HomeMade 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."

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

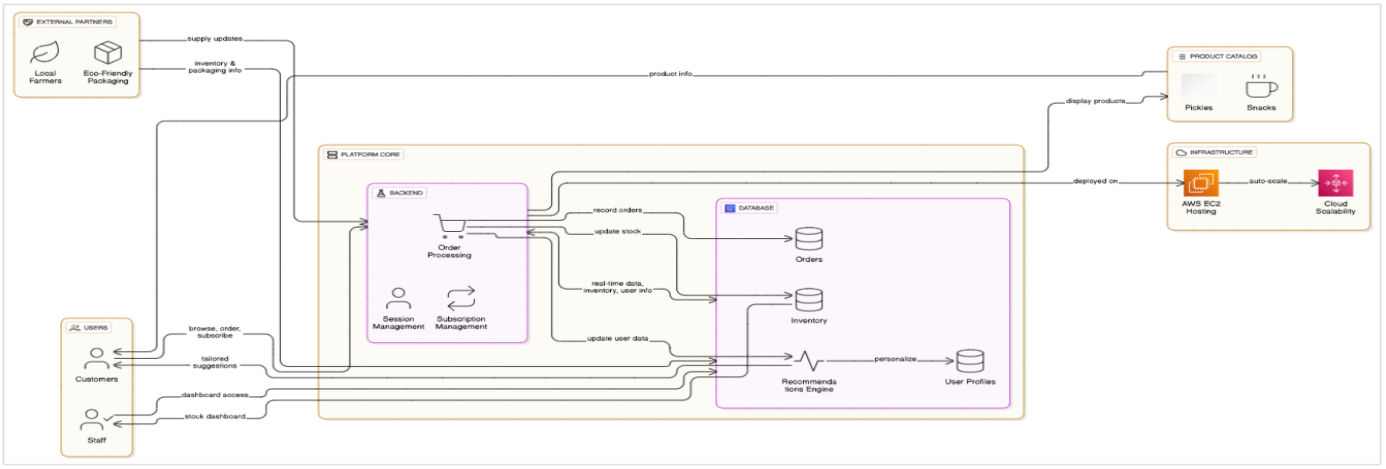
When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and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

Scenario 3: Personalized User Experience and Recommendations

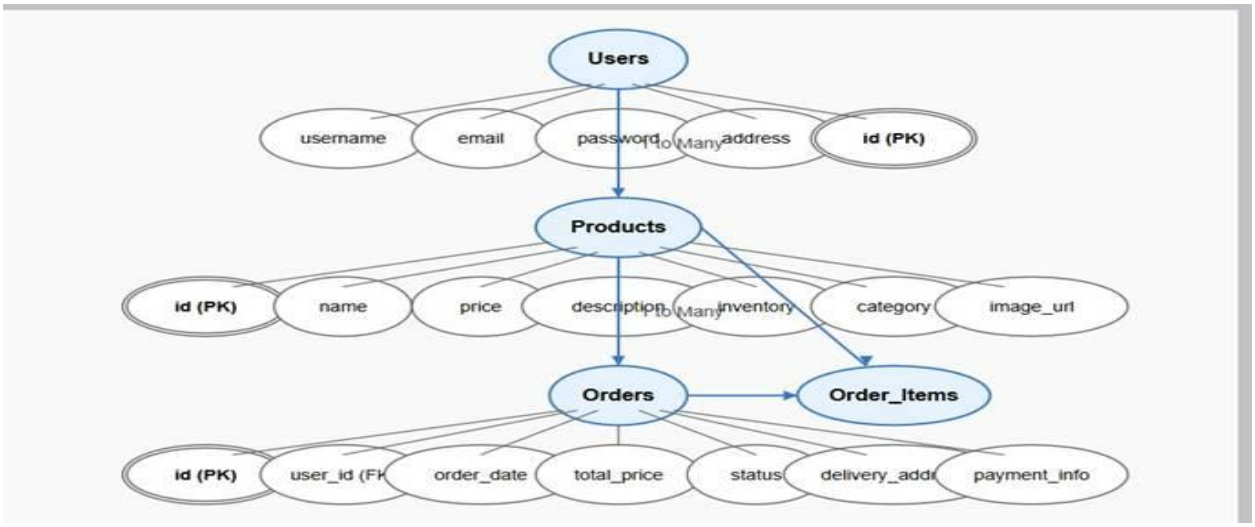
The platform leverages user behavior data to enhance engagement. A returning customer, for instance, views tailored recommendations based on past purchases and browsing history. The system dynamically adjusts suggestions in real-time, while maintaining fast response rates even during high traffic, creating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

AWS 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:



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

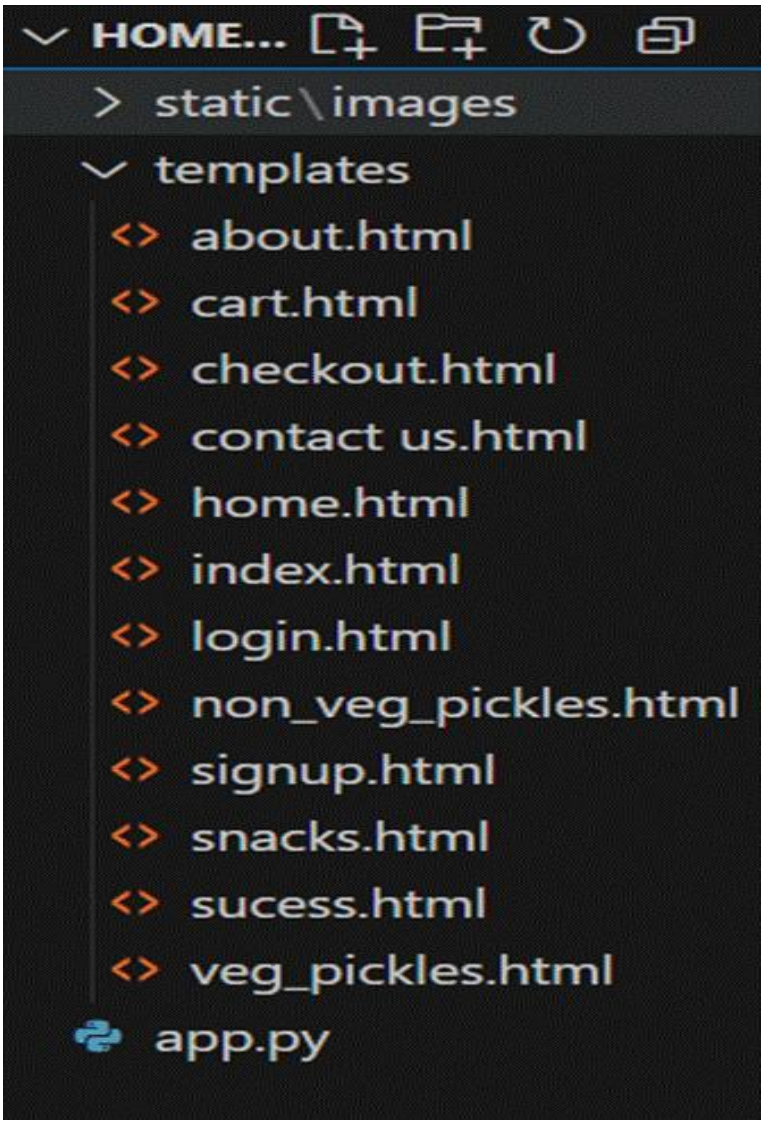
Milestone 8. Testing and Deployment

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

Milestone 1: Backend Development and Application SetUp

- **Activity 1.1:** Develop the Backend Using Flask

1. File Explorer Structure



Description: 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

Description of the code :

? Flask App Initialization

```python
from flask import Flask, render_template, request, redirect, url_for, flash, sessio
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import boto3
import smtplib
import logging
import uuid
from email.mime.text import MIMEText
import os
```

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

```python
dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
orders_table = dynamodb.Table('PickleOrders')
users_table = dynamodb.Table('users')
sns = boto3.client('sns', region_name=AWS_REGION)

#Email settings (loaded securely from .env)
EMAIL_HOST = os.getenv('EMAIL_HOST', 'smtp.gmail.com')
EMAIL_PORT = int(os.getenv('EMAIL_PORT', 587))
EMAIL_USER = os.getenv('EMAIL_USER')
EMAIL_PASSWORD = os.getenv('EMAIL_PASSWORD')


@app.context_processor
def inject_theme():
    return {"color": app.config["THEME_COLOR"], "year": datetime.now().year}

# ---------- Updated Product Inventory ----------
products = {
    "mango": {"name": "Mango Pickle", "price": 249, "stock": 10, "image": "mango.jpg"},
    "lemon": {"name": "Lemon Pickle", "price": 199, "stock": 8, "image": "lemon.jpg"},
    "gongura": {"name": "Gongura Pickle", "price": 229, "stock": 7, "image": "gongura.jpg"},
    "chicken": {"name": "Chicken Pickle", "price": 349, "stock": 12, "image": "chicken.jpg"},
    "fish": {"name": "Fish Pickle", "price": 329, "stock": 9, "image": "fish.jpg"},
    "prawns": {"name": "Prawns Pickle", "price": 399, "stock": 6, "image": "prawns.jpg"},
    "murukulu": {"name": "Murukulu", "price": 149, "stock": 20, "image": "murukulu.jpg"},
    "nippattu": {"name": "Nippattu", "price": 129, "stock": 15, "image": "nippattu.jpg"},
    "hot_maida_biscuit": {"name": "Hot Maida Biscuit", "price": 109, "stock": 25, "image": "hot_maida_biscuit.jpg"},
}

def get_products(prefix=None):
```

- 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":
        email = request.form["email"]
        pwd = request.form["password"]
        user = users.get(email)
        if user and check_password_hash(user["hash"], pwd):
            session["user"] = email
            flash("Logged in!", "success")
            return redirect(url_for("home"))
        flash("Invalid credentials", "danger")
    return render_template("login.html")

@app.route("/logout")
```

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

```
@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
        email = request.form["email"]
        pwd = request.form["password"]
        confirm = request.form["confirm"]
        if pwd != confirm:
            flash("Passwords don't match", "warning")
            return redirect(url_for("signup"))
        if email in users:
            flash("User exists, log in", "info")
            return redirect(url_for("login"))
        users[email] = {"hash": generate_password_hash(pwd)}
        flash("Signup complete — please log in", "success")
        return redirect(url_for("login"))
    return render_template("signup.html")
```

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

```python
# -------------- Routes --------------
@app.route("/")
def home():
    best = dict(list(get_products().items())[:6])
    return render_template("index.html", items=best)

@app.route("/veg")
def veg():
    return render_template("veg.html", items={
        k: v for k, v in products.items() if k in ["mango", "lemon", "gongura"]
    })

@app.route("/nonveg")
def nonveg():
    return render_template("nonveg.html", items={
        k: v for k, v in products.items() if k in ["chicken", "fish", "prawns"]
    })

@app.route("/snacks")
def snacks():
    return render_template("snacks.html", items={
        k: v for k, v in products.items() if k in ["murukulu", "nippattu", "hot_maida_biscuit"]
    })

@app.route("/cart")
def cart():
    items, total = {}, 0
    if session.get("cart"):
        for pid, qty in session["cart"].items():
            if pid in products:
                items[pid] = {**products[pid], "qty": qty}
                total += products[pid]["price"] * qty
    return render_template("cart.html", items=items, total=total)
```

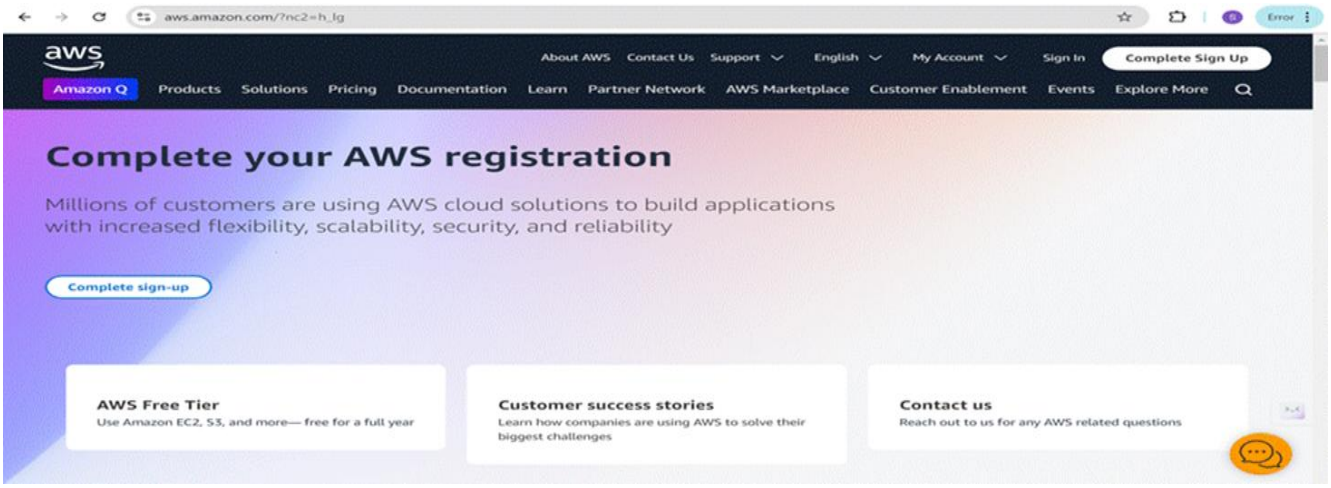
CheckOut

```python
@app.route("/checkout", methods=["GET", "POST"])
def checkout():
    cart = session.get("cart", {})
    if not cart:
        flash("Cart is empty", "warning")
        return redirect(url_for("home"))

    if request.method == "POST":
        name = request.form["name"]
        email = request.form["email"]
        address = request.form["address"]
        order_id = str(uuid.uuid4())
        order_time = datetime.now().isoformat()

        items = [
            {"product": pid, "price": products[pid]["price"], "quantity": qty}
            for pid, qty in cart.items()
        ]
        total = sum(p["price"] * p["quantity"] for p in items)
```

```python
    ]
    total = sum(p["price"] * p["quantity"] for p in items)

    order_data = {
        "order_id": order_id,
        "name": name,
        "email": email,
        "address": address,
        "order_time": order_time,
        "items": items,
        "total": total
    }

    save_order_to_dynamodb(order_data)

    summary = f"Order ID: {order_id}\nName: {name}\nTotal: ₹{total}\n\nThank you for your order!"
    send_order_email(email, summary)
    send_sns_notification(f"New order placed: {order_id}")

    session.pop("cart", None)
    flash("Order placed successfully!", "success")
    return redirect(url_for("success"))
return render_template("checkout.html")
```

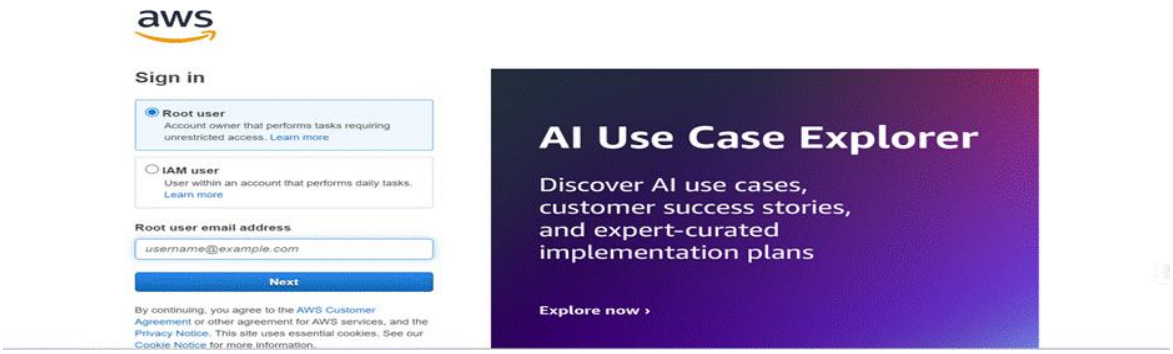
- **Milestone 2: AWS Account Setup and Login.**

    - **Activity 2.1:** Set up an AWS account if not already done.
        - Sign up for an AWS account and configure billing settings.



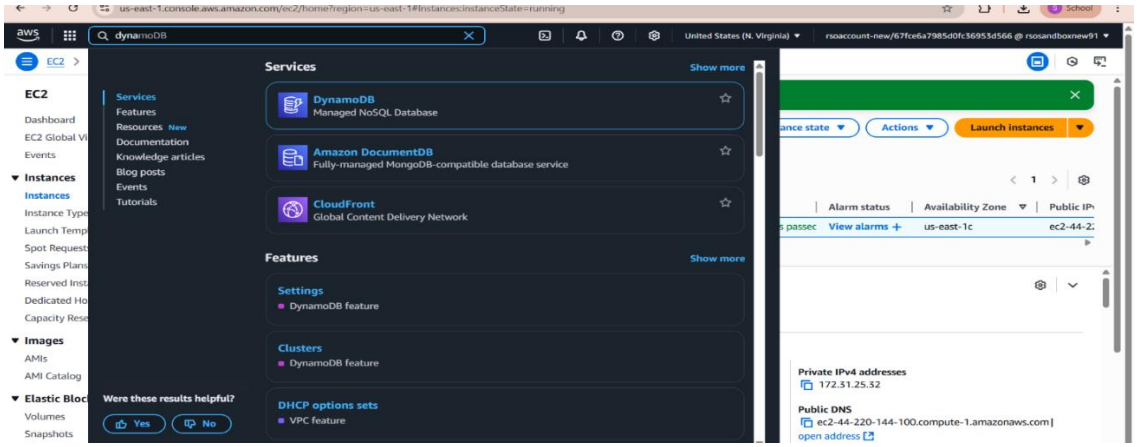
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit 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.

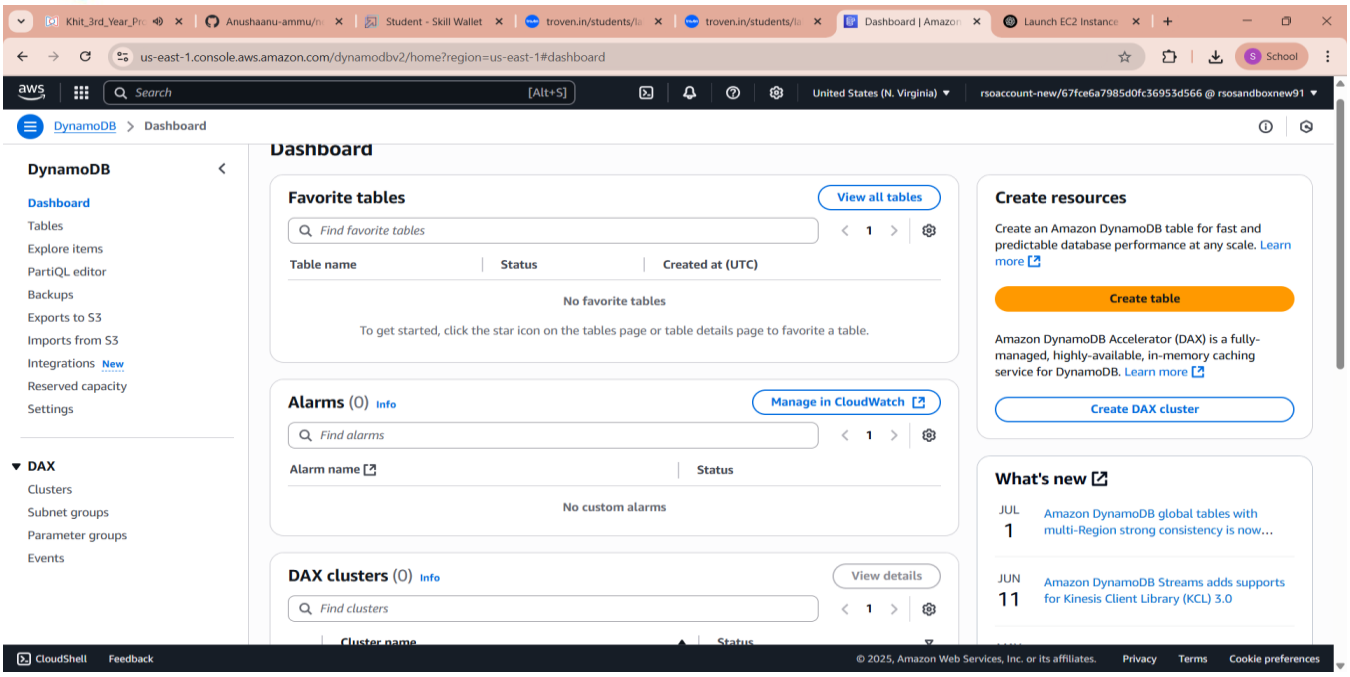
- **Activity2.2 :** Log in to the AWS Management Console
    - After setting up your account, log in to the [AWS Management Console](#).



Milestone 3: DynamoDB Database Creation and Setup

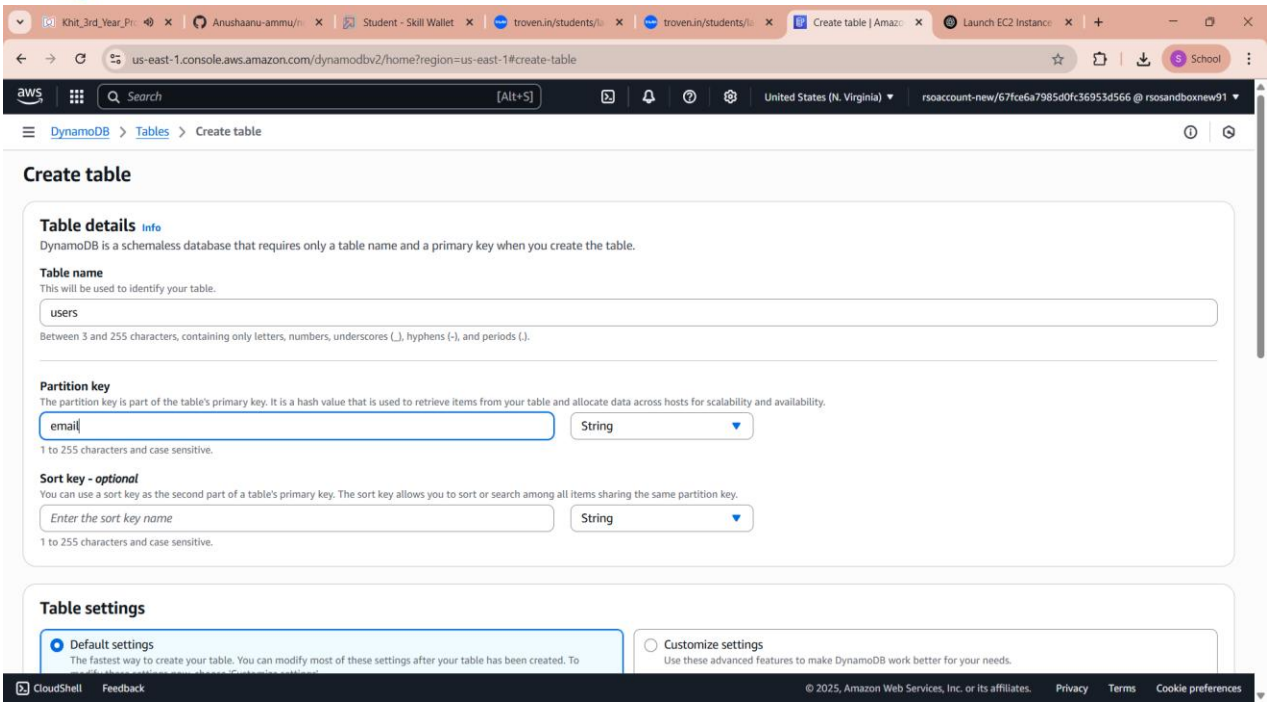
- **Activity 3.1:**Navigate to the DynamoDB
    - In the AWS Console, navigate to DynamoDB and click on create tables**.**

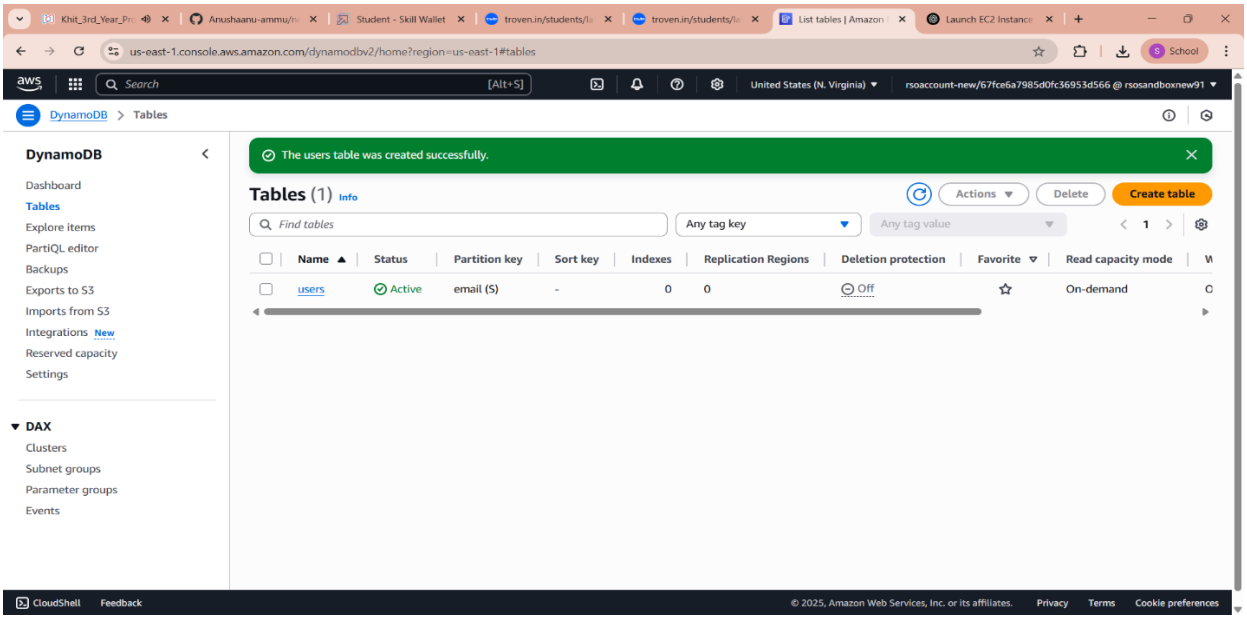




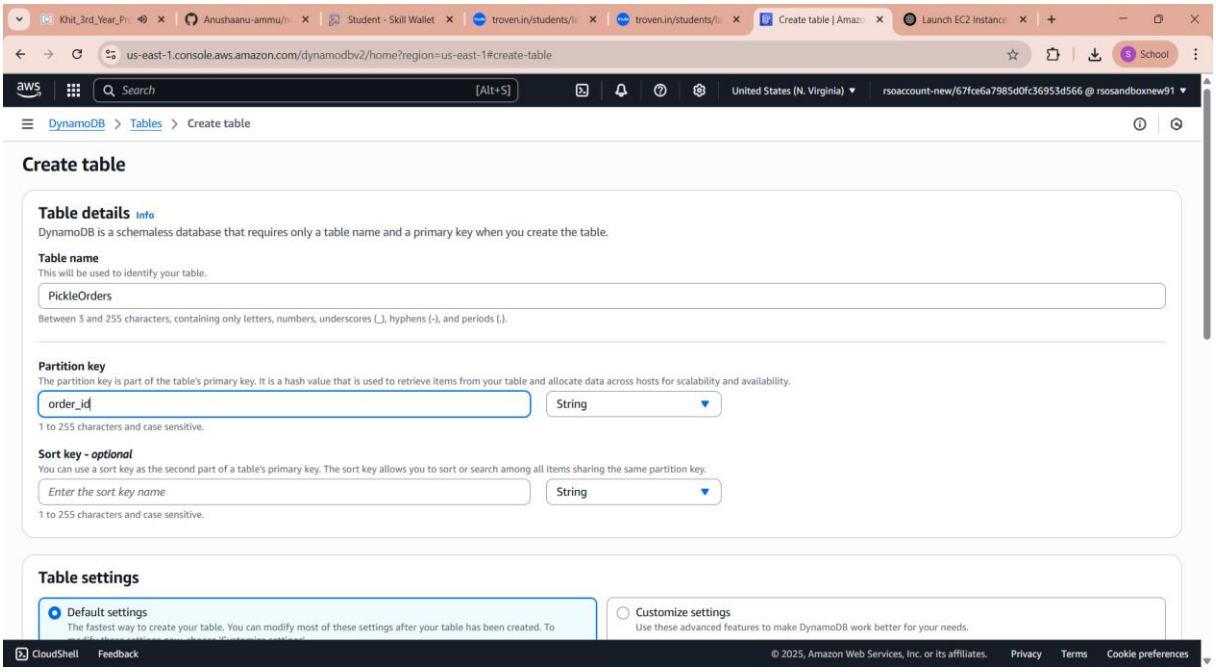
Activity 3.2: Create a DynamoDB table for storing registration details and book requests.

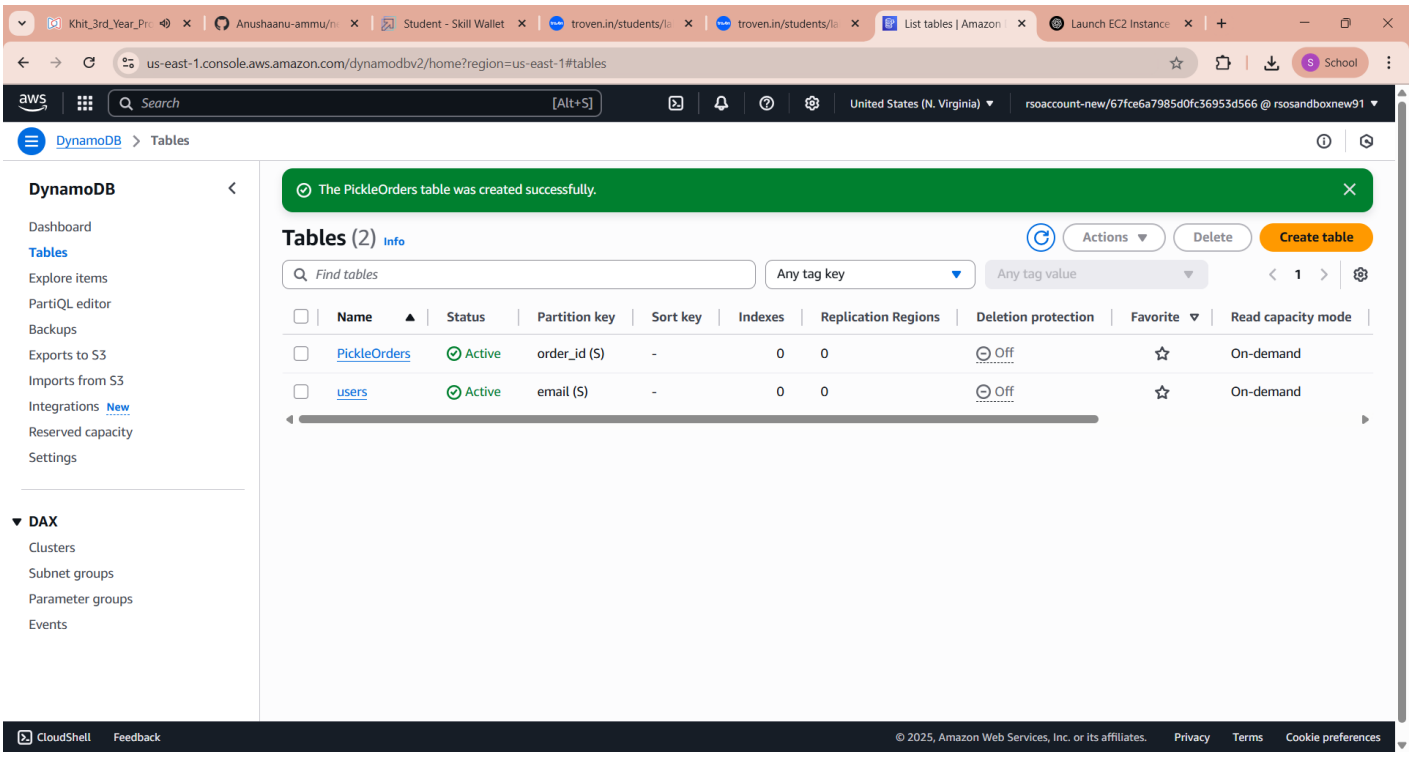
- Create Users table with partition key “Email ” with type String and click on create tables.


us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table
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.
users
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 availability.
email
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 settings
Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To
Customize settings
Use these advanced features to make DynamoDB work better for your needs.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables
DynamoDB > Tables
DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
The users table was created successfully.
Tables (1) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
Favorite
Read capacity mode
users
Active
email (S)
-
0
0
Off
On-demand
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

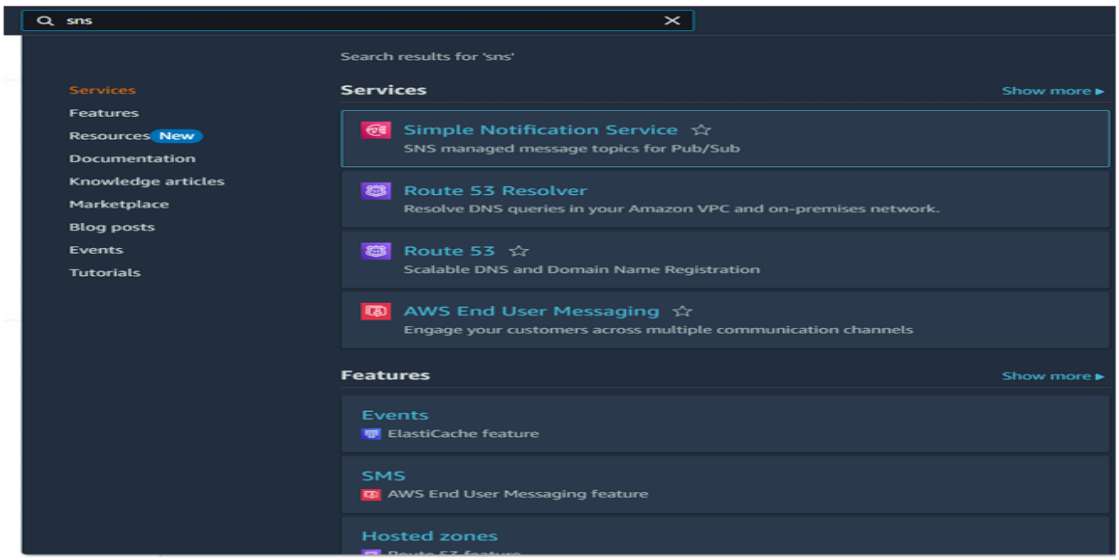
- Follow the same steps to create an Orders table with Order_id as the primary key to store Order details.



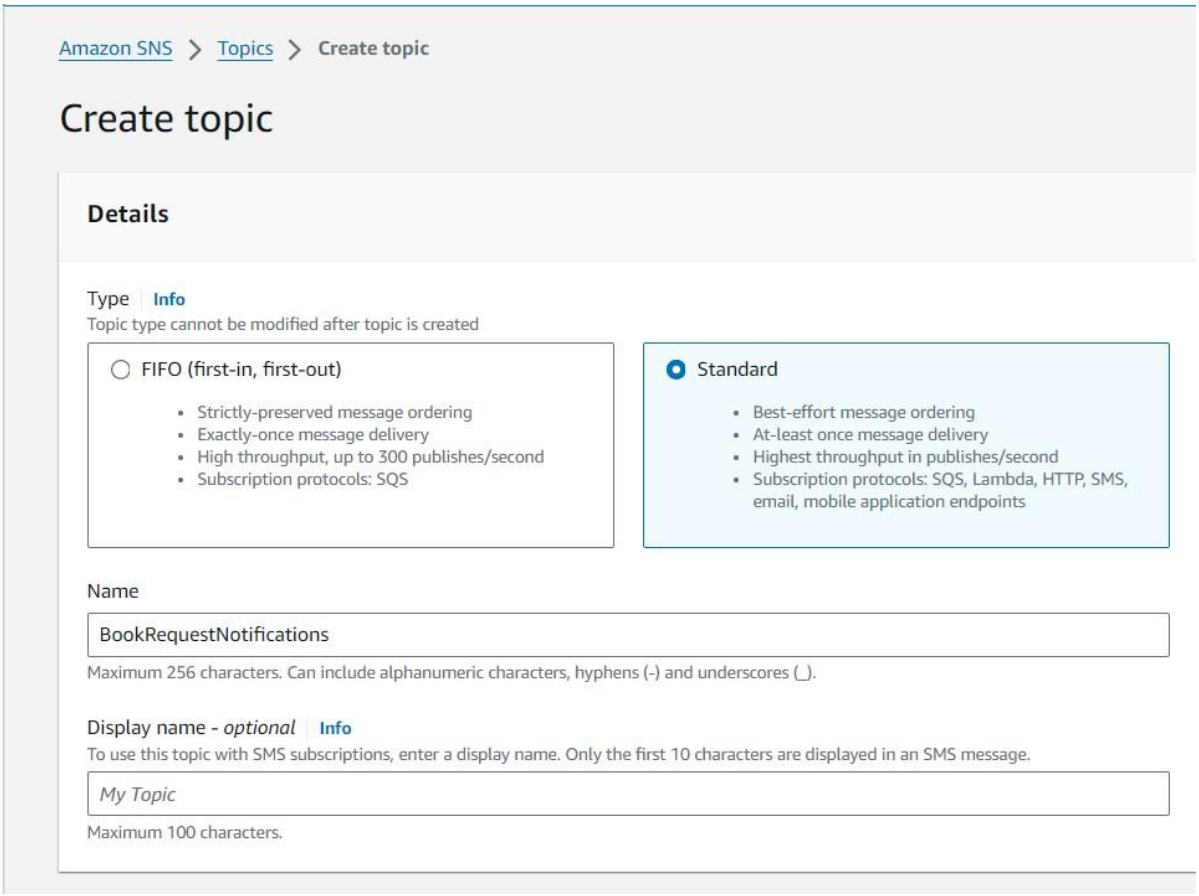


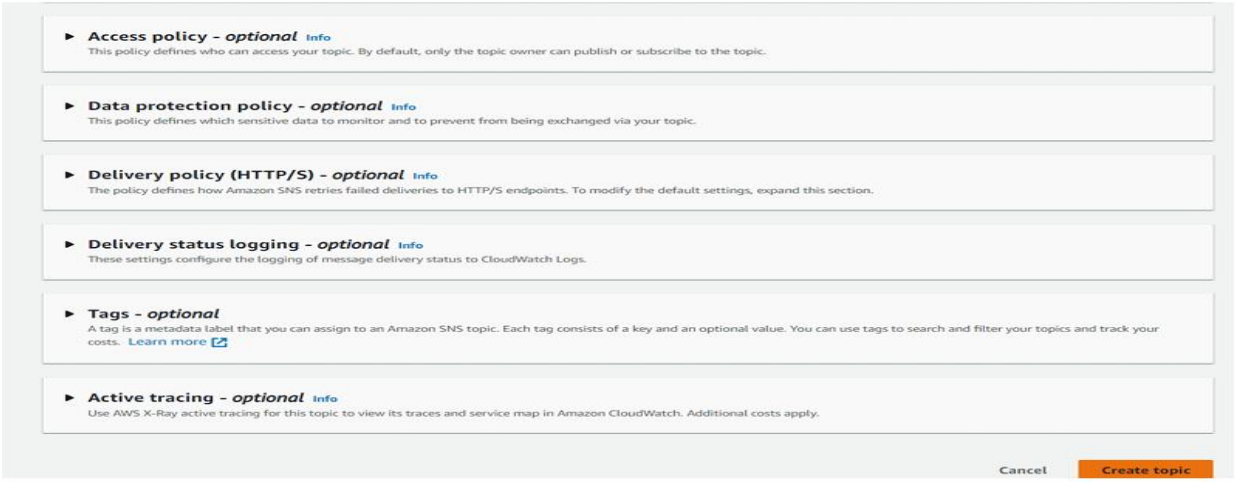
Milestone 4. SNS Notification Setup

- **Activiy4.1:** Create SNS topics for book request notifications**.**

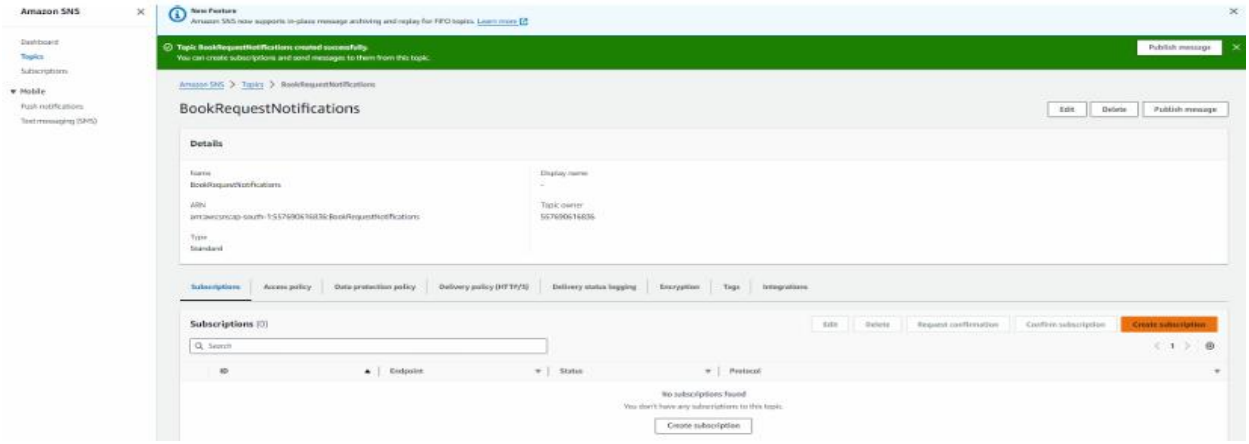


- Click on Create Topic and choose a name for the topic

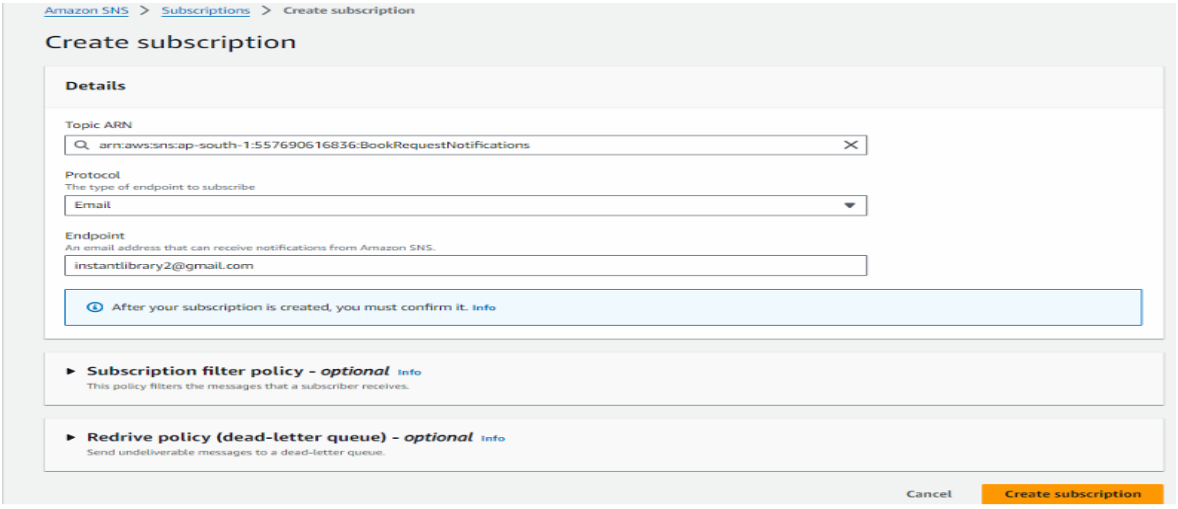


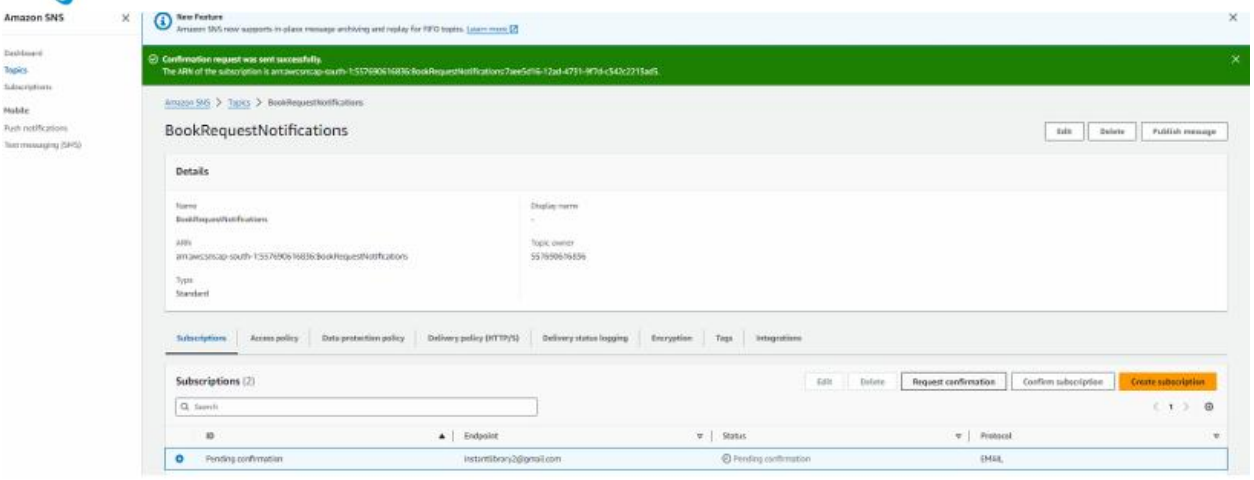


- Configure the SNS topic and note down the Topic ARN

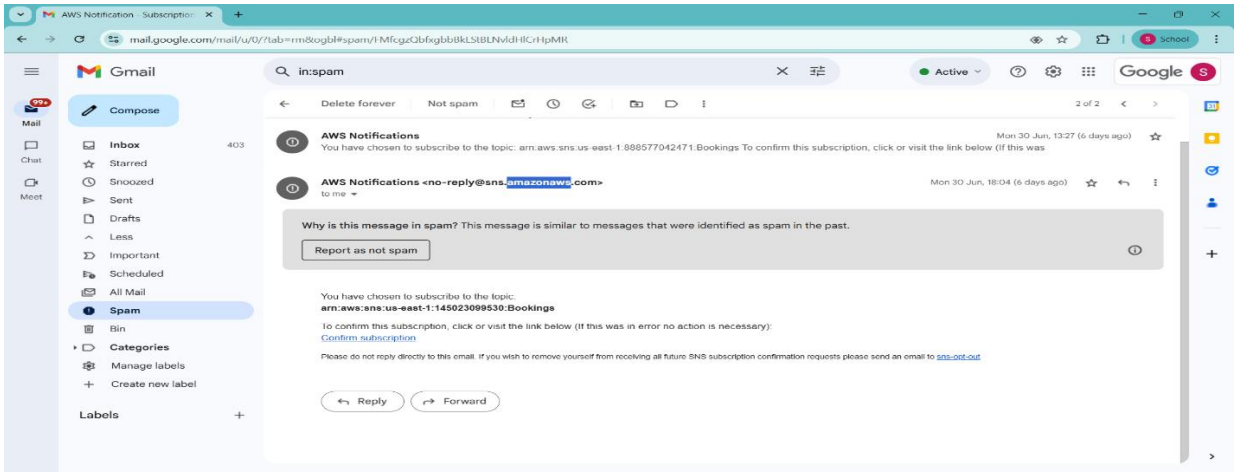


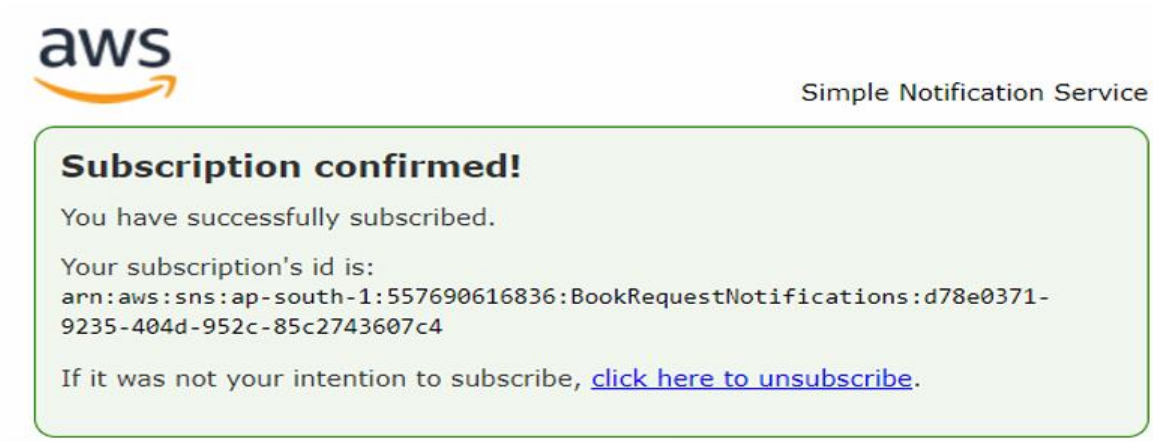
- Subscribe users and library staff to SNS email notifications.





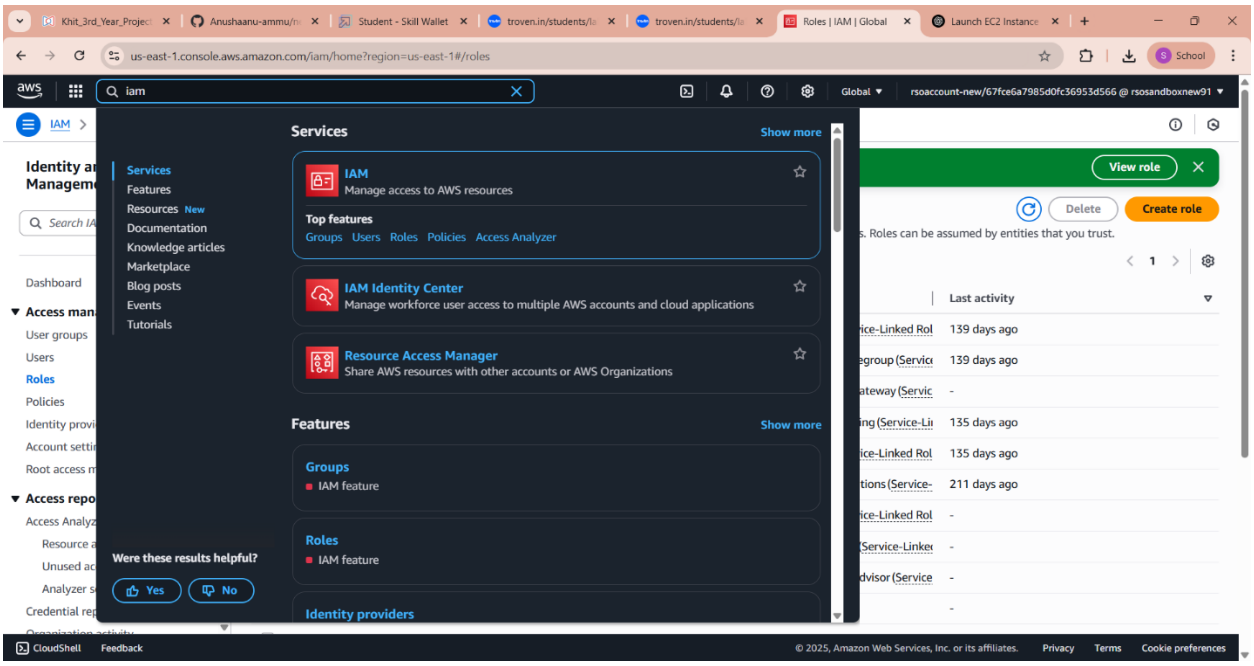
- After Comfirmation of Subscription going to Mail for comfirm Mail .

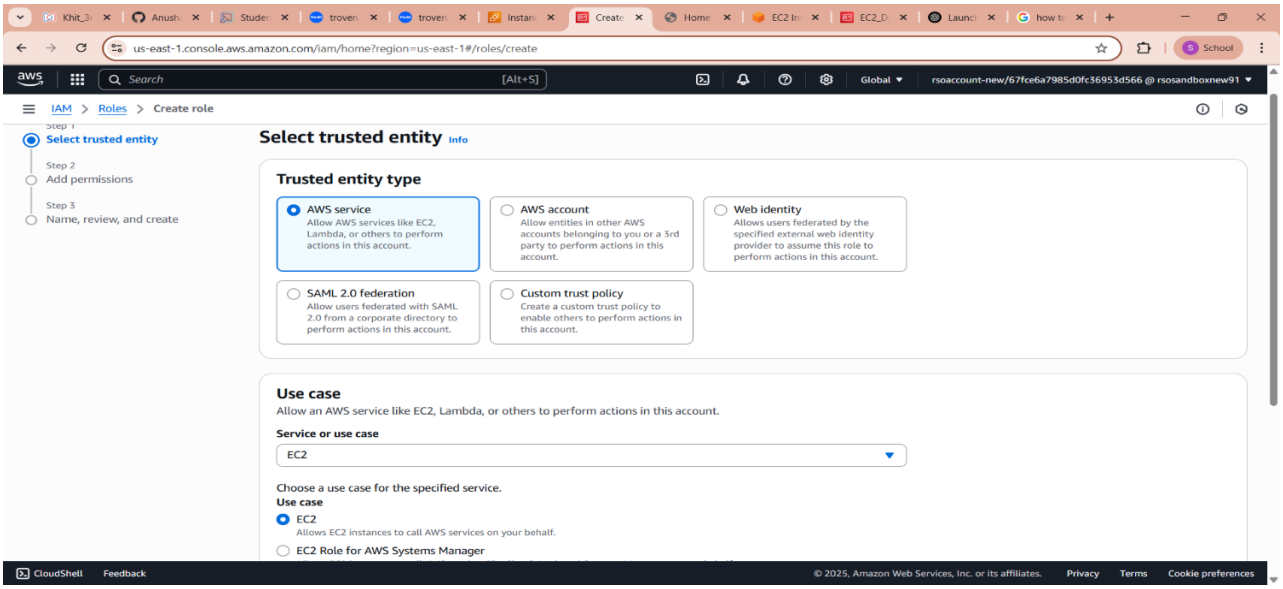




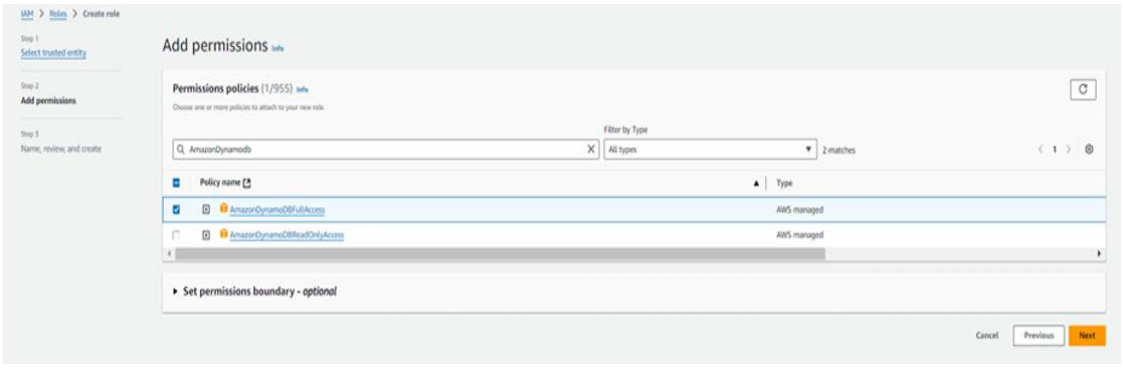
Milestone 5 : IAM Role Setup.

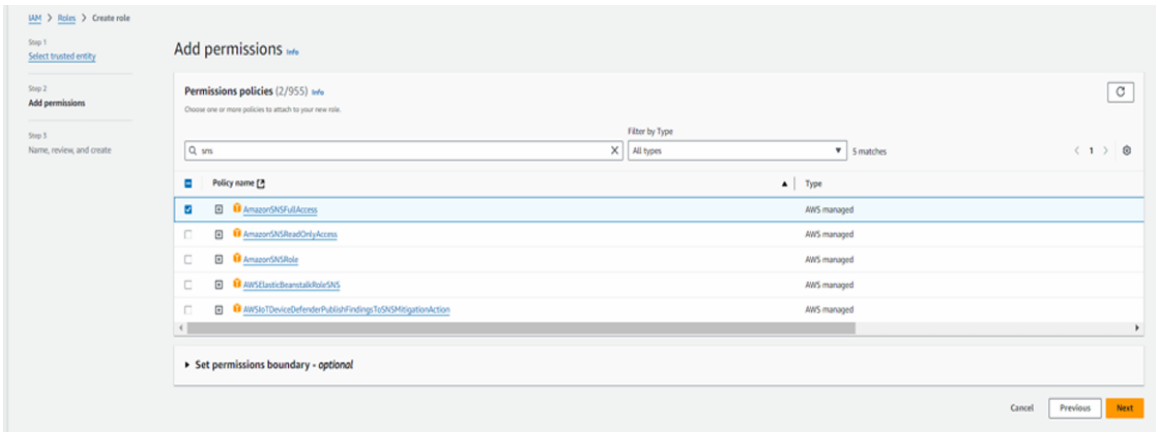
- **Activity 5.1 :** In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB .

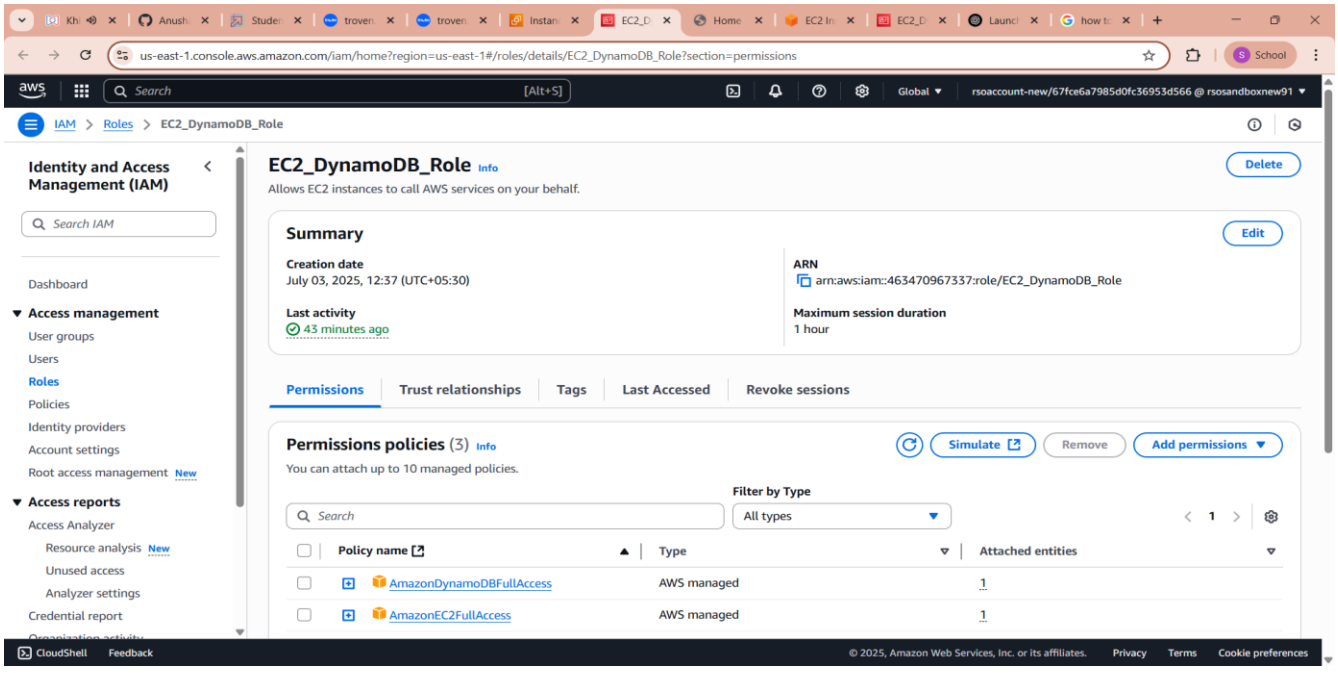




- **Activity 5.2 : Attach Policies**
- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

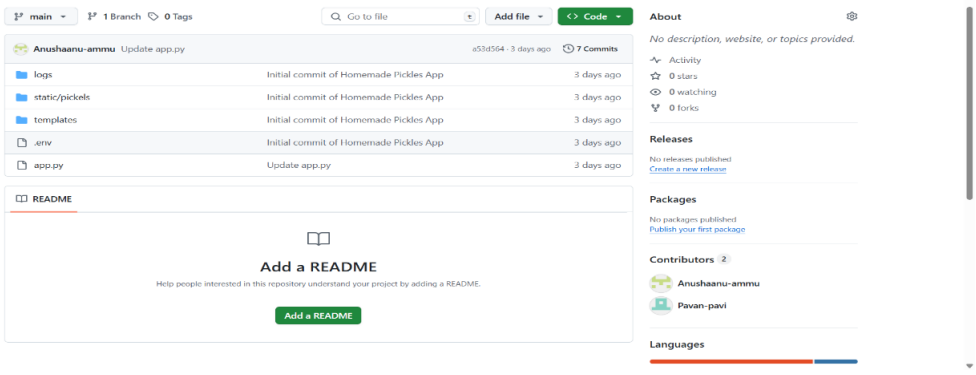

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (1/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
AmazonDynamodb
All types
2 matches
Policy name
Type
AmazonDynamoDBFullAccess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

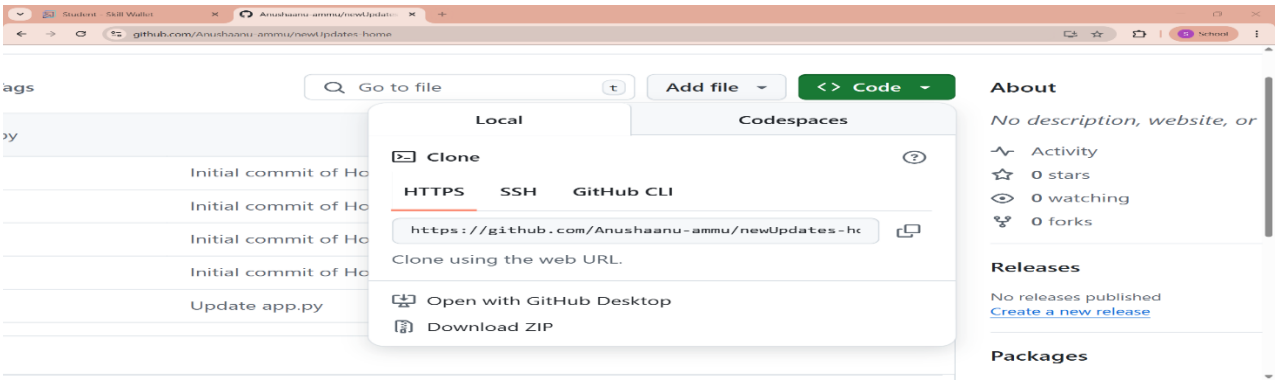

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
Policy name
Type
AmazonSNSFullAccess
AWS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


us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/EC2_DynamoDB_Role?section=permissions
Search
[Alt+S]
Global
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
IAM > Roles > EC2_DynamoDB_Role
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report
EC2_DynamoDB_Role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
July 03, 2025, 12:37 (UTC+05:30)
ARN
arn:aws:iam::463470967337:role/EC2_DynamoDB_Role
Last activity
43 minutes ago
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (3) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
1
AmazonEC2FullAccess
AWS managed
1
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

- ## Milestone 6. EC2 Instance Setup

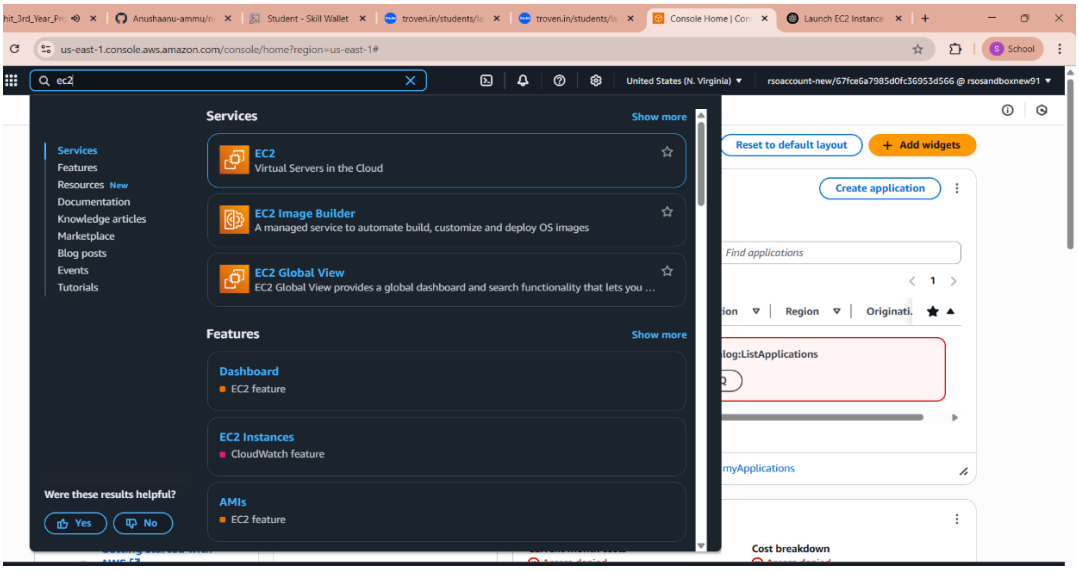
  Note: Load your Flask app and Html files into GitHub repository .





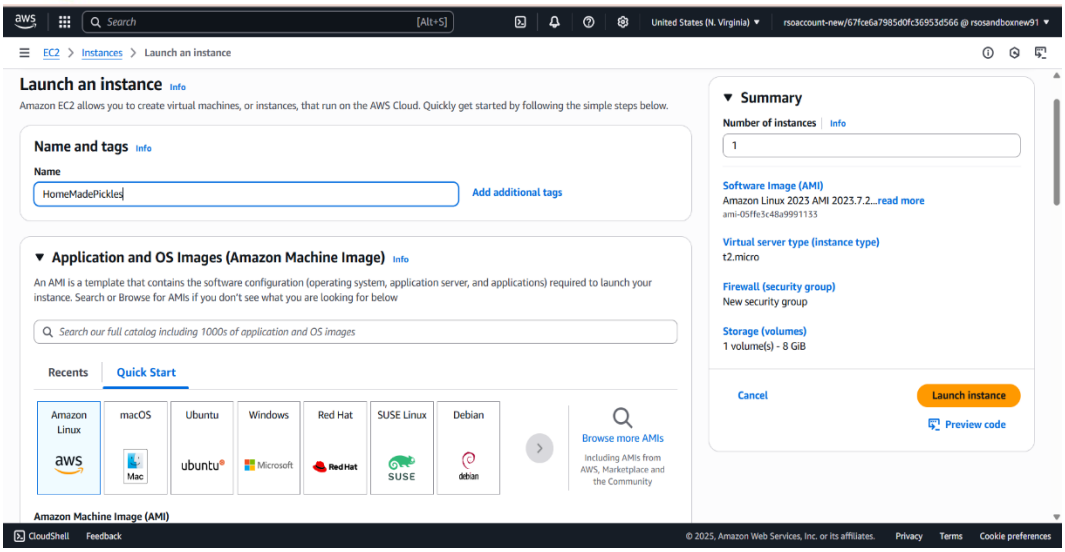
Activity 6.1: Launch an EC2 instance to host the Flask application.

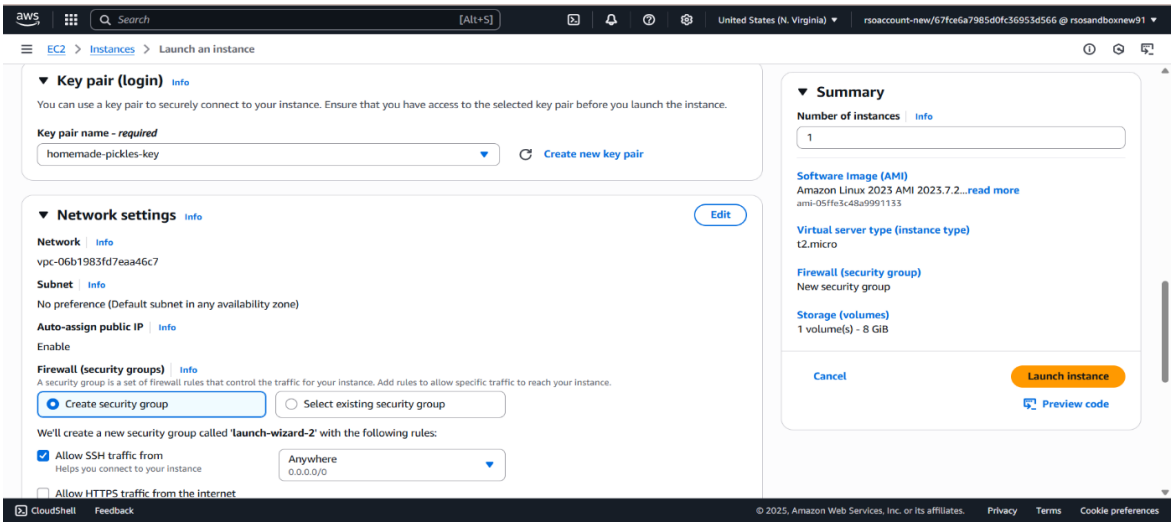
- In the AWS Console, navigate to EC2 and launch a new instance.

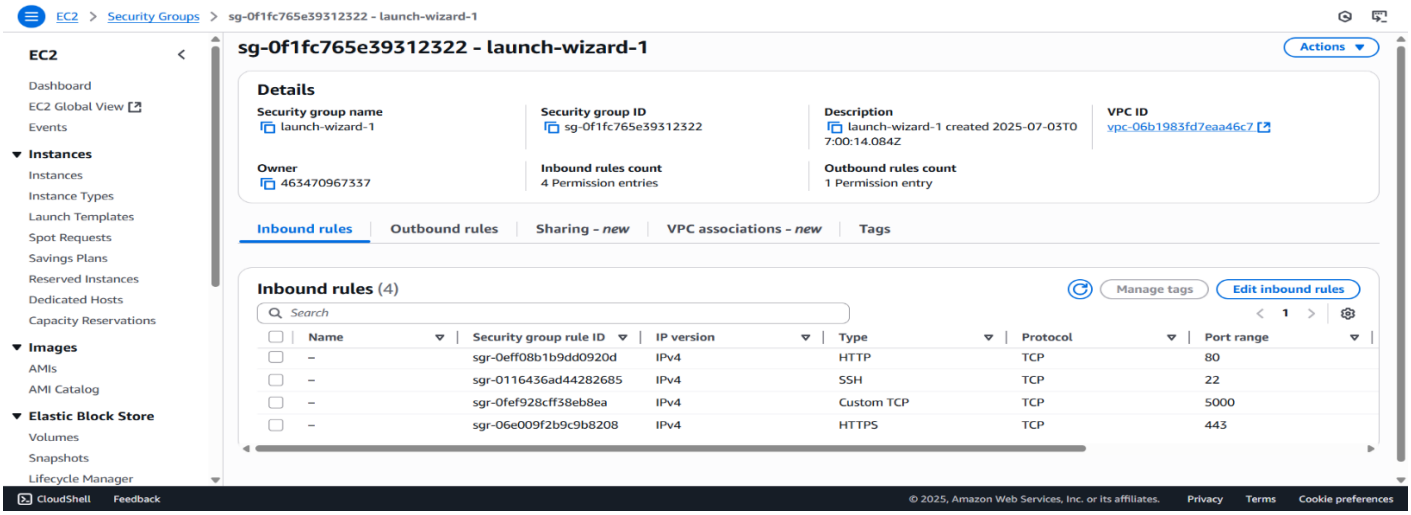


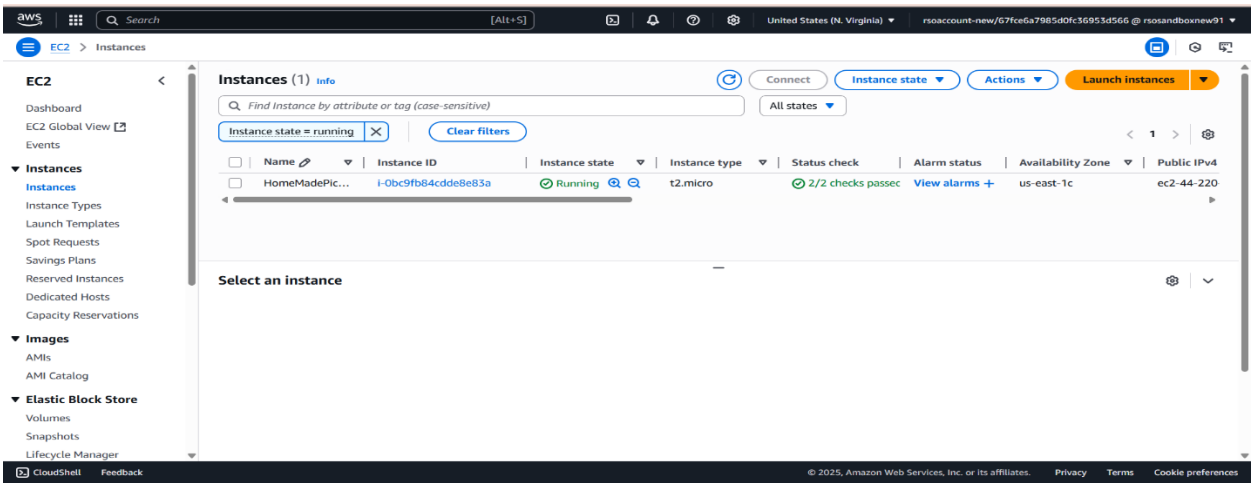
SMARTBRIDGE
Let's Bridge the Gap

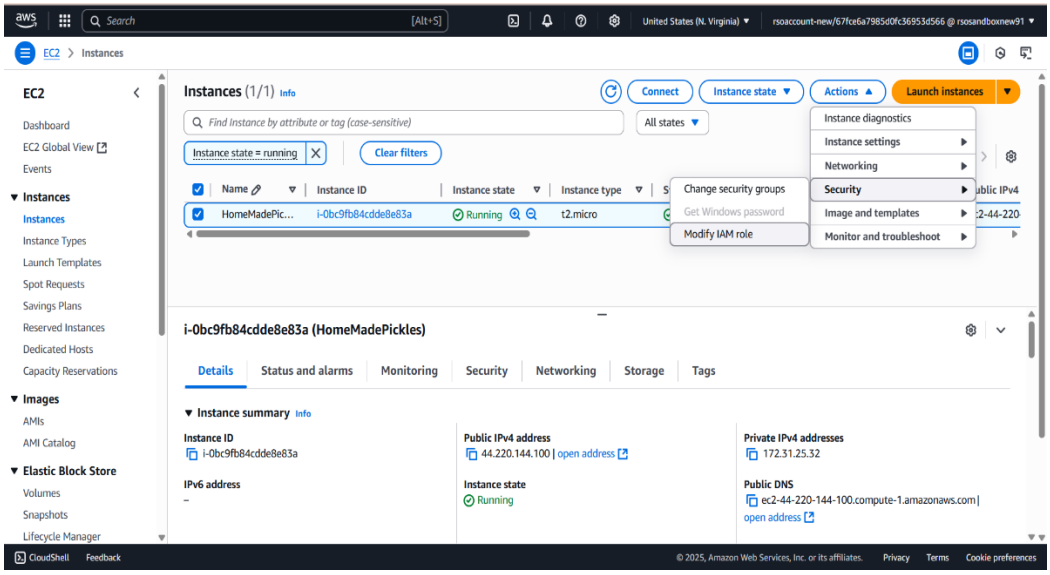
Smart Internz

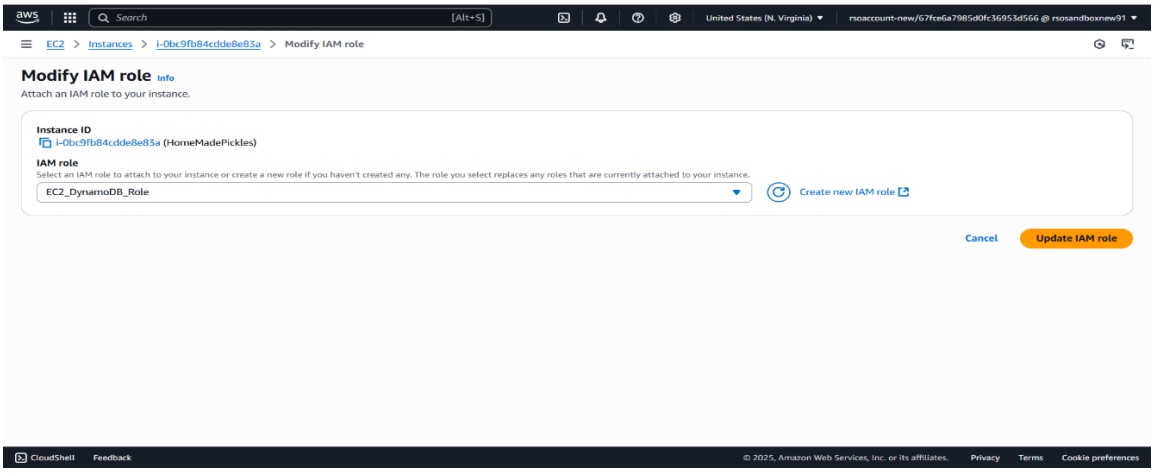


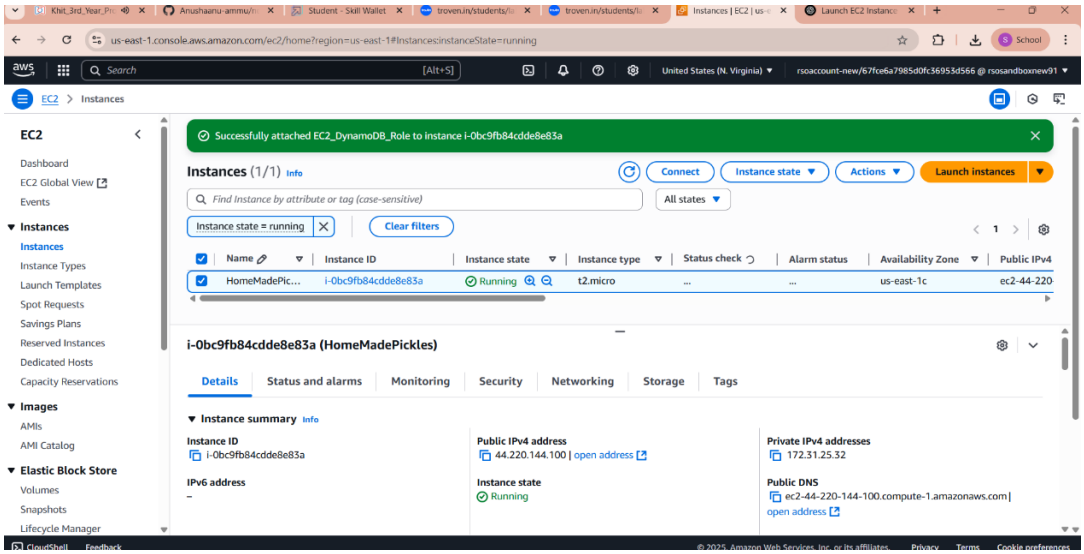


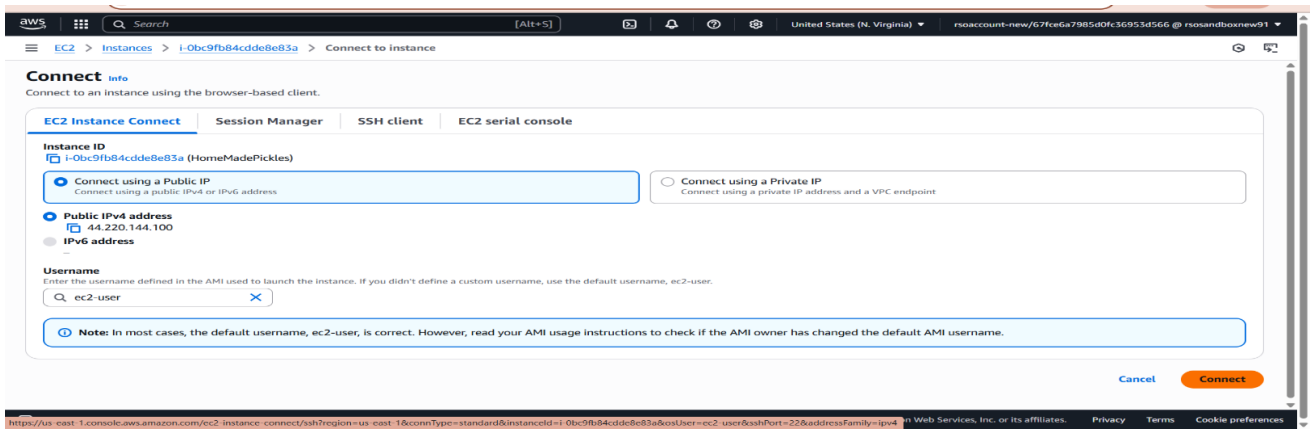


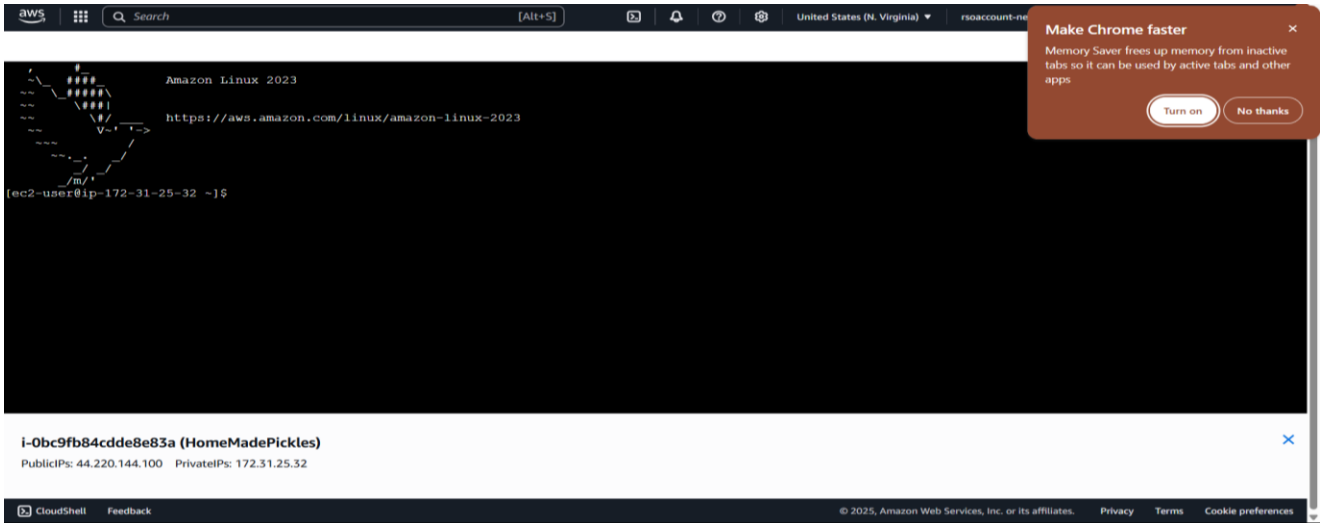

EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Instances (1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4
HomeMadePic...
i-0bc9fb84cdde8e83a
Running
t2.micro
2/2 checks passed
View alarms
us-east-1c
ec2-44-220
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


EC2 > Instances
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Instance diagnostics
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
i-0bc9fb84cdde8e83a (HomeMadePickles)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0bc9fb84cdde8e83a
Public IPv4 address
44.220.144.100 | open address
Private IPv4 addresses
172.31.25.32
IPv6 address
Instance state
Running
Public DNS
ec2-44-220-144-100.compute-1.amazonaws.com | open address


EC2 > Instances > i-0bc9fb84cdde8e83a > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-0bc9fb84cdde8e83a (HomeMadePickles)
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.
EC2_DynamoDB_Role
Create new IAM role
Cancel
Update IAM role


Successfully attached EC2_DynamoDB_Role to instance i-0bc9fb84cdde8e83a
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Instance state = running
Clear filters
HomeMadePic...
i-0bc9fb84cdde8e83a
Running
t2.micro
us-east-1c
i-0bc9fb84cdde8e83a (HomeMadePickles)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0bc9fb84cdde8e83a
Public IPv4 address
44.220.144.100 | open address
Private IPv4 addresses
172.31.25.32
IPv6 address
Instance state
Running
Public DNS
ec2-44-220-144-100.compute-1.amazonaws.com | open address


EC2 > Instances > i-0bc9fb84cdde8e83a > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0bc9fb84cdde8e83a (HomeMadePickles)
Connect using a Public IP
Connect using a Private IP
Public IPv4 address
44.220.144.100
IPv6 address
Username
ec2-user
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


Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-25-32 ~]$
Make Chrome faster
Memory Saver frees up memory from inactive tabs so it can be used by active tabs and other apps
Turn on
No thanks
i-0bc9fb84cdde8e83a (HomeMadePickles)
PublicIPs: 44.220.144.100 PrivateIPs: 172.31.25.32
CloudShell
Feedback

Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

- **Install Python3, Flask, and Git:**
- **On Amazon Linux 2:**
- **sudo yum update -y**
- **sudo yum install python3 git**
- **sudo pip3 install flask boto3**
- **Verify Installations:**
- **flask --version**
- **git –version**

Activity 7.2 : Clone Your Flask Project from GitHub.

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

Run :’ git clone https://github.com/Anushaanu-ammu/newUpdates-home’ .

This will download the Project to Ec2 instance .

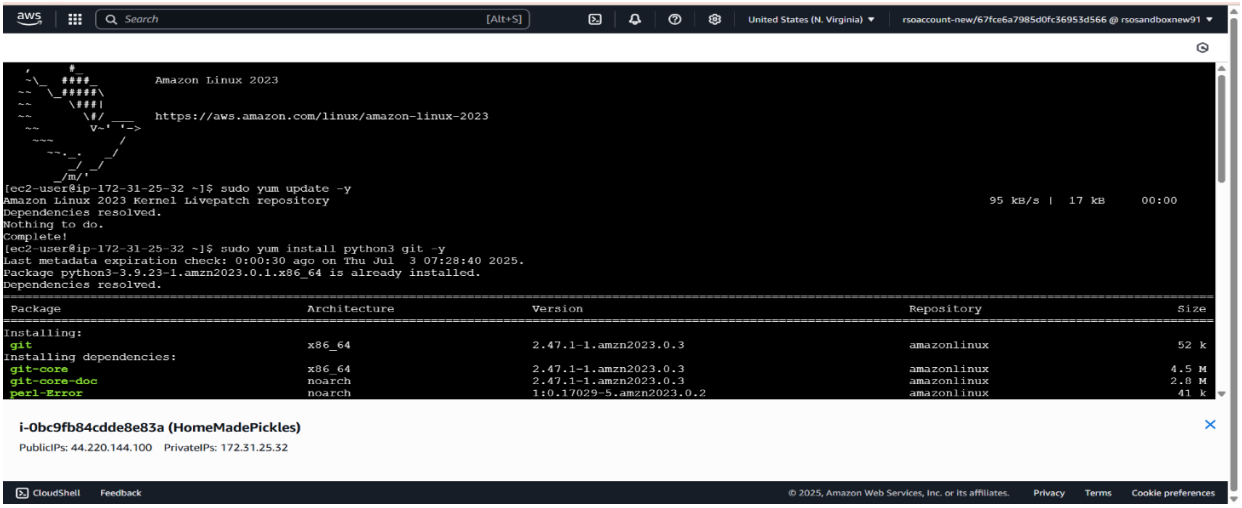
To navigate to the project directory, run the following command:

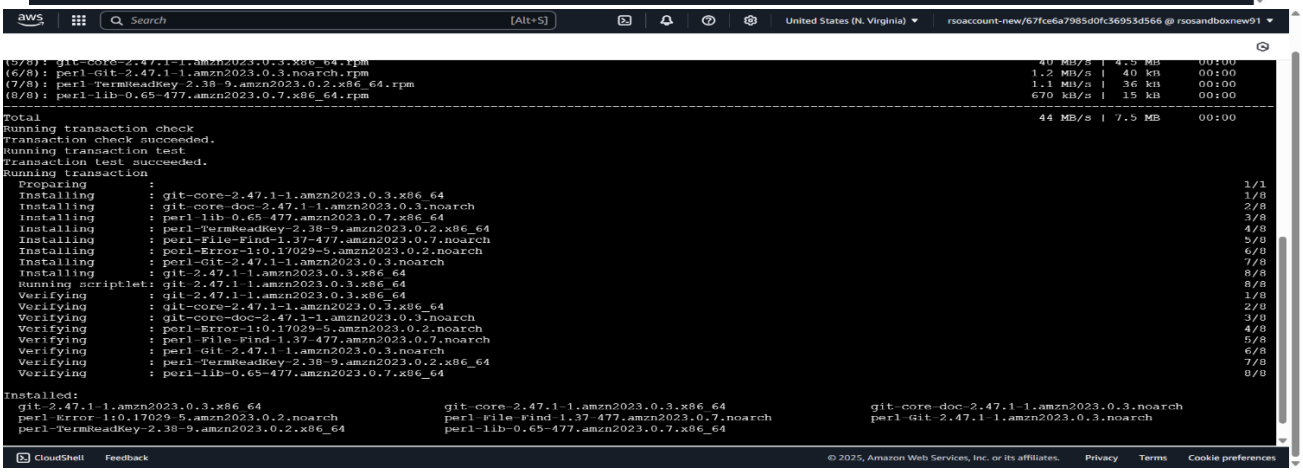
cd InstantLibrary

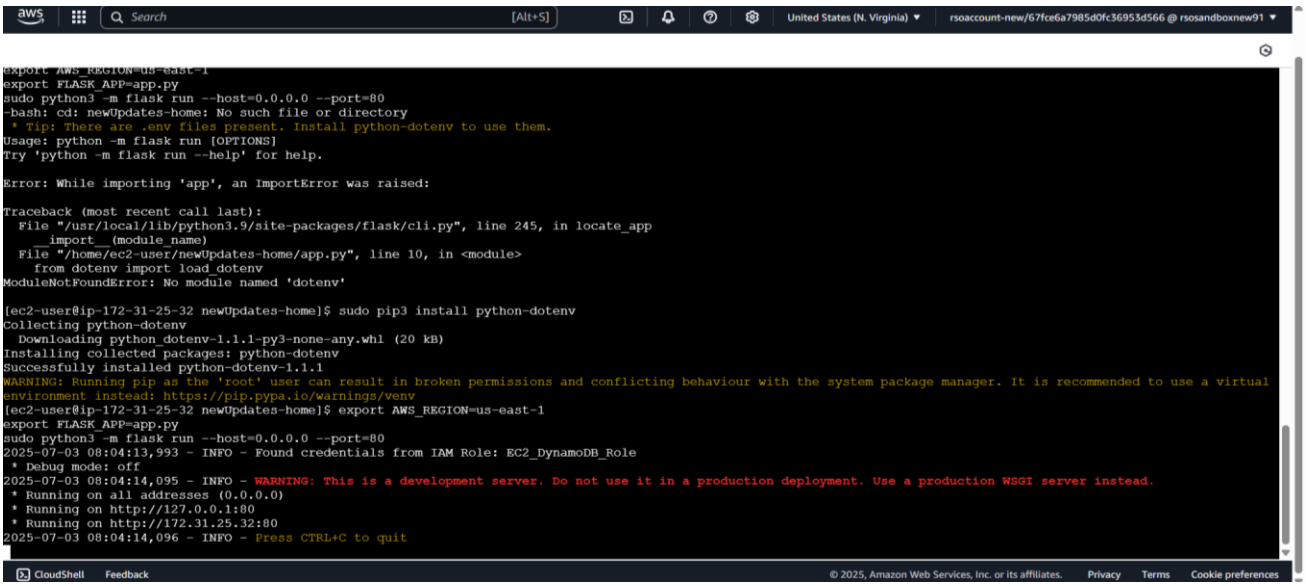
Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

sudo flask run --host=0.0.0.0 --port=80


Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-25-32 ~]$ sudo yum update -y
Amazon Linux 2023 Kernel Livepatch repository
95 kB/s | 17 kB 00:00
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-25-32 ~]$ sudo yum install python3 git -y
Last metadata expiration check: 0:00:30 ago on Thu Jul 3 07:28:40 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Package
Architecture
Version
Repository
Size
Installing:
git
x86_64
2.47.1-1.amzn2023.0.3
amazonlinux
52 k
Installing dependencies:
git-core
x86_64
2.47.1-1.amzn2023.0.3
amazonlinux
4.5 M
git-core-doc
noarch
2.47.1-1.amzn2023.0.3
amazonlinux
2.8 M
perl-Error
noarch
1:0.17029-5.amzn2023.0.2
amazonlinux
41 k
i-0bc9fb84cdde8e83a (HomeMadePickles)
PublicIPs: 44.220.144.100 PrivateIPs: 172.31.25.32
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
(6/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm
1.2 MB/s | 40 kB 00:00
(7/8): perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64.rpm
1.1 MB/s | 36 kB 00:00
(8/8): perl-lib-0.65-477.amzn2023.0.7.x86_64.rpm
670 kB/s | 15 kB 00:00
Total
44 MB/s | 7.5 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : git-core-2.47.1-1.amzn2023.0.3.x86_64 1/8
Installing : git-core-doc-2.47.1-1.amzn2023.0.3.noarch 2/8
Installing : perl-lib-0.65-477.amzn2023.0.7.x86_64 3/8
Installing : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 4/8
Installing : perl-File-Find-1.37-477.amzn2023.0.7.noarch 5/8
Installing : perl-Error-1:0.17029-5.amzn2023.0.2.noarch 6/8
Installing : perl-Git-2.47.1-1.amzn2023.0.3.noarch 7/8
Installing : git-2.47.1-1.amzn2023.0.3.x86_64 8/8
Running scriptlet: git-2.47.1-1.amzn2023.0.3.x86_64 8/8
Verifying : git-2.47.1-1.amzn2023.0.3.x86_64 1/8
Verifying : git-core-2.47.1-1.amzn2023.0.3.x86_64 2/8
Verifying : git-core-doc-2.47.1-1.amzn2023.0.3.noarch 3/8
Verifying : perl-Error-1:0.17029-5.amzn2023.0.2.noarch 4/8
Verifying : perl-File-Find-1.37-477.amzn2023.0.7.noarch 5/8
Verifying : perl-Git-2.47.1-1.amzn2023.0.3.noarch 6/8
Verifying : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 7/8
Verifying : perl-lib-0.65-477.amzn2023.0.7.x86_64 8/8
Installed:
git-2.47.1-1.amzn2023.0.3.x86_64
git-core-2.47.1-1.amzn2023.0.3.x86_64
git-core-doc-2.47.1-1.amzn2023.0.3.noarch
perl-Error-1:0.17029-5.amzn2023.0.2.noarch
perl-File-Find-1.37-477.amzn2023.0.7.noarch
perl-Git-2.47.1-1.amzn2023.0.3.noarch
perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64
perl-lib-0.65-477.amzn2023.0.7.x86_64
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
export AWS_REGION=us-east-1
export FLASK_APP=app.py
sudo python3 -m flask run --host=0.0.0.0 --port=80
-bash: cd: newUpdates-home: No such file or directory
* Tip: There are .env files present. Install python-dotenv to use them.
Usage: python -m flask run [OPTIONS]
Try 'python -m flask run --help' for help.
Error: While importing 'app', an ImportError was raised:
Traceback (most recent call last):
File "/usr/local/lib/python3.9/site-packages/flask/cli.py", line 245, in locate_app
__import__(module_name)
File "/home/ec2-user/newUpdates-home/app.py", line 10, in <module>
from dotenv import load_dotenv
ModuleNotFoundError: No module named 'dotenv'
[ec2-user@ip-172-31-25-32 newUpdates-home]$ sudo pip3 install python-dotenv
Collecting python-dotenv
Downloading python_dotenv-1.1.1-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.1
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-172-31-25-32 newUpdates-home]$ export AWS_REGION=us-east-1
export FLASK_APP=app.py
sudo python3 -m flask run --host=0.0.0.0 --port=80
2025-07-03 08:04:13,993 - INFO - Found credentials from IAM Role: EC2_DynamoDB_Role
* Debug mode: off
2025-07-03 08:04:14,095 - INFO - 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://127.0.0.1:80
* Running on http://172.31.25.32:80
2025-07-03 08:04:14,096 - INFO - Press CTRL+C to quit
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

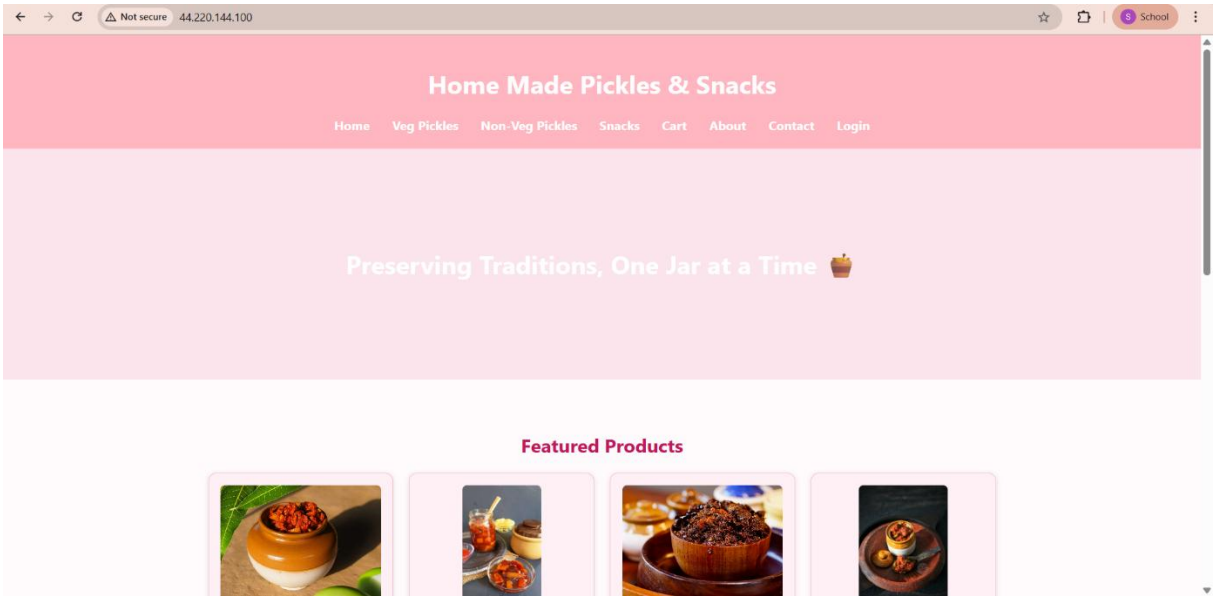
Access the website through:

Public IPs: http://44.220.144.100/

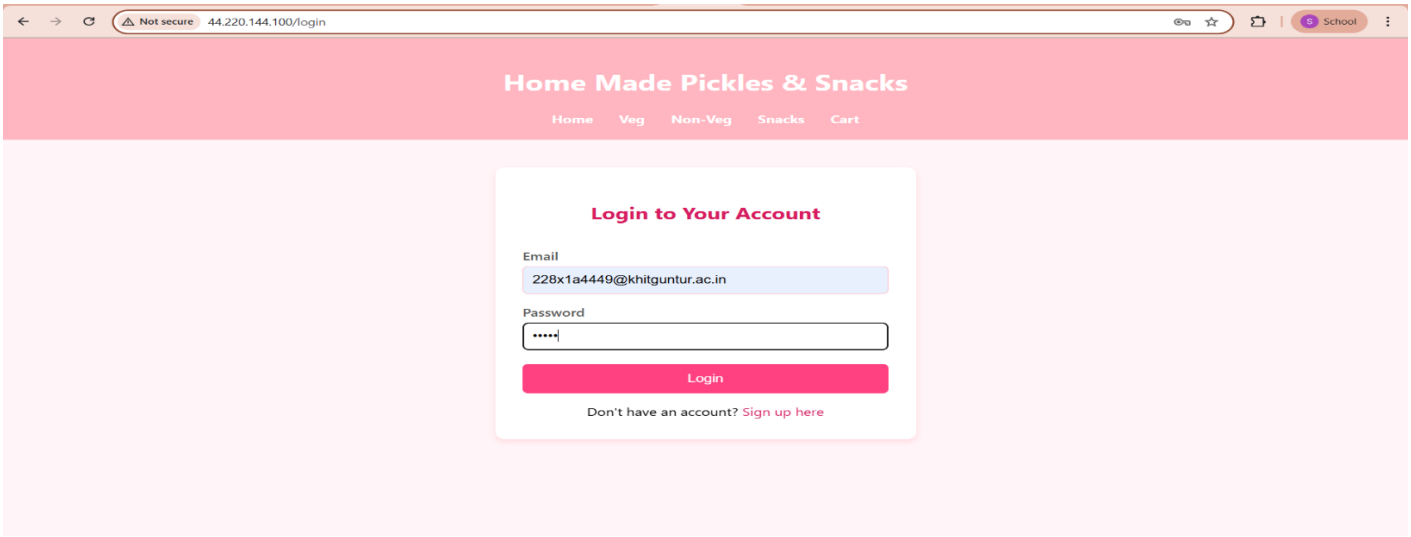
Milestone 8. Testing and Deployment

Activity 8.1 : Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

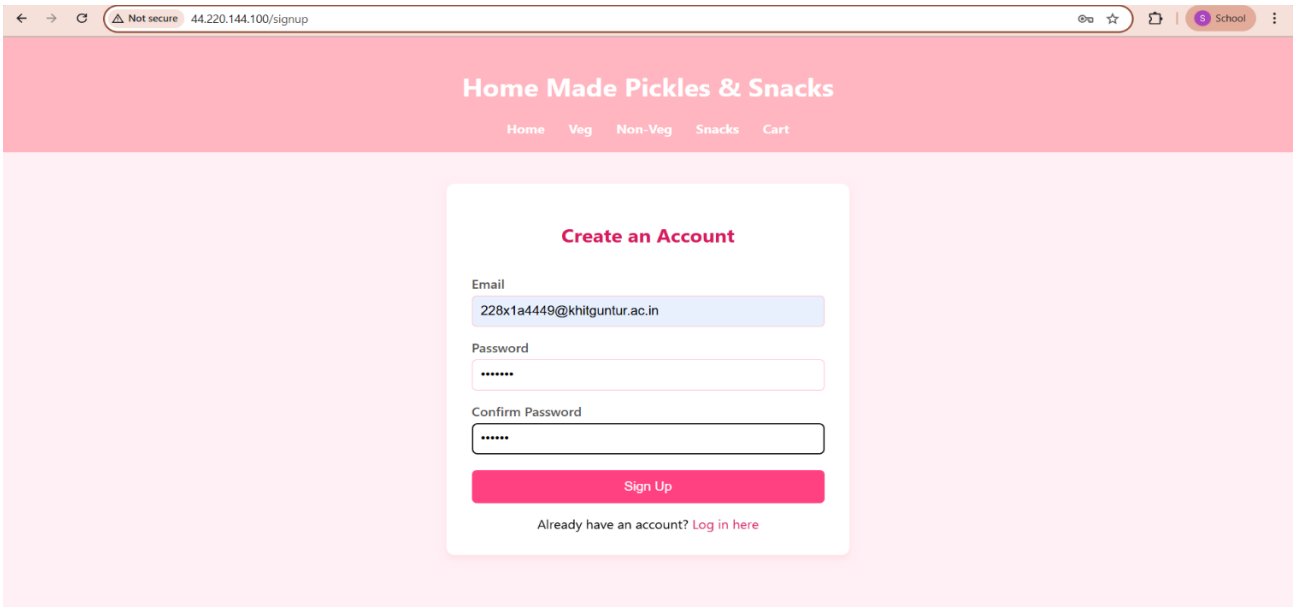
HOME PAGE:



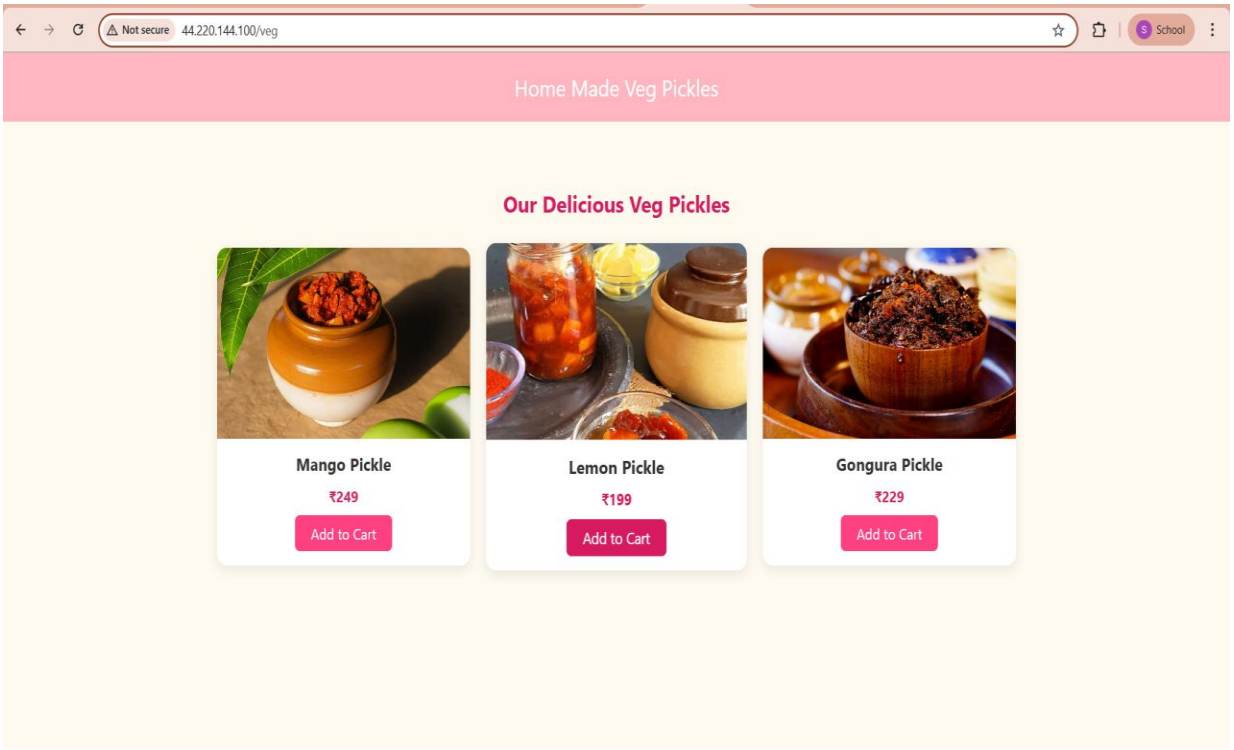
LOGIN PAGE :



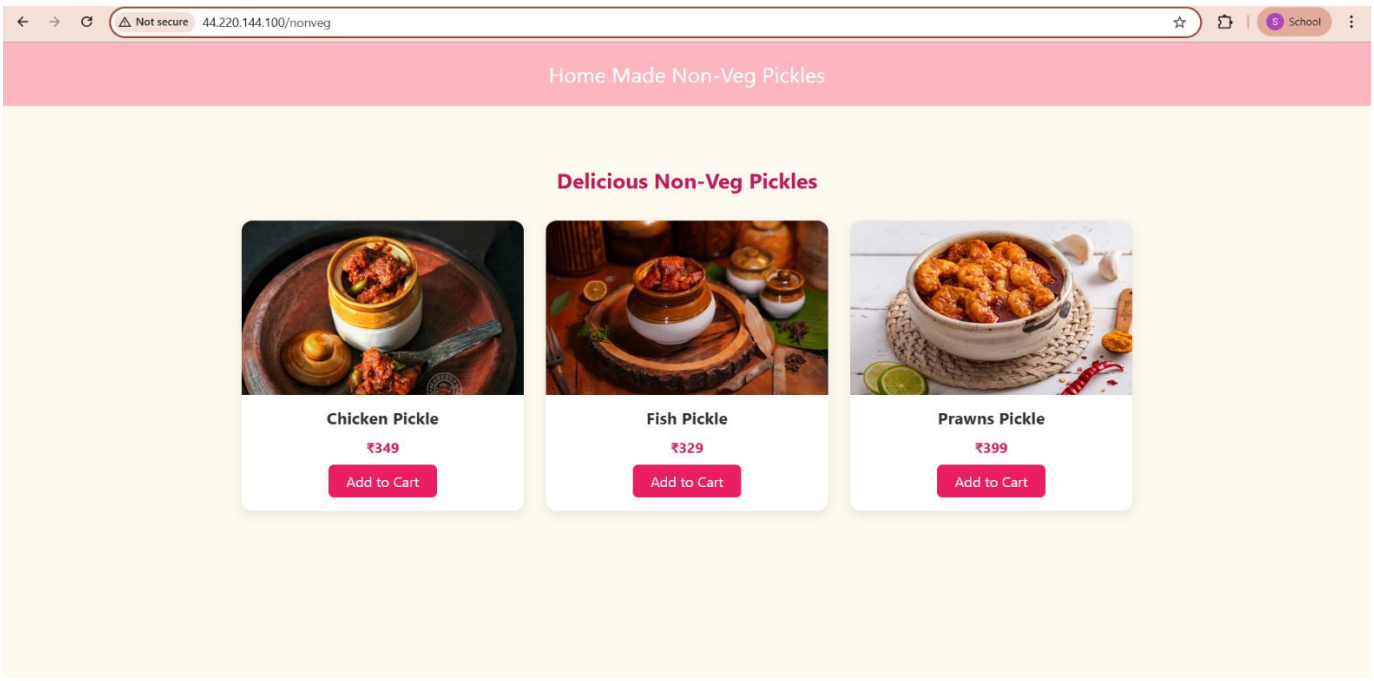
SIGNUP PAGE :



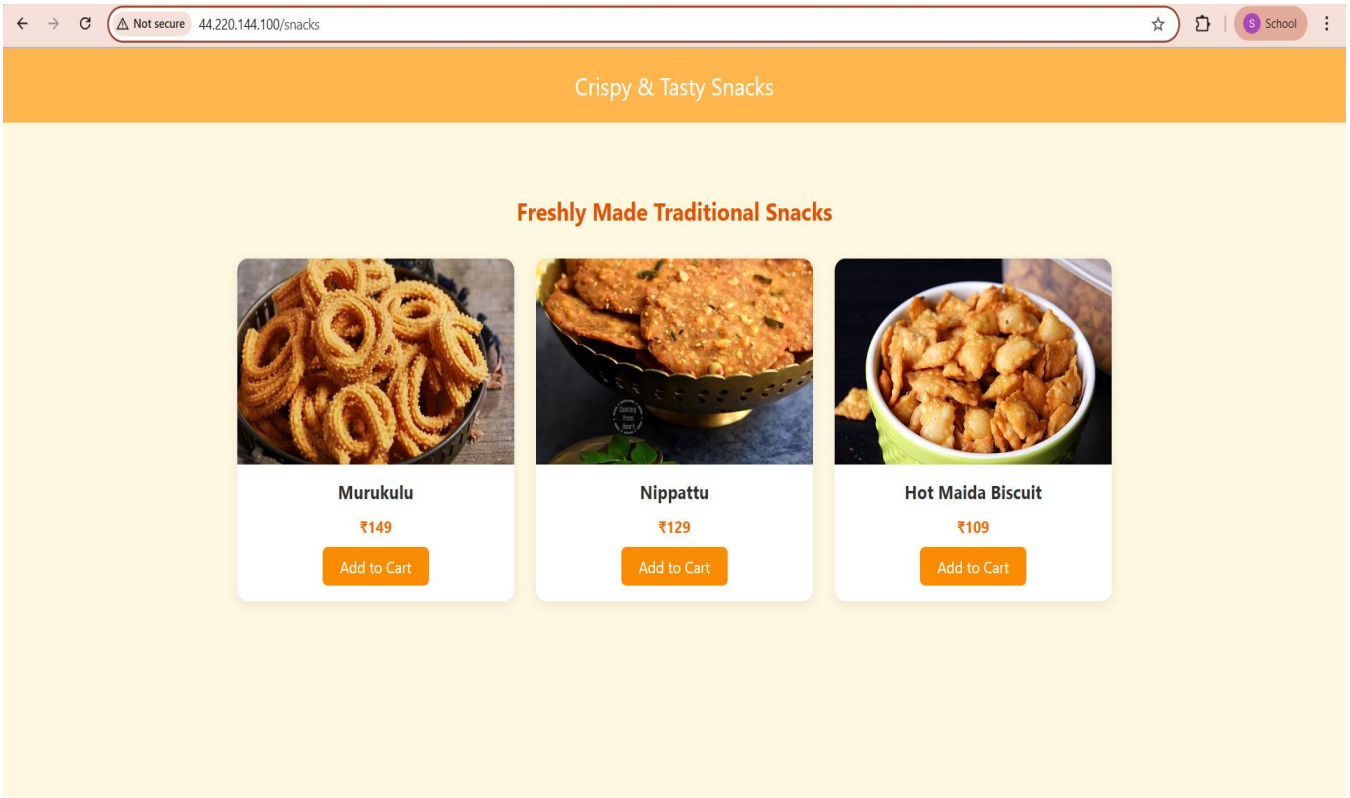
VEG PAGE :



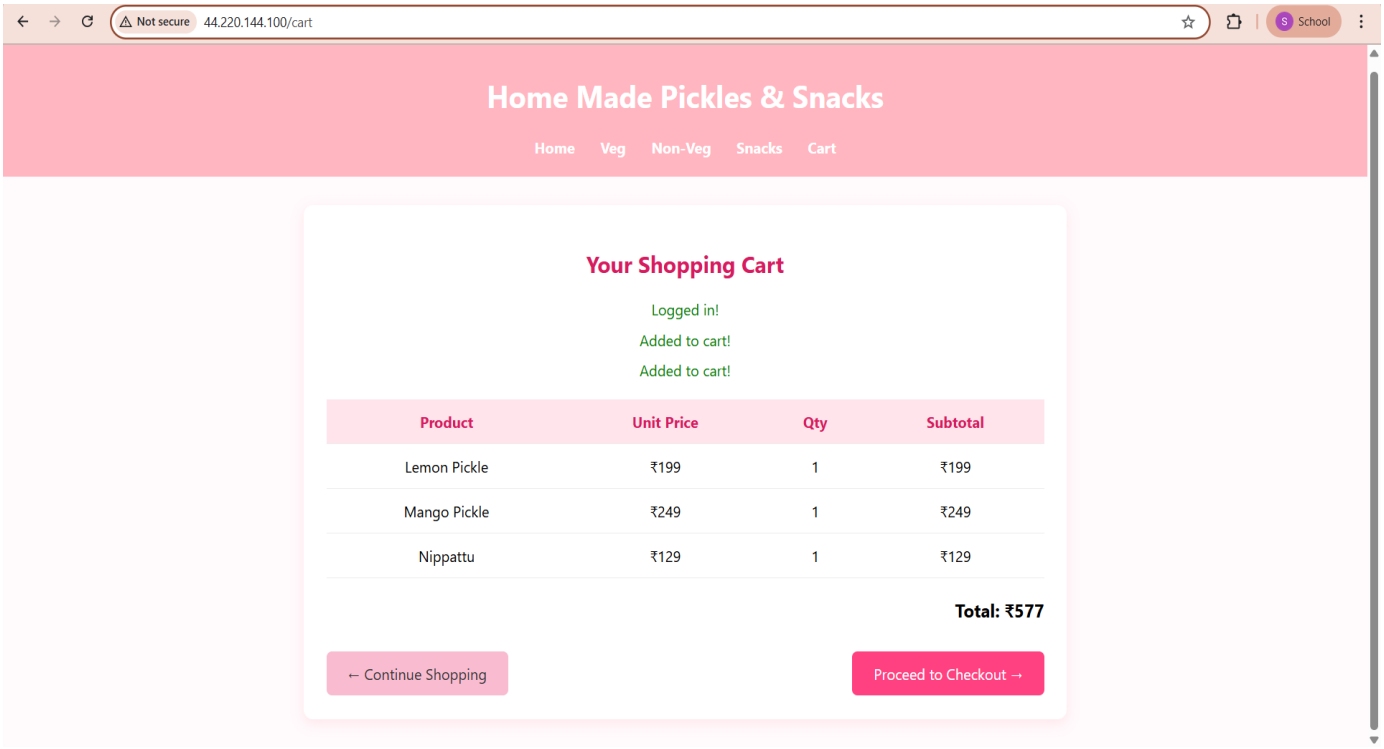
NON VEG PAGE :


Not secure
44.220.144.100/nonveg
School
Home Made Non-Veg Pickles
Delicious Non-Veg Pickles
Chicken Pickle
₹349
Add to Cart
Fish Pickle
₹329
Add to Cart
Prawns Pickle
₹399
Add to Cart

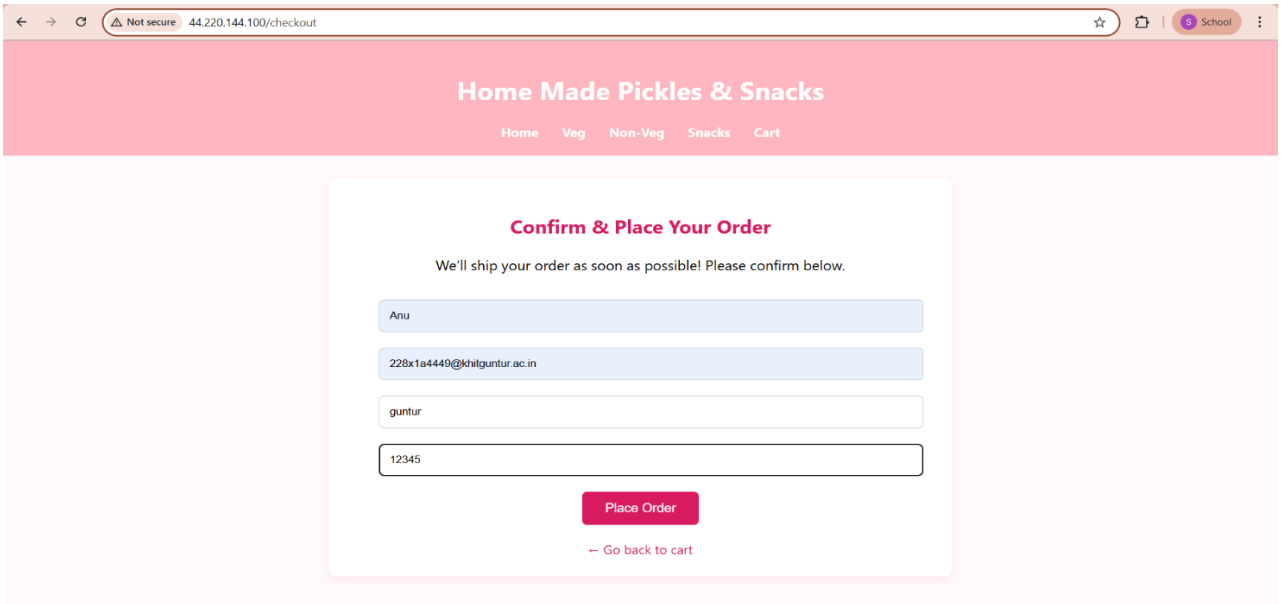
SNACKS PAGE :


Not secure
44.220.144.100/snacks
School
Crispy & Tasty Snacks
Freshly Made Traditional Snacks
Murukulu
₹149
Add to Cart
Nippattu
₹129
Add to Cart
Hot Maida Biscuit
₹109
Add to Cart

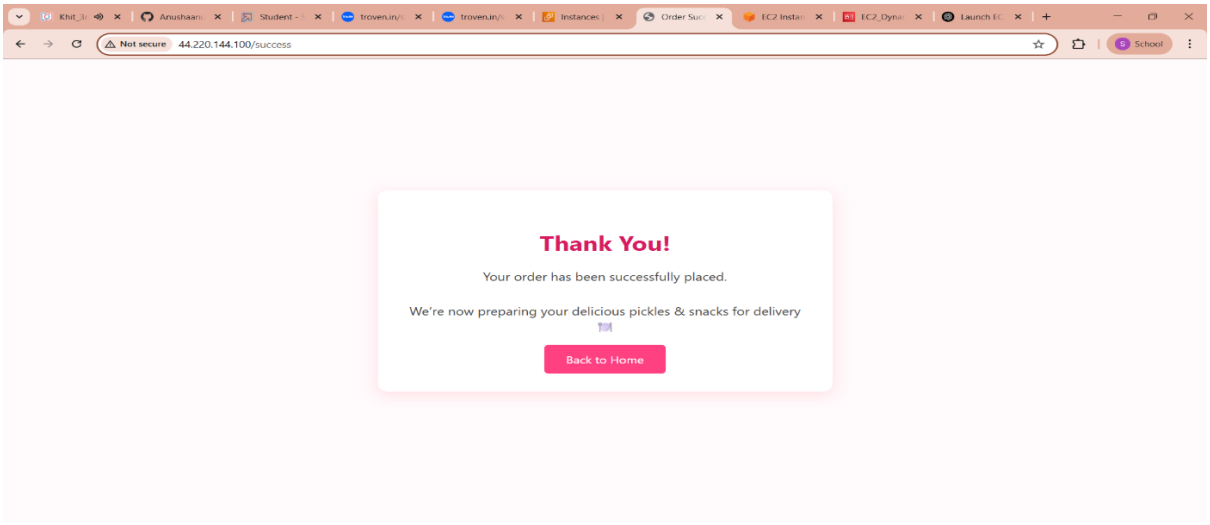
CART PAGE :

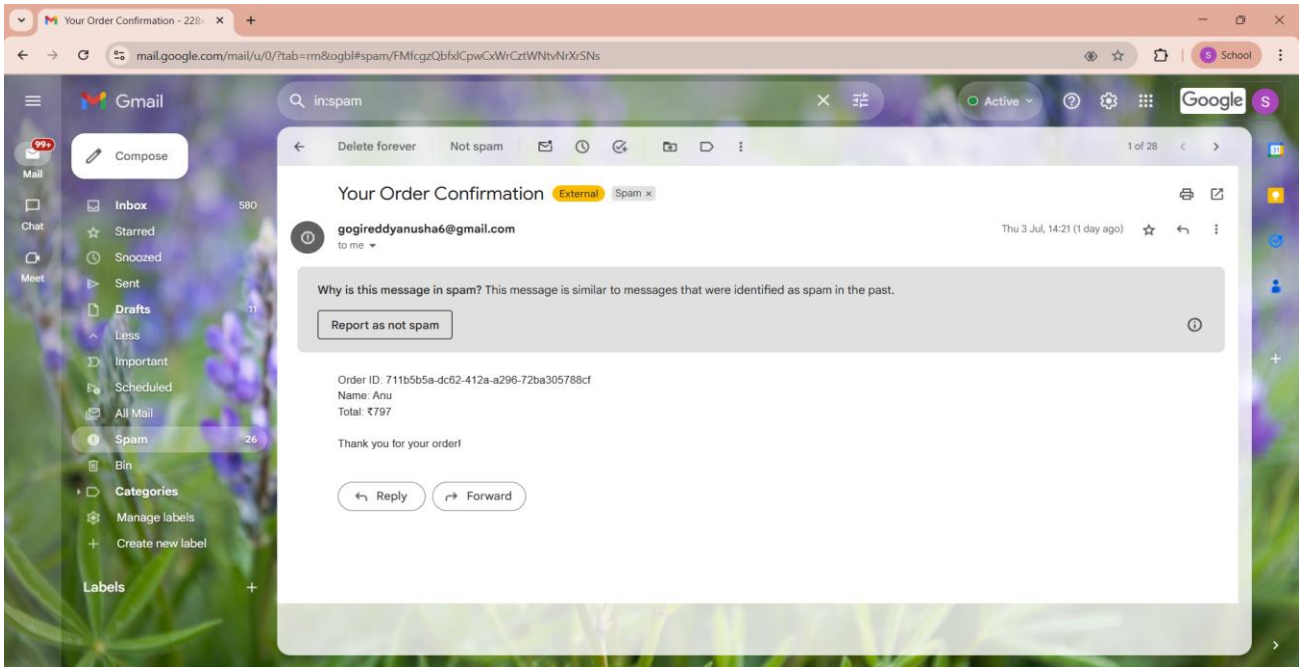


CHECK OUT PAGE :

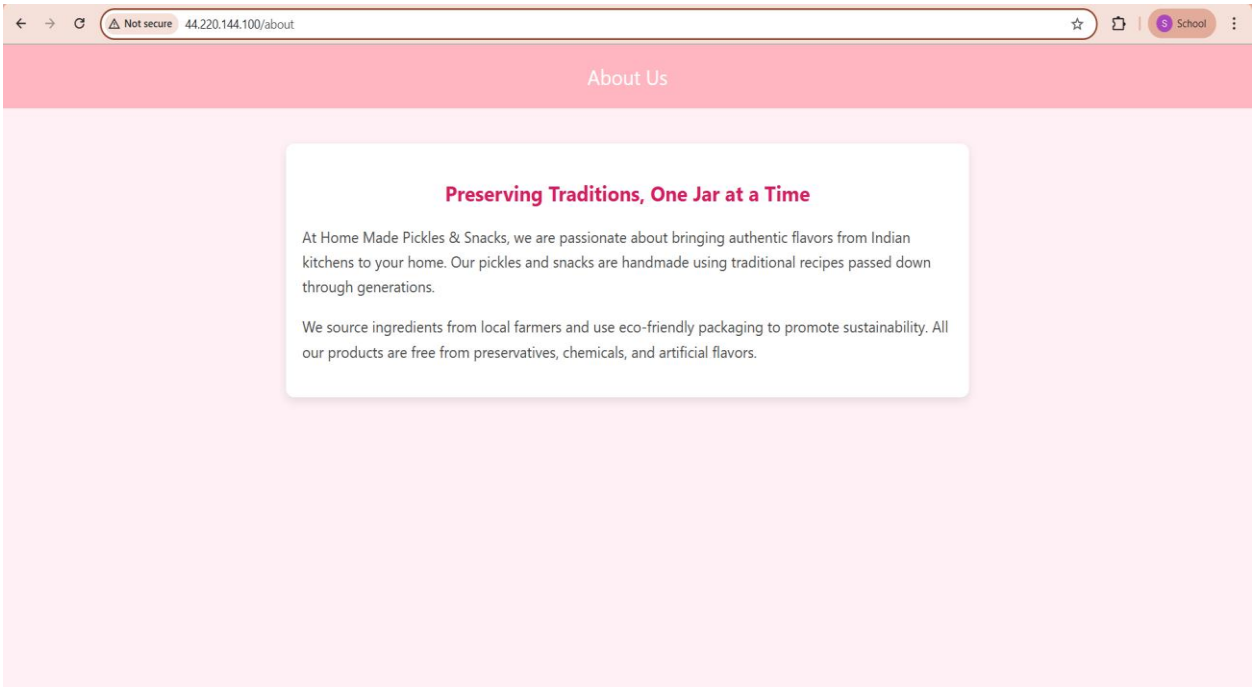


SUCCESS PAGE :

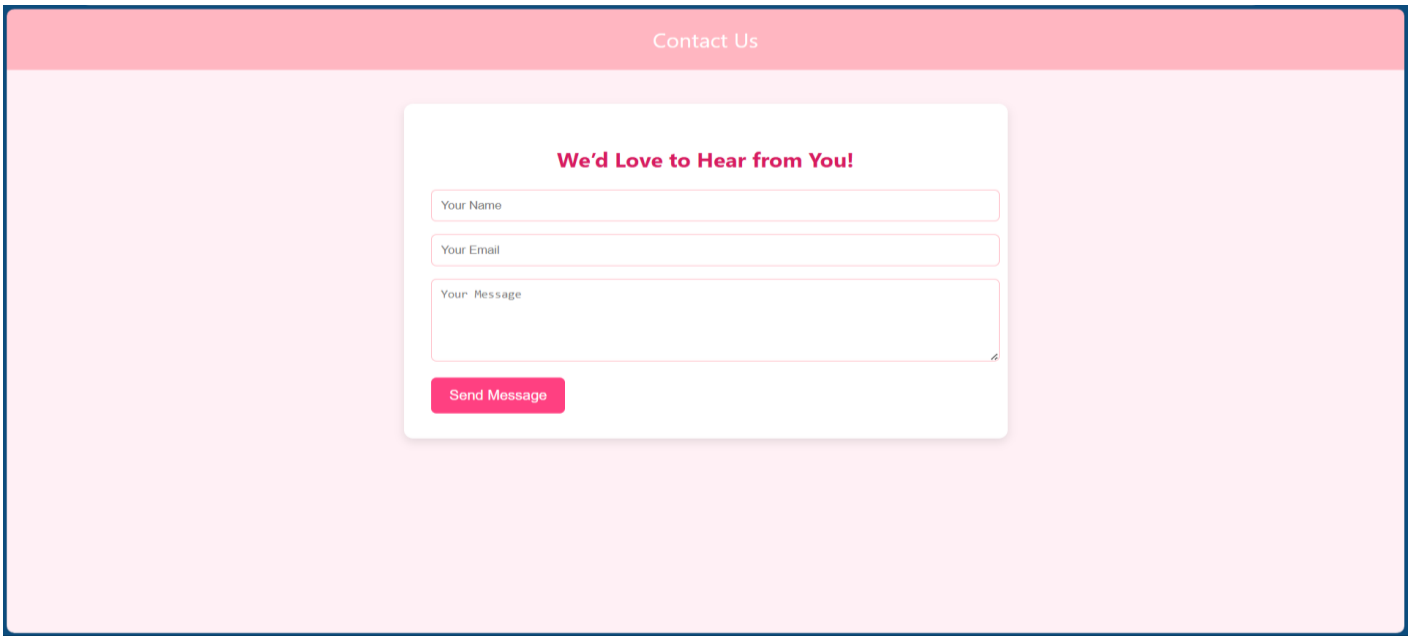




ABOUT PAGE :



CONTACT PAGE :



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.

……..THE END……