Address
guntur

Additional Notes (optional)

Place Order

Order success page:

Homemade Pickles & Snacks

Home Login About Veg Pickles Non-Veg Pickles Snacks Cart Contact

Thank You!

Your order has been placed successfully.

We'll be in touch soon with the delivery details. Get ready to taste tradition!

Back to Home

Contact page:

The screenshot shows a contact form titled "Contact Us". The form includes fields for "Your Name" (with "kittu" entered), "Your Email" (with "abc@gmail.com" entered), and "Your Message" (with "thank you" entered). A green "Send Message" button is at the bottom.

Homemade Pickles & Snacks

We'd love to hear from you!

Home Login About Veg Pickles Non-Veg Pickles Snacks Cart Contact

Contact Us

Your Name
kittu

Your Email
abc@gmail.com

Your Message
thank you

Send Message

About page:

"We chose to give the About page a different background to make it stand out, as it tells the unique story of our homemade pickles. This helps users clearly understand what makes our products special."

The screenshot shows the "Our Story" section of the about page. It contains a paragraph about the company's heritage, a quote about regional diversity, a quote about modernizing traditional recipes, and a quote about the emotional connection to food.

Homemade Pickles & Snacks

Home Login About Veg Pickles Non-Veg Pickles Snacks Cart Contact

Our Story

In a quiet kitchen filled with the rich aroma of spices, our story began — not in a factory, but in a home where recipes were passed down like heirlooms, from grandmother to mother, and now to us.

"From the tangy gongura of Andhra to the fiery red chili masala of Telangana, our pickles reflect the vibrant diversity of Indian kitchens. We take pride in preserving regional tastes, handcrafted using ingredients sourced from local farmers and spice markets."

"We've taken age-old recipes and given them a contemporary touch — carefully balancing bold flavors with health-conscious choices. Cold-pressed oils, hygienic kitchens, and sustainable packaging meet traditional masalas and time-honored techniques."

Just real food, real flavor, and a whole lot of love — just like how your mom or grandma would make.

"Our pickles and snacks are more than products — they are memories in a jar. They carry the warmth of your grandmother's kitchen, the smell of festive afternoons, and the crunch of monsoon evenings. Food has a way of bringing people together — and we're here to keep those connections alive."

Database updates:

1. Users table:

✓ Completed. Read capacity units consumed: 2 X

Items returned (3)

C Actions ▾ Create item

< 1 > ⚙️ ✎

	username (String)	email	password
<input type="checkbox"/>	Shiva	kilarukusu...	scrypt:32768:8:1\$W5tA59Z7nQjLXbt\$d6bfef2b3e14bbe9d3d3e3f1c...
<input type="checkbox"/>	kusuma	<empty>	<empty>
<input type="checkbox"/>	Alekhya	alekhya@g...	scrypt:32768:8:1\$EwCDTI0iaGcKutw3\$cd5dbf5c12ec17cb518f7c15cd...

2. Orders table:

✓ Completed. Read capacity units consumed: 2 X

Items returned (4)

C Actions ▾ Create item

< 1 > ⚙️ ✎

	order_id (String)	address	items	name	payment_met...	phone
<input type="checkbox"/>	7c6bd84e-f2c7-4fe0...	Kothur	[{"M": {"n..."}]	Siri	cod	8187810
<input type="checkbox"/>	3de0fe0c-9539-4fb6-...	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849889
<input type="checkbox"/>	fbc41d6d-d6f2-4158...	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849889
<input type="checkbox"/>	1q	<empty>	[]	<empty>	<empty>	0

Conclusion

The Homemade Pickles and Snacks platform has been meticulously crafted to deliver a seamless and delightful experience for food enthusiasts seeking authentic, handcrafted flavors. By leveraging modern web technologies such as Flask for backend logic, secure user authentication, and dynamic cart management, the platform ensures a user-friendly interface for browsing, customizing, and ordering artisanal pickles and snacks.

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.

By combining time-honored recipes with modern e-commerce capabilities, this website not only preserves culinary heritage but also adapts to the digital age, ensuring that every jar of pickle or snack reaches customers with the same care and quality as a homemade meal. As the platform evolves, it stands ready to scale, introduce new product lines, and foster a community of food lovers united by a passion for authentic flavors.

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