

# 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

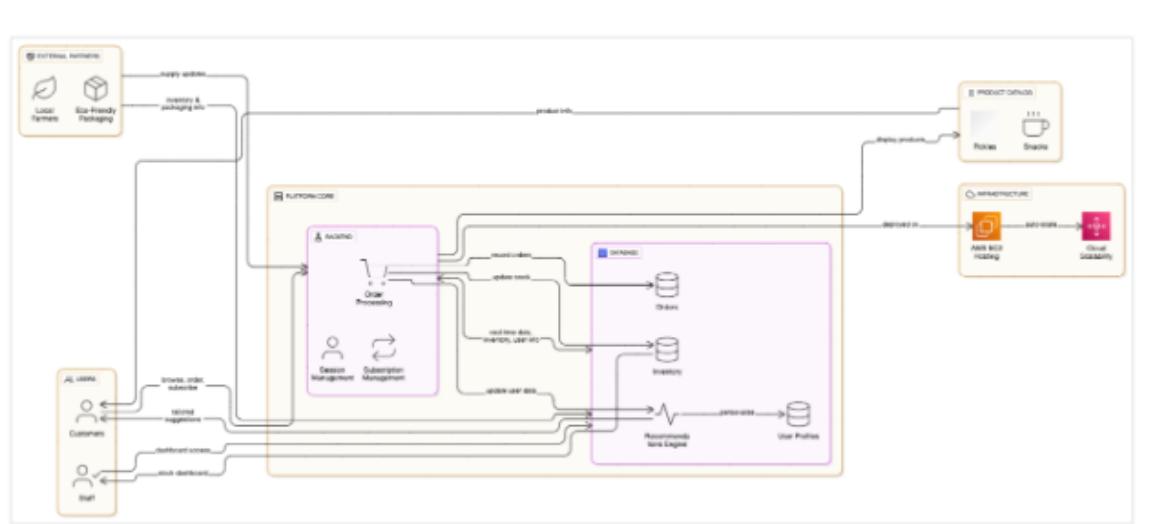
and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and fulfillment progress, ensuring timely restocking and minimizing overselling risks.

### 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.

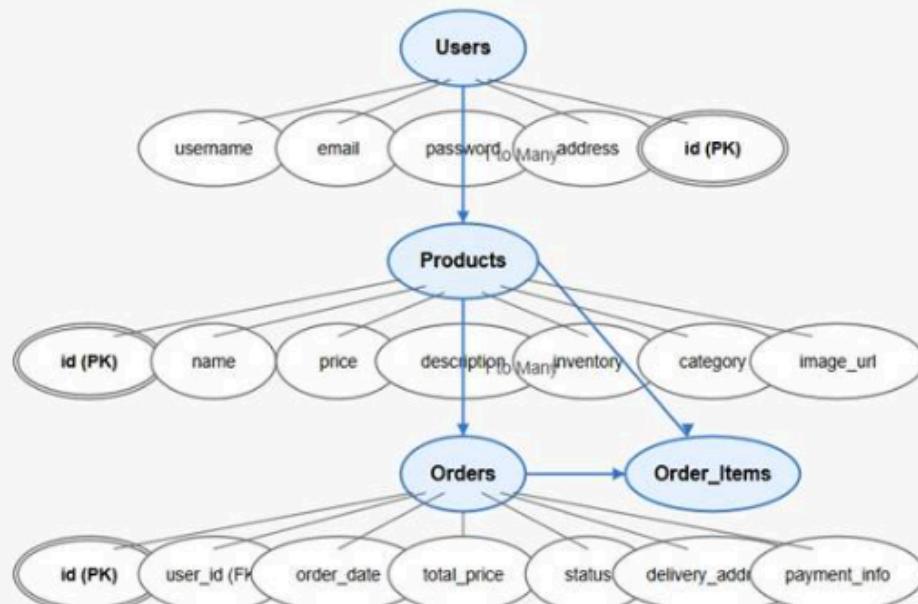
## Architecture

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..



## Entity Relationship (ER) Diagram

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



## Pre-requisites

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

## Project WorkFlow

## **Milestone 1. Backend Development and Application Setup**

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

## **Milestone 2. AWS Account Setup and Login**

- Set up an AWS account if not already done.
- Log in to the AWS Management Console

## **Milestone 3. DynamoDB Database Creation and Setup**

- Create a DynamoDB Table.
- Configure Attributes for User Data and 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.

### 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.

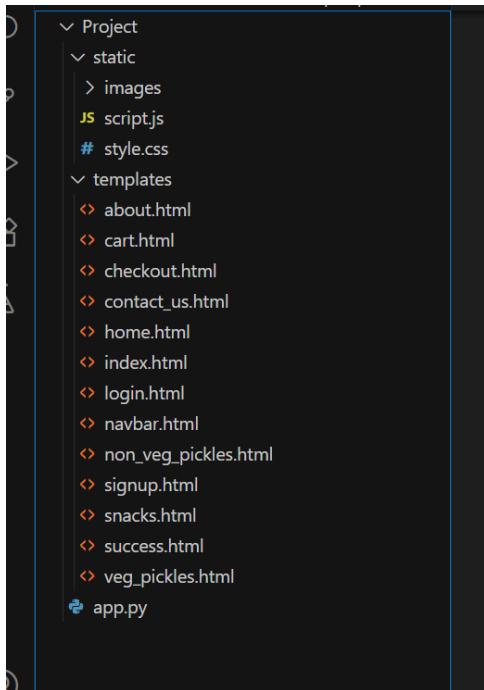
# **Milestone 1 : Web Application Development and**

# 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.

## LOCAL DEPLOYMENT

- File Explorer Structure



Description of the code :

? Flask App Initialization

```
Project > app.py > ...
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  import boto3
3  import uuid
4  import smtplib
5  from email.mime.text import MIMEText
6  from email.mime.multipart import MIMEMultipart
7  import os
8  import bcrypt
9
10
11 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.

```
13
14 # AWS Configuration
15 dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
16 user_table = dynamodb.Table('Users')
17 orders_table = dynamodb.Table('Orders')
18
19
```

- Routes for Web Pages

```
19
20 # Email Configuration
21 EMAIL_ADDRESS = 'susmithaghantasala7@gmail.com'
22 EMAIL_PASSWORD = 'hvop ohwp vtku ihuu'
23
24
25
26 @app.route('/')
27 def index():
28     return render_template('index.html')
29
30 @app.route('/home')
31 def home():
32     return render_template('index.html')
33
34 @app.route('/veg-pickles')
35 def veg_pickles():
36     return render_template('veg_pickles.html')
37
38 @app.route('/non-veg-pickles')
39 def non_veg_pickles():
40     return render_template('non_veg_pickles.html')
41
42 @app.route('/snacks')
43 def snacks():
44     return render_template('snacks.html')
45
46 @app.route('/cart')
```

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

```

55
56     @app.route('/success', methods=['GET', 'POST'])
57     def success():
58         return render_template('success.html')
59
60     @app.route('/login', methods=['GET', 'POST'])
61     def login():
62         if request.method == 'POST':
63             username = request.form['username']
64             password = request.form['password'].encode('utf-8')
65
66             # Get user from DynamoDB
67             response = user_table.get_item(Key={'username': username})
68             user = response.get('Item')
69
70             if user:
71                 hashed_password = user['password'].value if hasattr(user['password'], 'value') else user[
72                     'password'].decode('utf-8')
73                 if bcrypt.checkpw(password, hashed_password.encode('utf-8')):
74                     session['username'] = username
75                     flash('Login successful!', 'success')
76                     return redirect(url_for('home'))
77                 else:
78                     flash('Invalid password', 'danger')
79             else:
80                 flash('User not found', 'danger')
81
82     return render_template('login.html')

```

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

```

82
83
84     @app.route('/signup', methods=['GET', 'POST'])
85     def signup():
86         if request.method == 'POST':
87             username = request.form['username']
88             email = request.form['email']
89             password = request.form['password'].encode('utf-8')
90
91             # Check if user already exists
92             response = user_table.get_item(Key={'username': username})
93             if 'Item' in response:
94                 flash('Username already exists!', 'danger')
95                 return render_template('signup.html')
96
97             # Hash the password
98             hashed = bcrypt.hashpw(password, bcrypt.gensalt())
99
100            # Save user to DynamoDB
101            user_table.put_item(
102                Item={
103                    'username': username,
104                    'email': email,
105                    'password': hashed.decode('utf-8') # Store as string
106                }
107            )
108
109            # Send welcome email
110            send_email(email, 'Welcome to Home made Pickles!', 'Thank you for signing up!')

```

```

118
119     @app.route('/about')
120     def about():
121         return render_template('about.html')
122
123     @app.route('/contact_us', methods=['GET', 'POST'])
124     def contact_us():
125         if request.method == 'POST':
126             name = request.form['name']
127             email = request.form['email']
128             message_content = request.form['message']
129
130             # Combine the message
131             body = f"Name: {name}\nEmail: {email}\nMessage:\n{message_content}"
132
133             # Send email to yourself
134             send_email(EMAIL_ADDRESS, 'New Contact Us Message', body)
135
136             flash('Your message has been sent! Thank you for contacting us.', 'success')
137             return redirect(url_for('contact_us'))
138
139         return render_template('contact_us.html')
140
141
142     # Email Sending Function
143     def send_email(to_email, subject, body):
144         try:
145             # Email sending function
146             def send_email(to_email, subject, body):
147                 try:
148                     msg = MIMEText(body)
149                     msg['From'] = EMAIL_ADDRESS
150                     msg['To'] = to_email
151                     msg['Subject'] = subject
152                     msg.attach(MIMEText(body, 'plain'))
153
154                     server = smtplib.SMTP('smtp.gmail.com', 587)
155                     server.starttls()
156                     server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
157                     server.send_message(msg)
158                     server.quit()
159                 except Exception as e:
160                     print(f"Failed to send email: {e}")
161
162             if __name__ == '__main__':
163                 app.run(debug=True, host='0.0.0.0', port=5000)

```

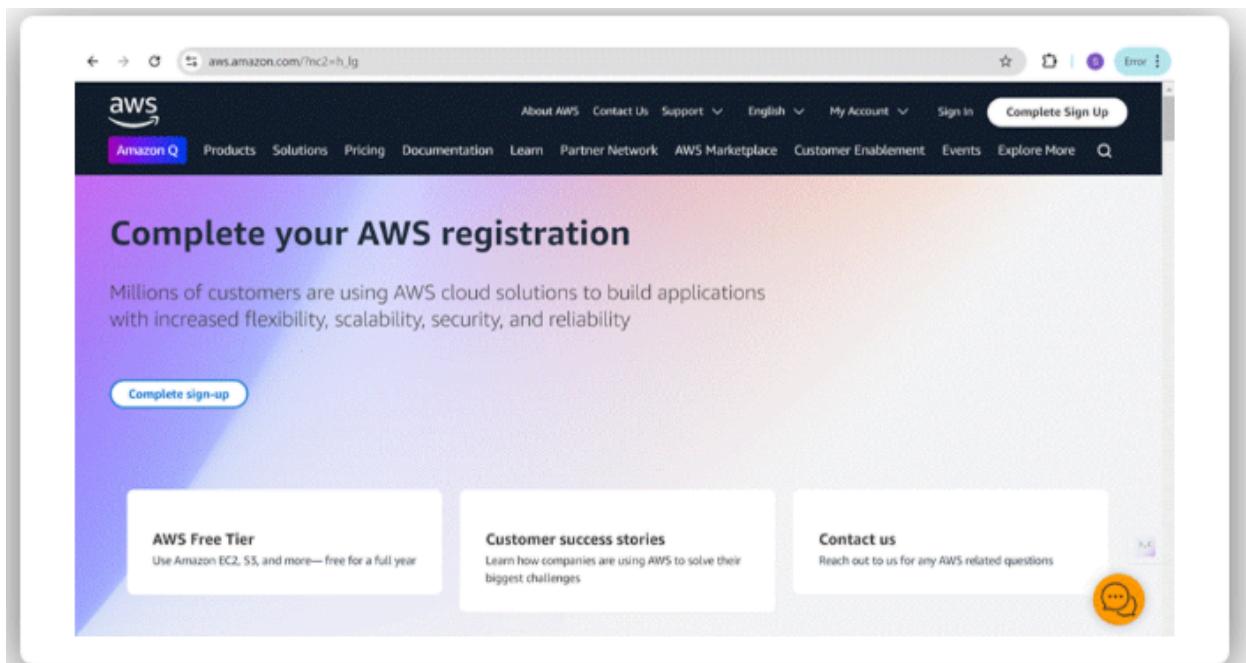
## Milestone 2 : AWS Account Setup

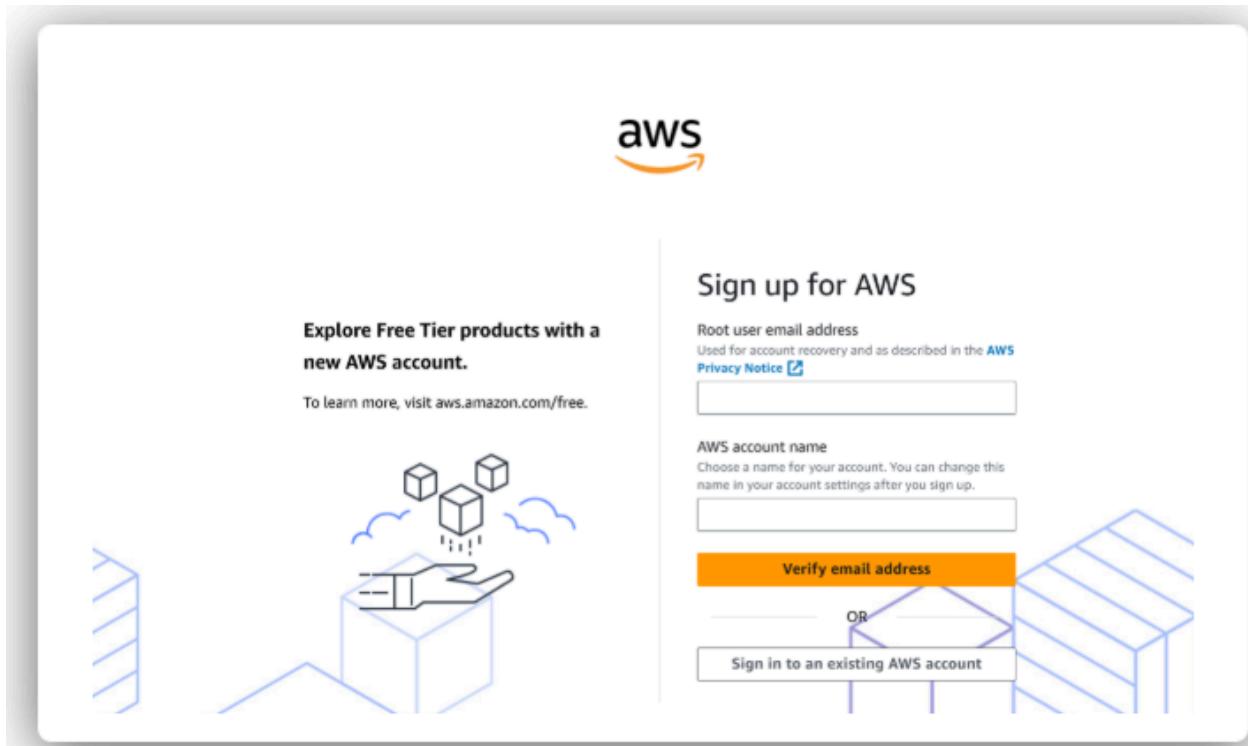
Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

# 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.





## 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.

The screenshot shows the AWS IAM console. A search bar at the top left contains the text "dynamodb". On the left sidebar, under the "Services" section, "DynamoDB" is selected. The main pane displays the "EC2\_DynamoDB\_Role" role. The "Attached entities" section shows one entity: "77042471:role/EC2\_DynamoDB\_Role". Below this, there are "Simulate", "Remove", and "Add permissions" buttons. The status bar at the bottom right indicates the date as 03-07-2025.

The screenshot shows the AWS DynamoDB Dashboard. The left sidebar includes links for "Dashboard", "Tables", "Explore items", "PartiQL editor", "Backups", "Exports to S3", "Imports from S3", "Integrations", "Reserved capacity", and "Settings". Under the "DAX" section, there are links for "Clusters", "Subnet groups", "Parameter groups", and "Events". The main content area features sections for "Alarms (0)", "DAX clusters (0)", and a "Create resources" section. The "Create resources" section includes a "Create table" button and information about Amazon DynamoDB Accelerator (DAX). A "What's new" section at the bottom right mentions a recent update about AWS Cost Management.

The screenshot shows the AWS DynamoDB Tables page. The left sidebar includes links for "Dashboard" and "Tables". The main content area displays a table titled "Tables (0)". The columns are: Name, Status, Partition key, Sort key, Indices, Deletion protection, Read capacity mode, Write capacity mode, and Total size. A message at the bottom of the table area states: "You have no tables in this account in this AWS Region." There is a "Create table" button at the bottom of the table.

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings, DAX Clusters, Subnet groups, Parameter groups, and Events. The main area is titled 'Tables (2) Info' and lists two tables: 'Orders' and 'Users'. The 'Orders' table has a partition key 'order\_id' and a sort key '-'. It has 0 indexes, 0 replication regions, and deletion protection off. The 'Users' table has a partition key 'username' and a sort key '-'. It also has 0 indexes, 0 replication regions, and deletion protection off. Both tables are marked as 'Active'. At the top, there are notifications for 'Share your feedback on Amazon DynamoDB' and 'The Orders table was created successfully.' A 'Create table' button is visible at the top right.

## 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.

## Attach Policies

Attach the following policies to the role:

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

Screenshot of the AWS IAM 'Create role' wizard - Step 1: Select trusted entities.

**Role details**

**Role name:** EC2\_DynamoDB\_Role

**Description:** Allows EC2 instances to call AWS services on your behalf.

**Step 1: Select trusted entities**

**Trust policy:**

```
1 < {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "sts:AssumeRole"
8             ],
9             "Principal": [
10                "arn:aws:iam::123456789012:root"
11            ]
12        }
13    ]
14}
```

CloudShell Feedback 31°C Mostly cloudy ENG IN 20:38 04-07-2025

Screenshot of the AWS IAM 'Create role' wizard - Step 2: Add permissions.

**Permissions policy summary**

Policy name	Type	Attached as
AmazonDynamoDBFullAccess	AWS managed	Permissions policy
AmazonSNSFullAccess	AWS managed	Permissions policy

**Step 3: Add tags**

**Add tags - optional**

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

Cancel Previous Create role

CloudShell Feedback 31°C Mostly cloudy ENG IN 20:38 04-07-2025

The screenshot shows the AWS IAM Roles page. A green banner at the top indicates that a role named 'EC2\_DynamoDB\_Role' has been created. Below this, a table lists 12 existing roles, each with a checkbox, role name, trusted entities, and last activity date. The roles listed are: AWSServiceRoleForAmazonEKS, AWSServiceRoleForAmazonEKSNodegroup, AWSServiceRoleForAPIGateway, AWSServiceRoleForAutoScaling, AWSServiceRoleForECS, AWSServiceRoleForOrganizations, AWSServiceRoleForSSO, AWSServiceRoleForSupport, AWSServiceRoleForTrustedAdvisor, and two partially visible roles starting with 'AWSLambda...' and 'EC2...'. The left sidebar shows navigation options like Dashboard, Access management, and Access reports.

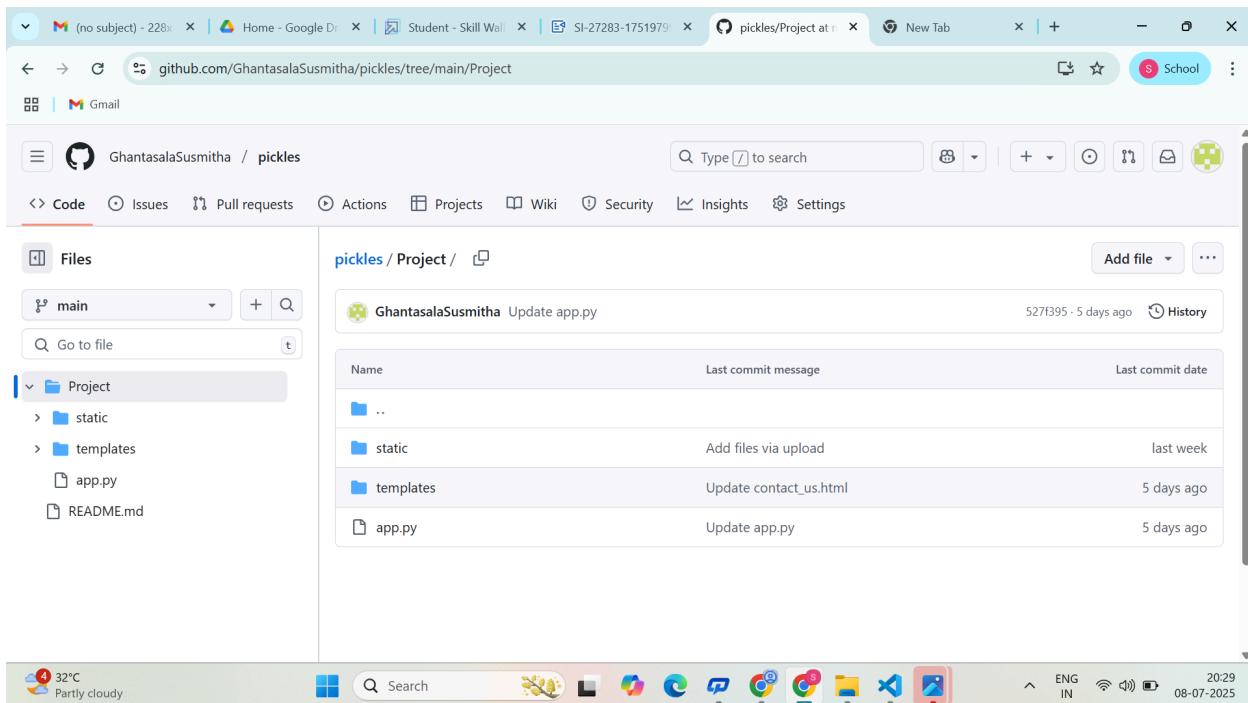
Role name	Trusted entities	Last activity
AWSServiceRoleForAmazonEKS	AWS Service: eks (Service-Linked Role)	140 days ago
AWSServiceRoleForAmazonEKSNodegroup	AWS Service: eks-nodegroup (Service-Linked Role)	141 days ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked Role)	-
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Linked Role)	136 days ago
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	136 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	212 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
...	...	...
...	...	...

## 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.

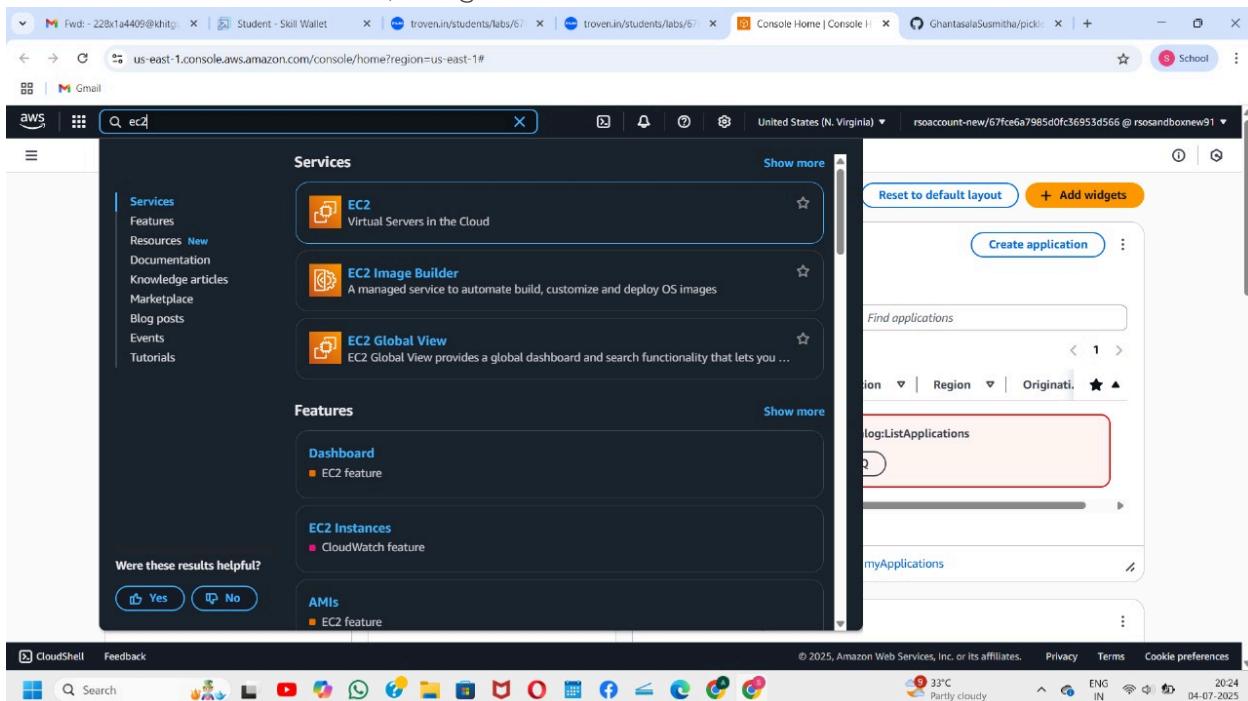
## Load your Project Files to GitHub

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

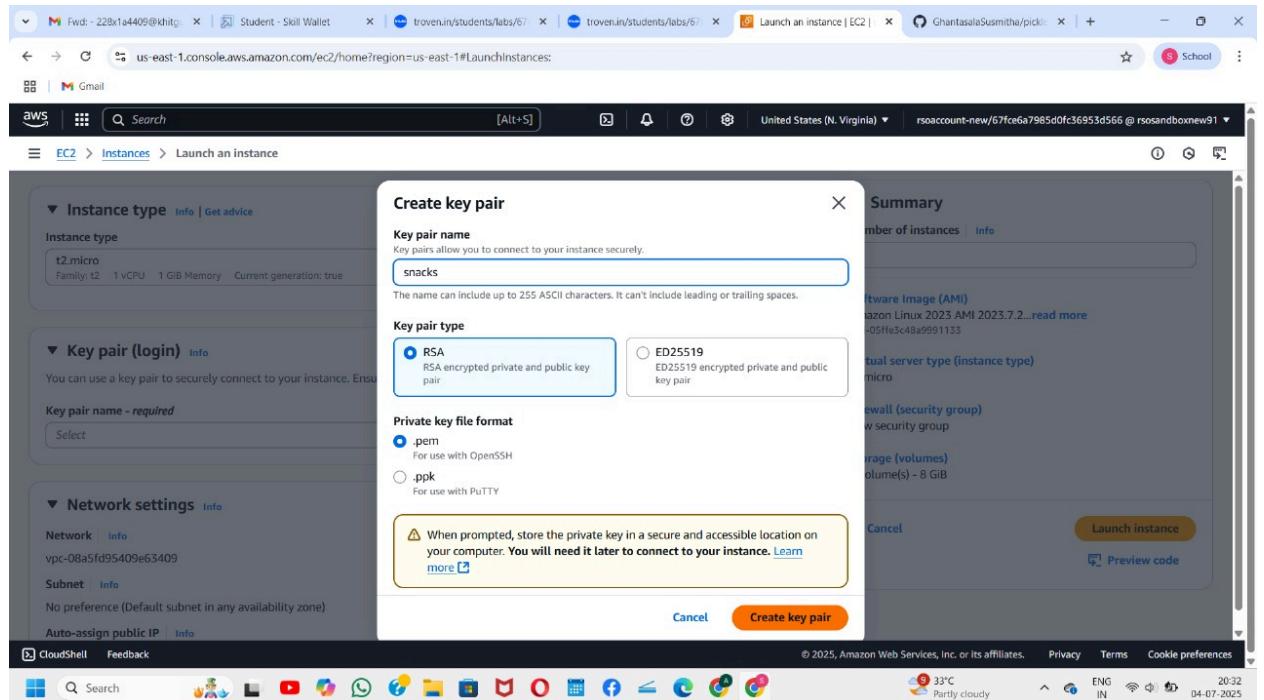


## Launch an EC2 instance

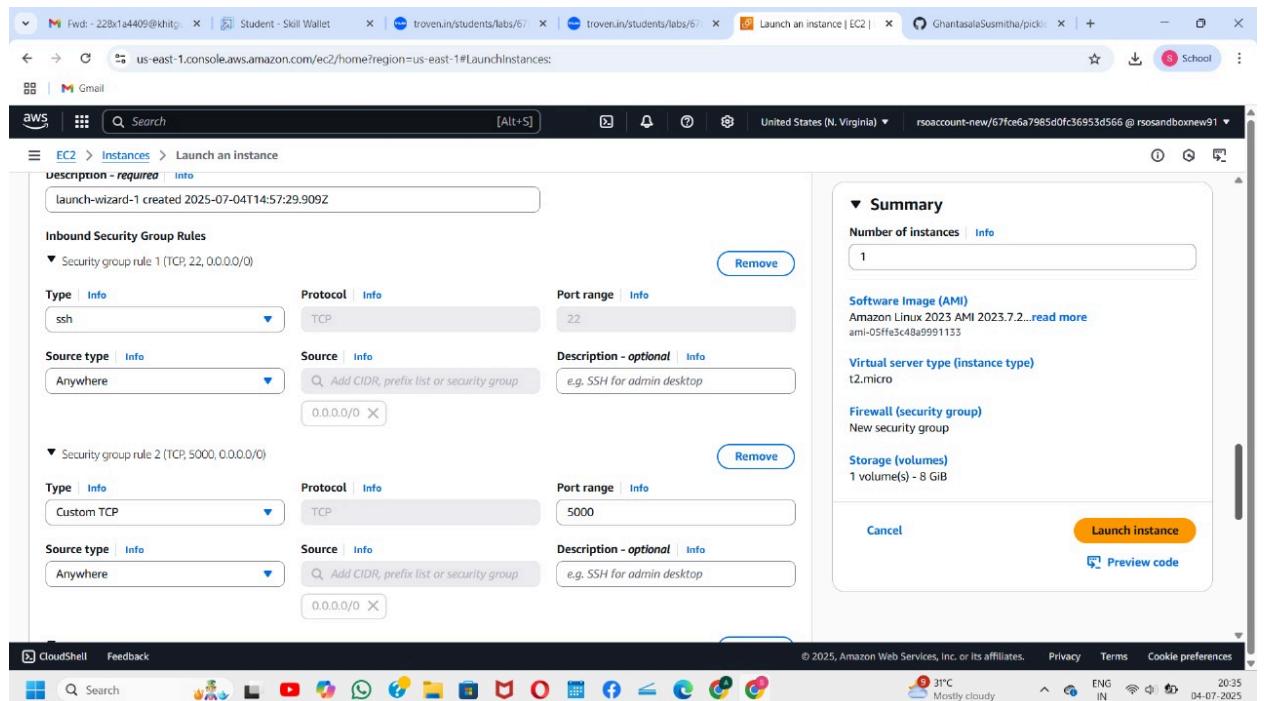
- In the AWS Console, navigate to EC2 and launch a new instance.



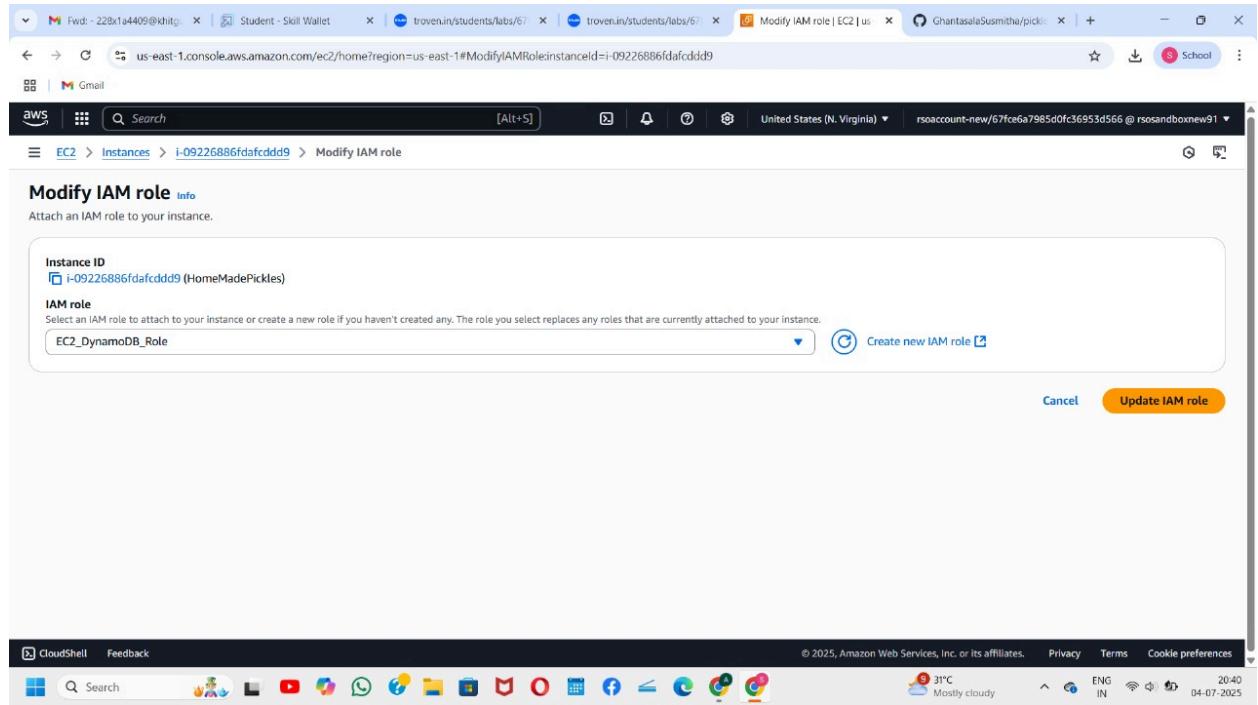
- Click on Launch instance to launch EC2 instance



## 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.



- Now connect the EC2 with the files

```

Power release of "Amazon Linux" is available.
Version 2023.6.20241010:
"/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
t-user@ip-172-31-3-5 ~ $ 

```

# 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/GhantasalaSusmitha/pickles.git](https://github.com/GhantasalaSusmitha/pickles.git)

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

```

  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
    ━━━━━━━━━━━━━━━━ 224 kB 21.7 MB/s
Collecting itsdangerous==2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting importlib-metadata==3.6.0
  Downloading importlib-metadata-3.6.0-py3-none-any.whl (27 kB)
Collecting zip==3.20
  Downloading zip-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: werkzeug, itsdangerous, importlib-metadata, click, blinker, Flask
Successfully installed Flask-3.1.1 blinker-1.9.0 click-8.1.8 importlib-metadata-3.6.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3.23.0
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
    ━━━━━━━━━━━━━━ 139 kB 7.8 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 kB)
    ━━━━━━━━━━━━ 13.8 kB 26.3 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    ━━━━━━━━━━━━ 85 kB 6.9 MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (1.25.10)
Requirement already satisfied: six>1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.3->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.3 botocore-1.39.3 s3transfer-0.13.0
[ec2-user@ip-172-31-21-248 ~]$ git clone https://github.com/GhantasalaSusmitha/pickles.git

```

i-09226886fdafddd9 (HomeMadePickles)  
PublicIPs: 54.165.233.195 PrivateIPs: 172.31.21.248

```

Resolving deltas: 100% (11/11), done.
[ec2-user@ip-172-31-21-248 ~]$ cd pickles
[ec2-user@ip-172-31-21-248 pickles]$ cd Project
[ec2-user@ip-172-31-21-248 Project]$ python3 app.py
Traceback (most recent call last):
  File "/home/ec2-user/pickles/Project/app.py", line 8, in <module>
    import bcrypt
ModuleNotFoundError: No module named 'bcrypt'
[ec2-user@ip-172-31-21-248 Project]$ pip3 install bcrypt
Defaulting to user installation because normal site-packages is not writeable
Collecting bcrypt
  Downloading bcrypt-4.3.0-cp39-manylinux_2_34_x86_64.whl (284 kB)
    ━━━━━━━━━━━━ 284 kB 15.1 MB/s
Installing collected packages: bcrypt
Successfully installed bcrypt-4.3.0
[ec2-user@ip-172-31-21-248 Project]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
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://172.31.21.248:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-379-551

```

i-09226886fdafddd9 (HomeMadePickles)  
PublicIPs: 54.165.233.195 PrivateIPs: 172.31.21.248

Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```

Resolving deltas: 100% (11/11), done.
[ec2-user@ip-172-31-21-248 ~]$ cd pickles
[ec2-user@ip-172-31-21-248 pickles]$ cd Project
[ec2-user@ip-172-31-21-248 Project]$ python3 app.py
Traceback (most recent call last):
  File "/home/ec2-user/pickles/Project/app.py", line 8, in <module>
    import bcrypt
ModuleNotFoundError: No module named 'bcrypt'
[ec2-user@ip-172-31-21-248 Project]$ pip3 install bcrypt
Defaulting to user installation because normal site-packages is not writeable
Collecting bcrypt
  Downloading bcrypt-4.3.0-cp39-manylinux_2_34_x86_64.whl (284 kB)
    ━━━━━━━━━━━━ 284 kB 15.1 MB/s
Installing collected packages: bcrypt
Successfully installed bcrypt-4.3.0
[ec2-user@ip-172-31-21-248 Project]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
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://172.31.21.248:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-379-551

```

i-09226886fdafddd9 (HomeMadePickles)  
PublicIPs: 54.165.233.195 PrivateIPs: 172.31.21.248

```

Resolving deltas: 100% (11/11), done.
[ec2-user@ip-172-31-21-248 ~]$ cd pickles
[ec2-user@ip-172-31-21-248 pickles]$ cd Project
[ec2-user@ip-172-31-21-248 Project]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
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://172.31.21.248:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-379-551

```

i-09226886fdafddd9 (HomeMadePickles)  
PublicIPs: 54.165.233.195 PrivateIPs: 172.31.21.248

Access the website through:

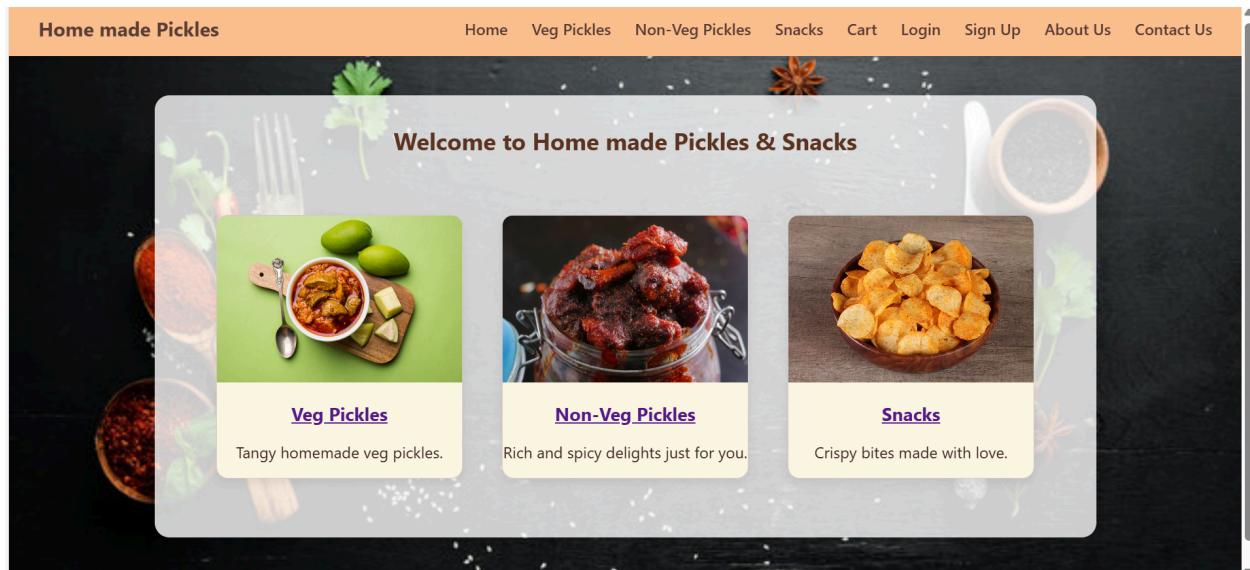
PublicIPs: <http://54.165.233.195:5000/>

## Milestone 7 : Testing and Deployment

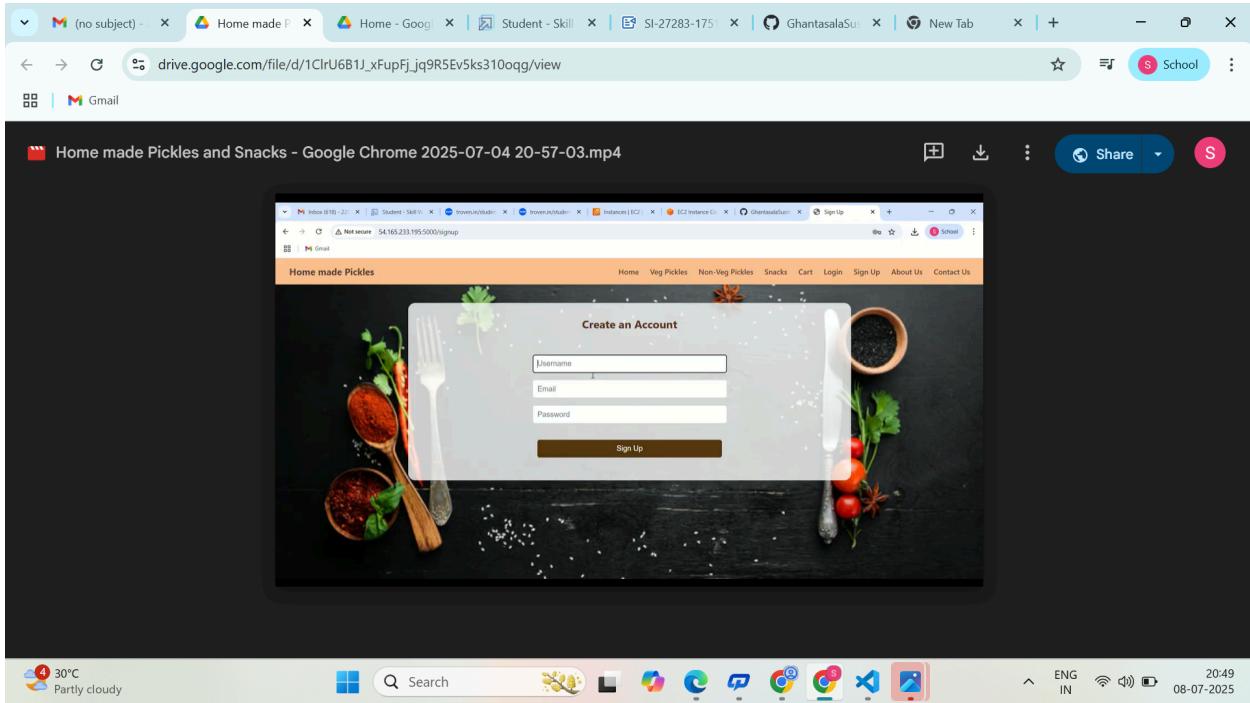
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

## Functional testing to verify the Project

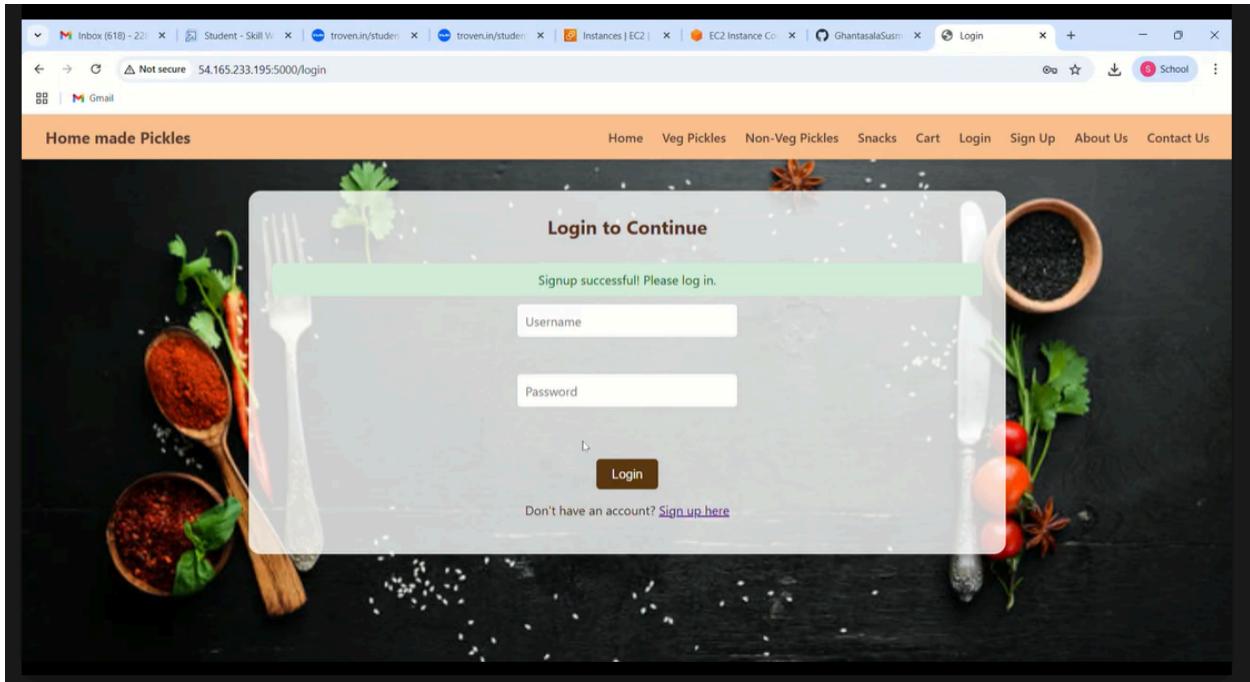
Welcome page:



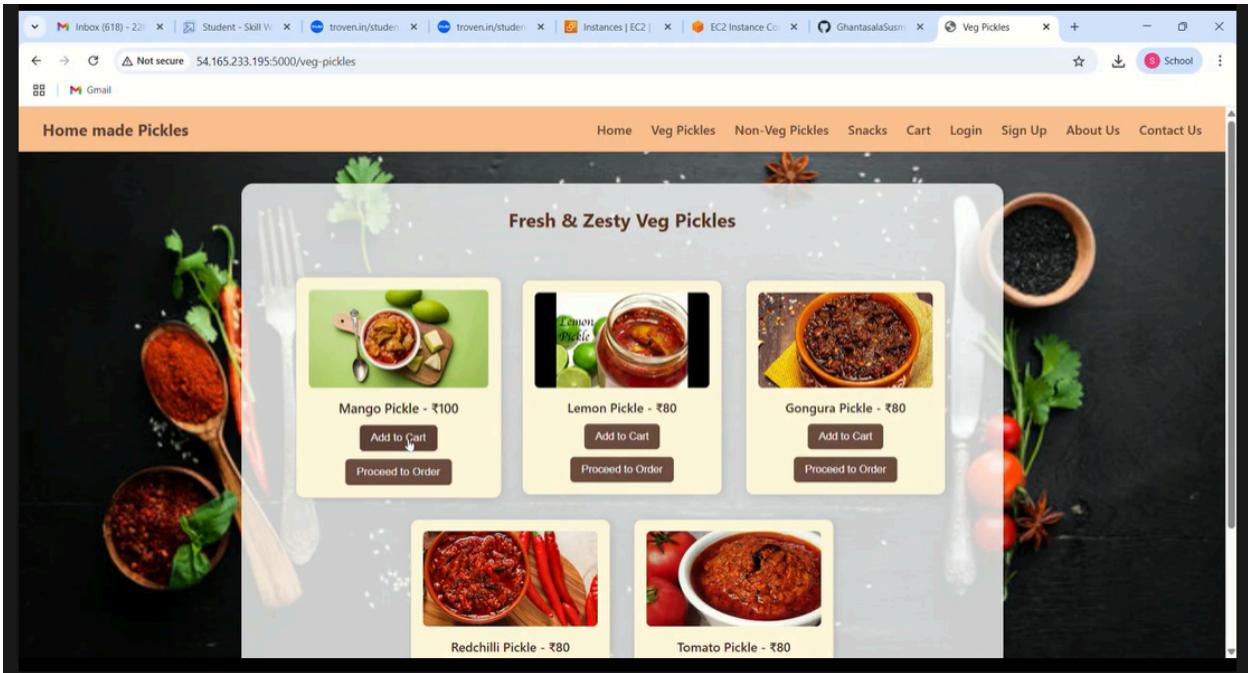
## Sign up page:



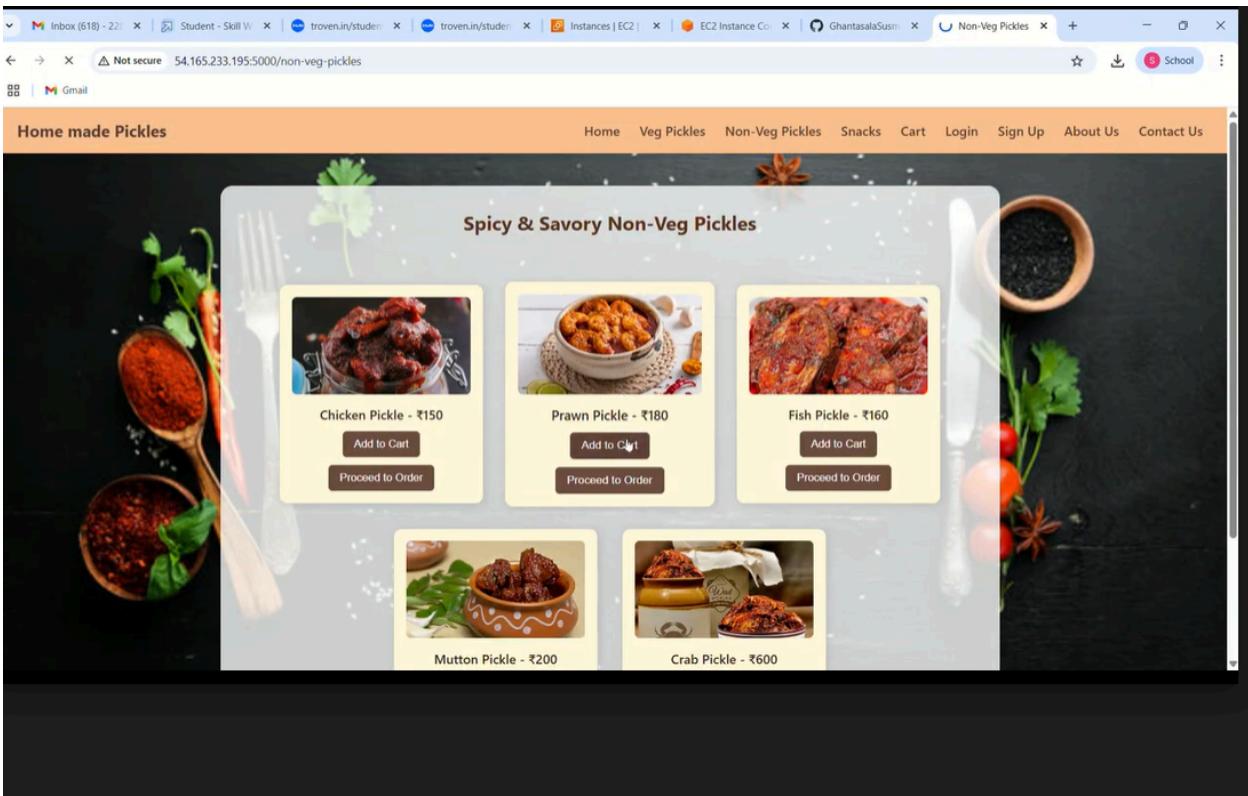
## Login page:



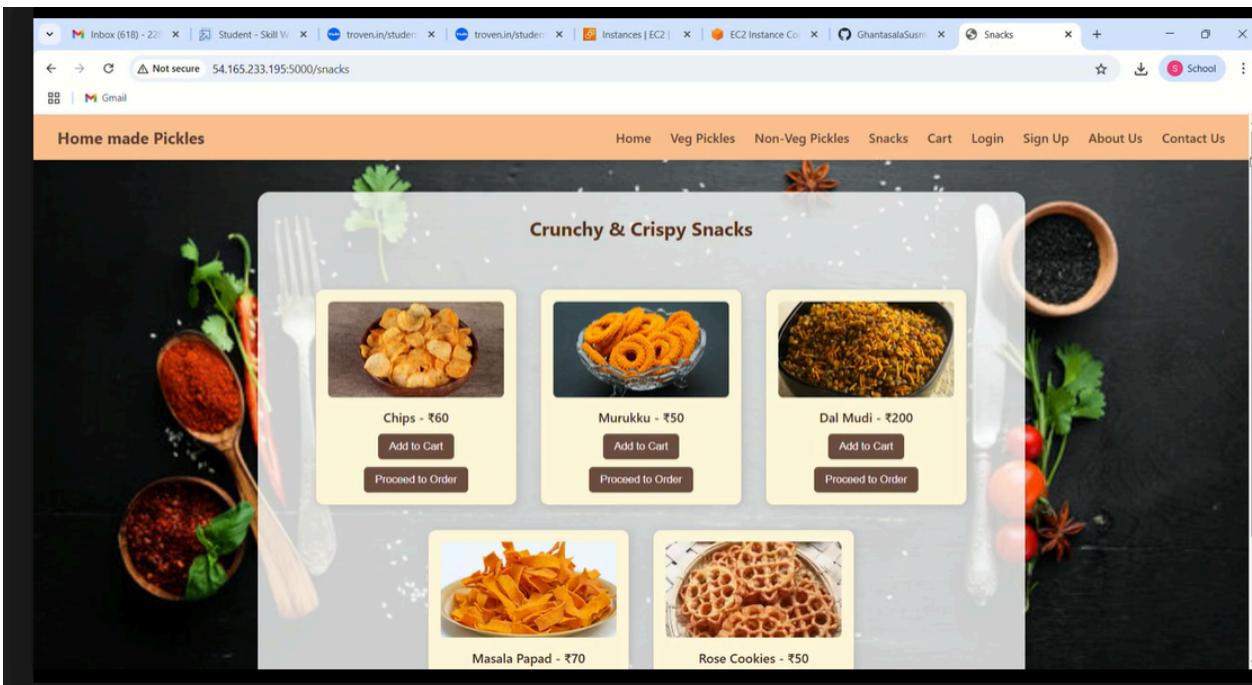
## Veg Pickles:



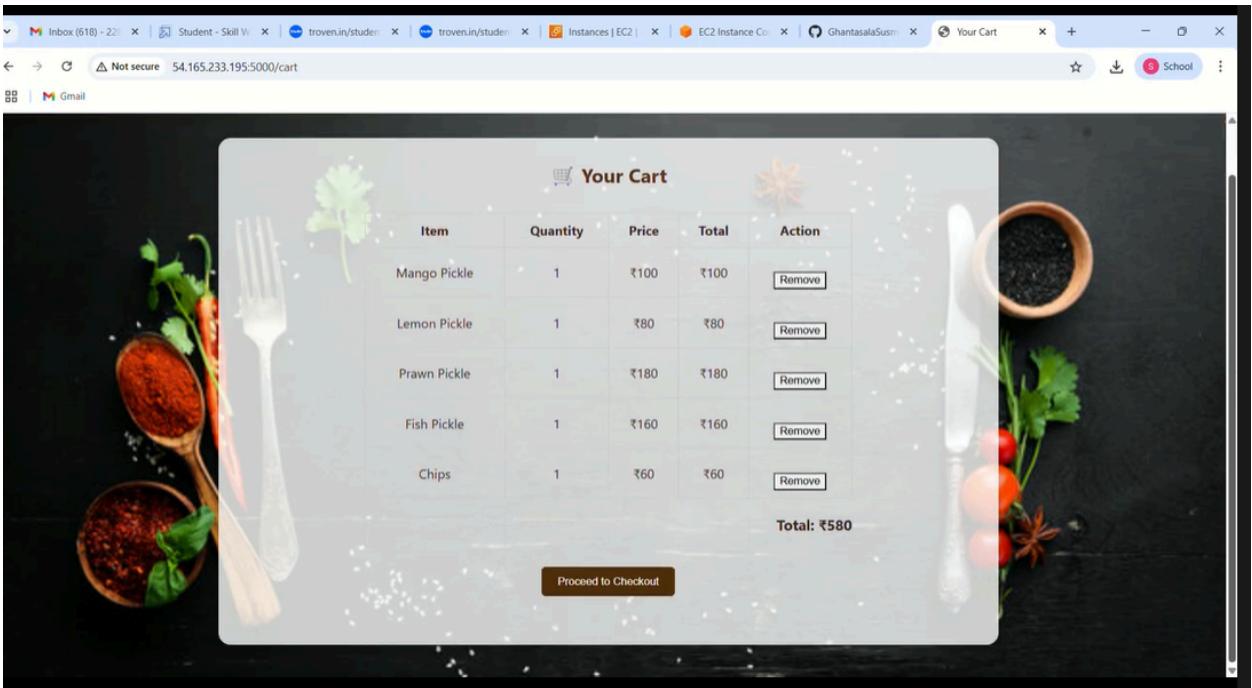
## Non-veg pickles:



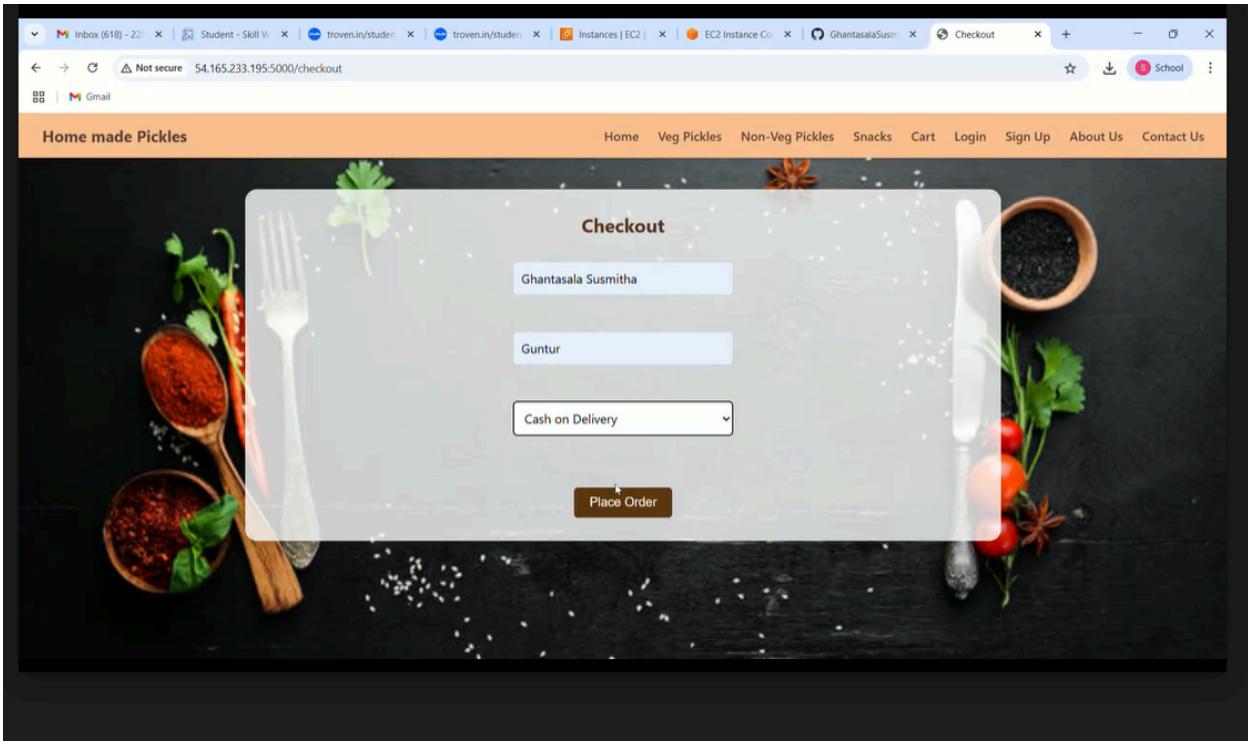
## Snacks:



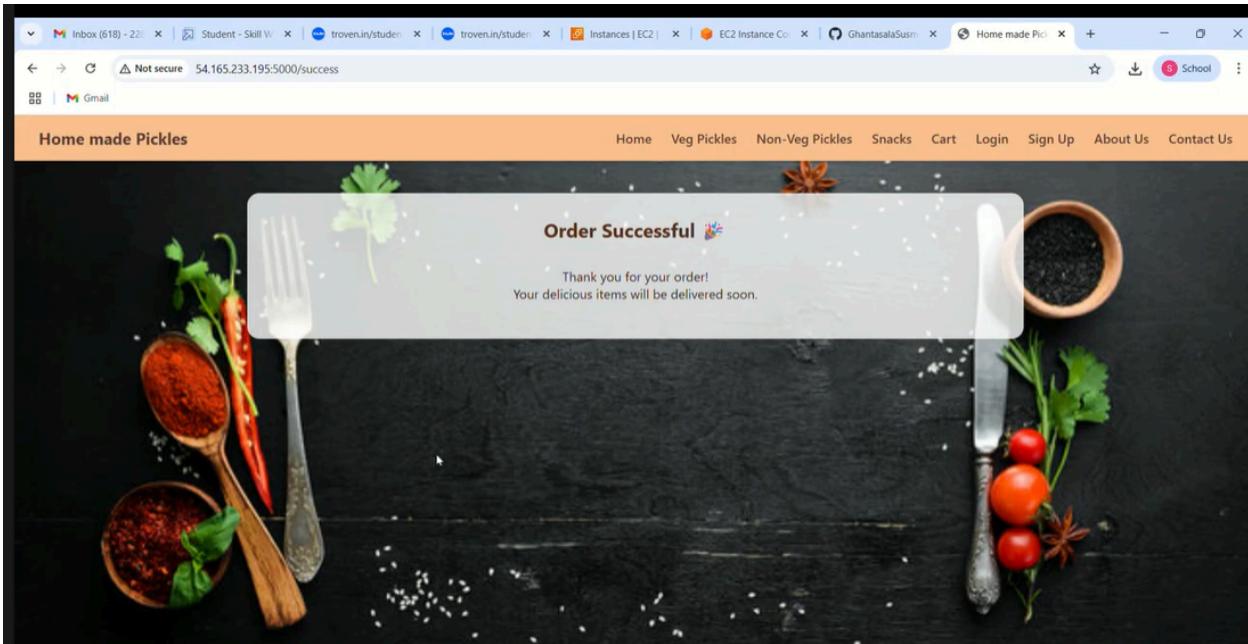
## Cart page:



## Checkout Page:



## Success Page:



Dynamodb Database updatons :

1. Users table :

Completed. Read capacity units consumed: 2

Items returned (3)

Actions Create item

	username (String)	email	password
<input type="checkbox"/>	<a href="#">Shiva</a>	kilarukusu...	scrypt:32768:8:1\$W5tA59Z7nQjLXbtx\$d6befef2b3e14bbe9d3d3e3f1c...
<input type="checkbox"/>	<a href="#">kusuma</a>	<empty>	<empty>
<input type="checkbox"/>	<a href="#">Alekhya</a>	alekhya@g...	scrypt:32768:8:1\$EwCDTl0iaGcKutw3\$cd5dbf5c12ec17cb518f7c15cd...

2. Orders table :

► Filters

Run Reset

Completed. Read capacity units consumed: 2

Items returned (4)

Actions Create item

	order_id (String)	address	items	name	payment_met...	phone
<input type="checkbox"/>	<a href="#">7c6bd84e-f2c7-4fe0...</a>	Kothur	[{"M": {"n..."}]	Siri	cod	8187810...
<input type="checkbox"/>	<a href="#">3de0fe0c-9539-4fb6...</a>	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849881...
<input type="checkbox"/>	<a href="#">fbc41d6d-d6f2-4158-...</a>	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849881...
<input type="checkbox"/>	<a href="#">1g</a>	<empty>	[ ]	<empty>	<empty>	0

