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.

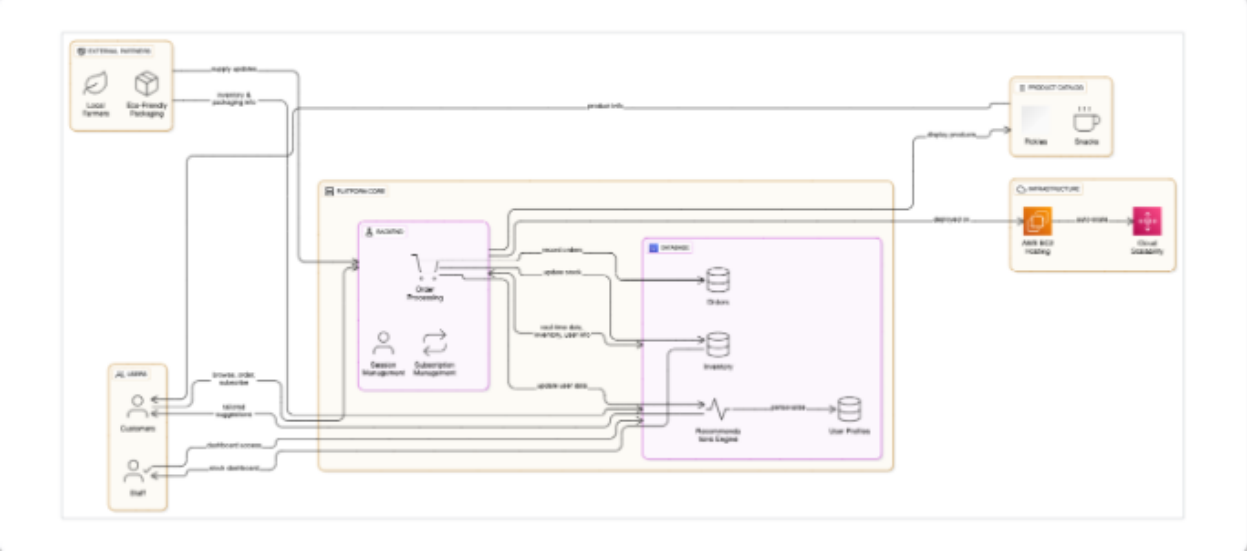
Scenario 3: Personalized User Experience and Recommendations

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

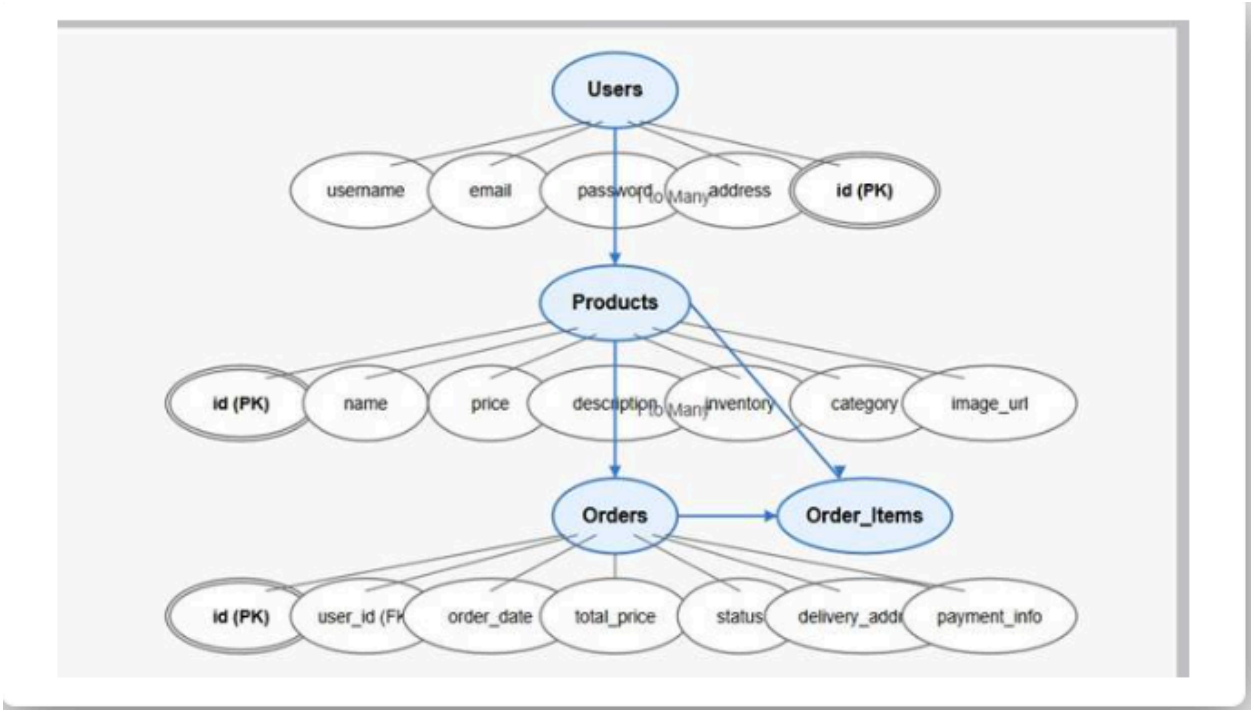
This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for 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.

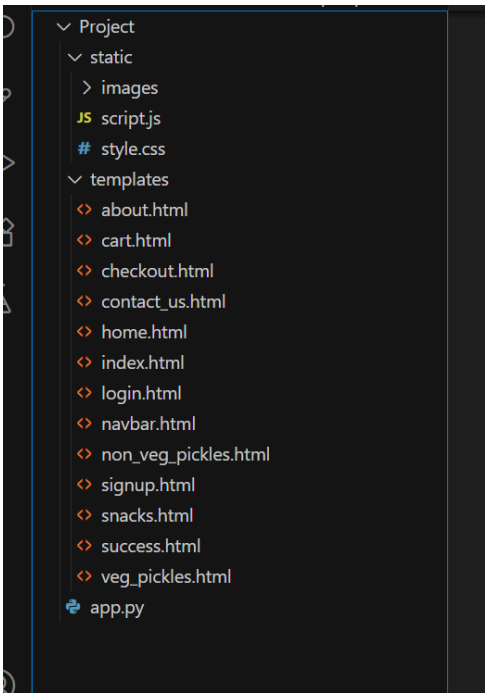
Milestone 1 : Web Application Development and

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.

```
# AWS Configuration
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
user_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

- Routes for Web Pages

```
# Email Configuration
EMAIL_ADDRESS = 'susmithaghantasala7@gmail.com'
EMAIL_PASSWORD = 'hvop ohwp vtku ihuu'



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

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

@app.route('/veg-pickles')
def veg_pickles():
    return render_template('veg_pickles.html')

@app.route('/non-veg-pickles')
def non_veg_pickles():
    return render_template('non_veg_pickles.html')

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

@app.route('/cart')
```

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

```python
@app.route('/success', methods=['GET', 'POST'])
def success():
    return render_template('success.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password'].encode('utf-8')

        # Get user from DynamoDB
        response = user_table.get_item(Key={'username': username})
        user = response.get('Item')

        if user:
            hashed_password = user['password'].value if hasattr(user['password'], 'value') else user[
            if bcrypt.checkpw(password, hashed_password.encode('utf-8')):
                session['username'] = username
                flash('Login successful!', 'success')
                return redirect(url_for('home'))
            else:
                flash('Invalid password', 'danger')
        else:
            flash('User not found', 'danger')

    return render_template('login.html')

```

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

```python


@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password'].encode('utf-8')

        # Check if user already exists
        response = user_table.get_item(Key={'username': username})
        if 'Item' in response:
            flash('Username already exists!', 'danger')
            return render_template('signup.html')

        # Hash the password
        hashed = bcrypt.hashpw(password, bcrypt.gensalt())

        # Save user to DynamoDB
        user_table.put_item(
            Item={
                'username': username,
                'email': email,
                'password': hashed.decode('utf-8')  # Store as string
            }
        )

        # Send welcome email
        send_email(email, 'Welcome to Home made Pickles!', 'Thank you for signing up!')
```

```python

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

@app.route('/contact_us', methods=['GET', 'POST'])
def contact_us():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        message_content = request.form['message']

        # Combine the message
        body = f"Name: {name}\nEmail: {email}\nMessage:\n{message_content}"

        # Send email to yourself
        send_email(EMAIL_ADDRESS, 'New Contact Us Message', body)

        flash('Your message has been sent! Thank you for contacting us.', 'success')
        return redirect(url_for('contact_us'))

    return render_template('contact_us.html')


# Email Sending Function
def send_email(to_email, subject, body):
    try:
```

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

        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
        server.send_message(msg)
        server.quit()
    except Exception as e:
        print(f"Failed to send email: {e}")

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

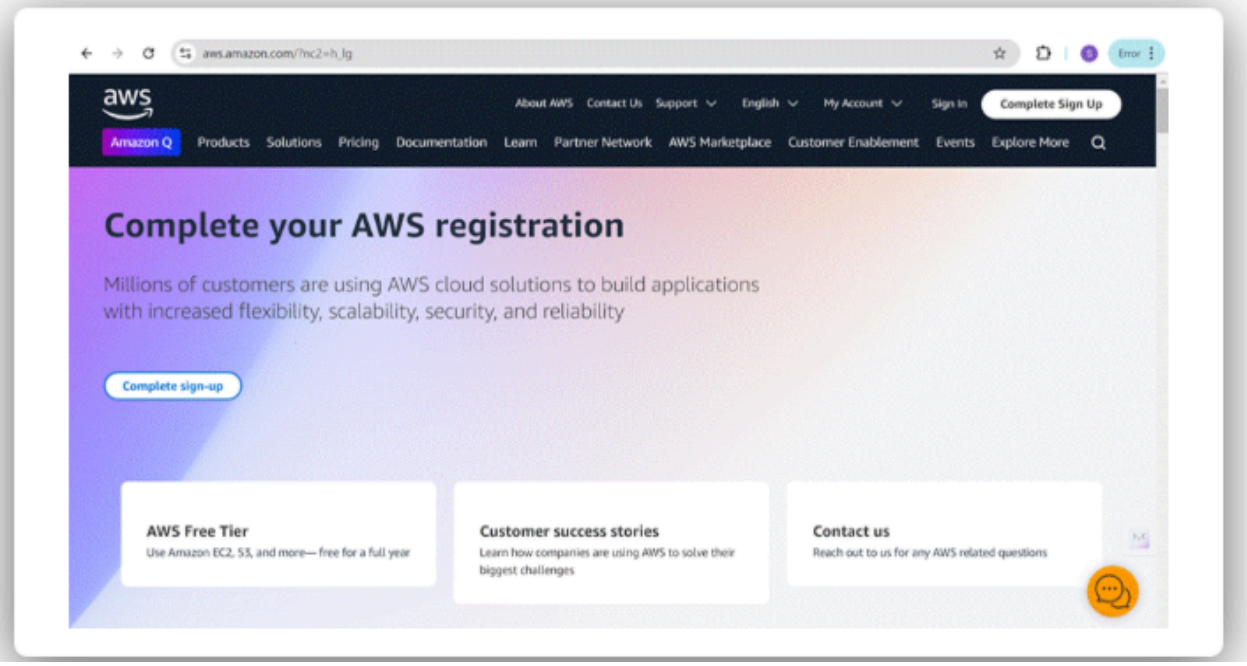
Milestone 2 : AWS Account Setup

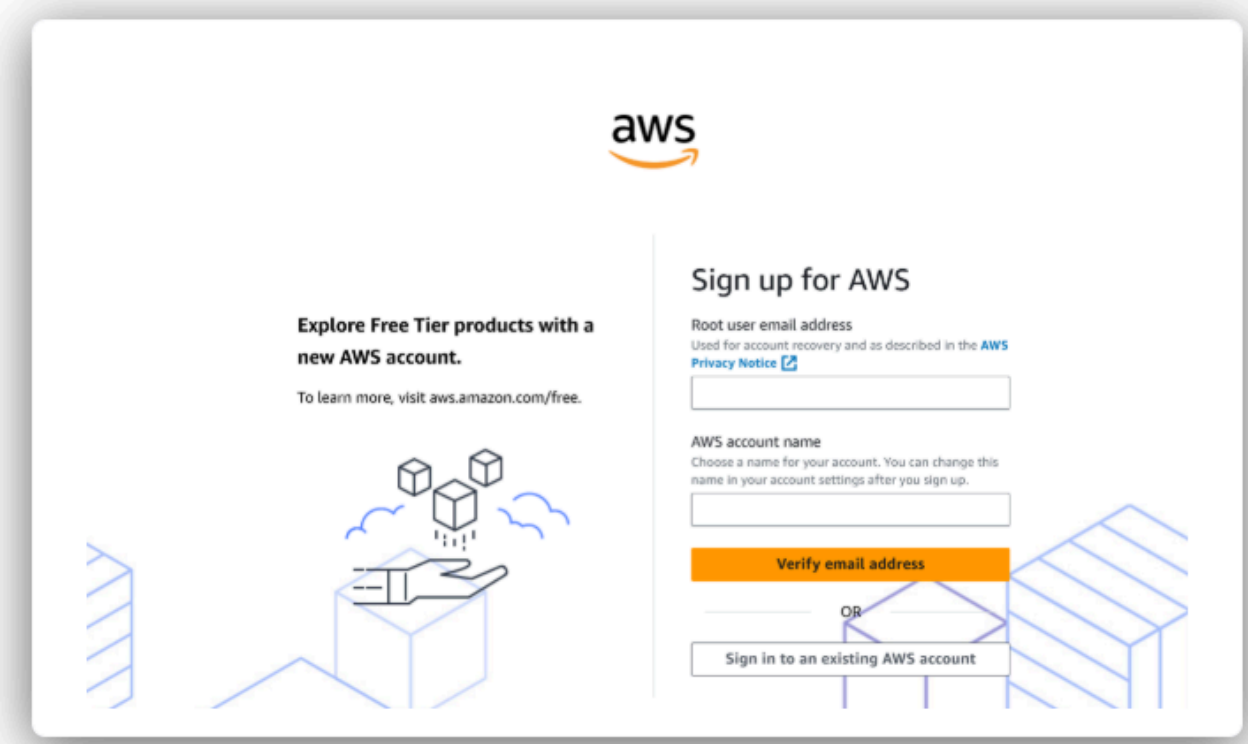
Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

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.

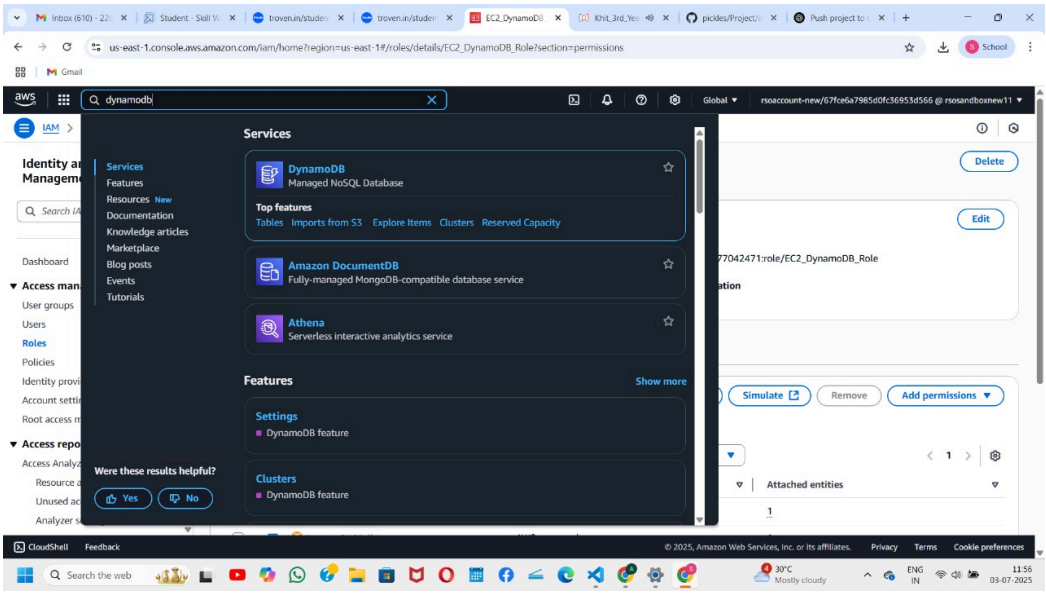
- Go to the AWS website (https://aws.amazon.com/).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: 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.

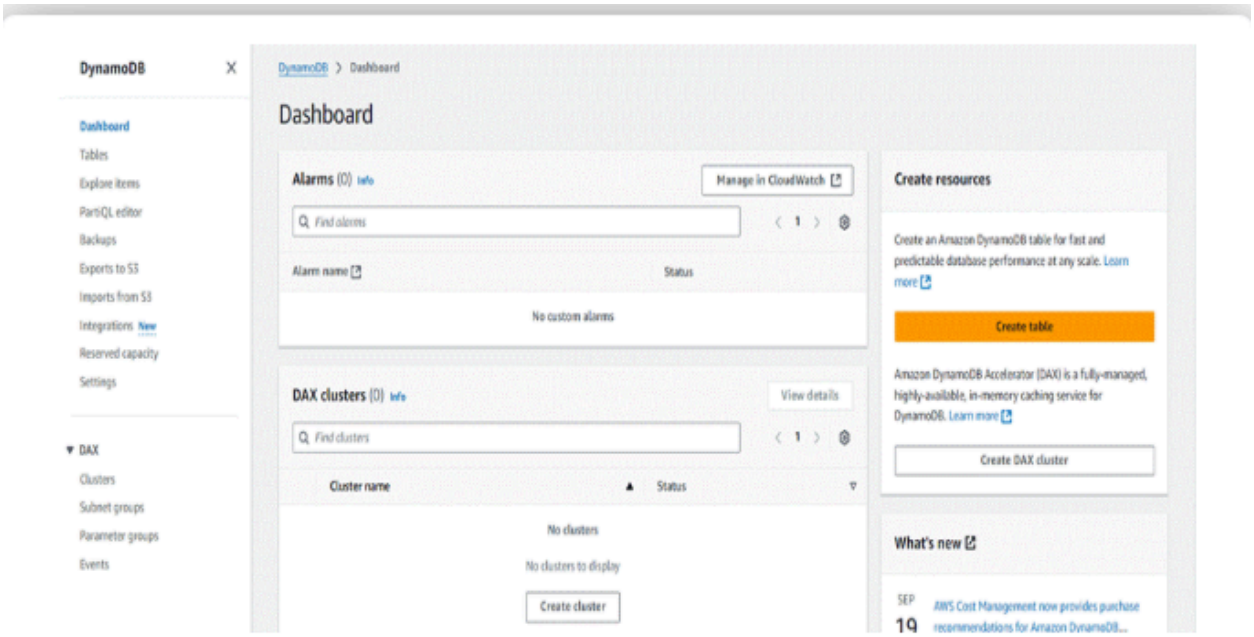




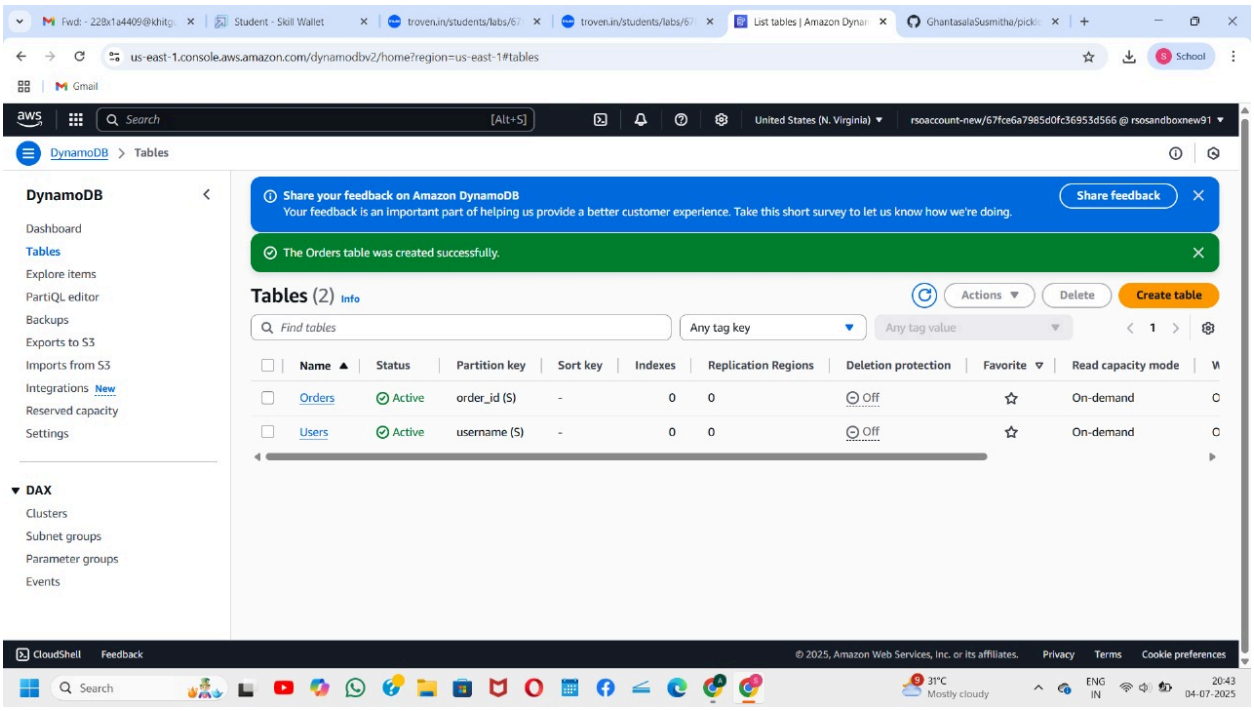
Milestone 3 : DynamoDB Database Creation and Setup

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.


dynamodb
Services
DynamoDB
Managed NoSQL Database
Top features
Tables Imports from S3 Explore Items Clusters Reserved Capacity
Amazon DocumentDB
Fully-managed MongoDB-compatible database service
Athena
Serverless interactive analytics service
Features
Show more
Settings
DynamoDB feature
Clusters
DynamoDB feature
Were these results helpful?
Yes
No
Delete
Edit
Simulate
Remove
Add permissions
Attached entities
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


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
DynamoDB > Dashboard
Dashboard
Alarms (0) Info
Manage in CloudWatch
Find alarms
Alarm name
Status
No custom alarms
DAX clusters (0) Info
View details
Find clusters
Cluster name
Status
No clusters
No clusters to display
Create cluster
Create resources
Create an Amazon DynamoDB table for fast and predictable database performance at any scale. Learn more
Create table
Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more
Create DAX cluster
What's new
SEP 19
AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...


DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
DynamoDB > Tables
Tables (0) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Deletion protection
Read capacity mode
Write capacity mode
Total size
You have no tables in this account in this AWS Region.
Create table

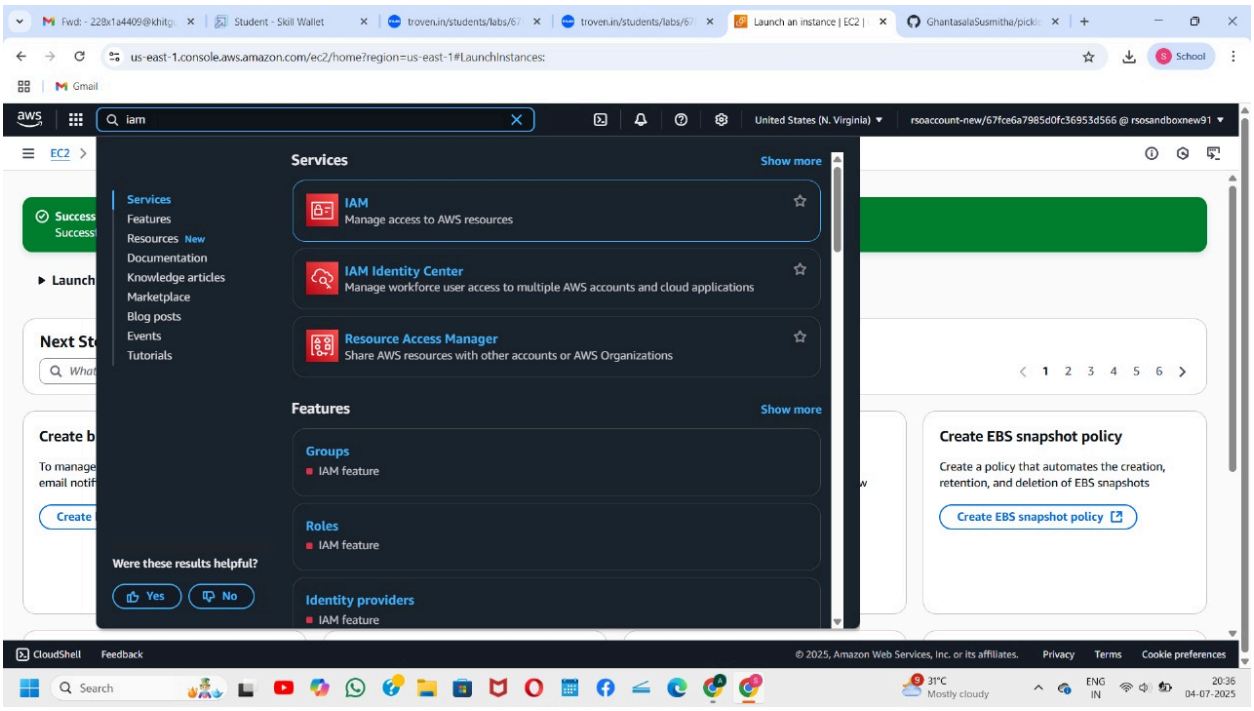


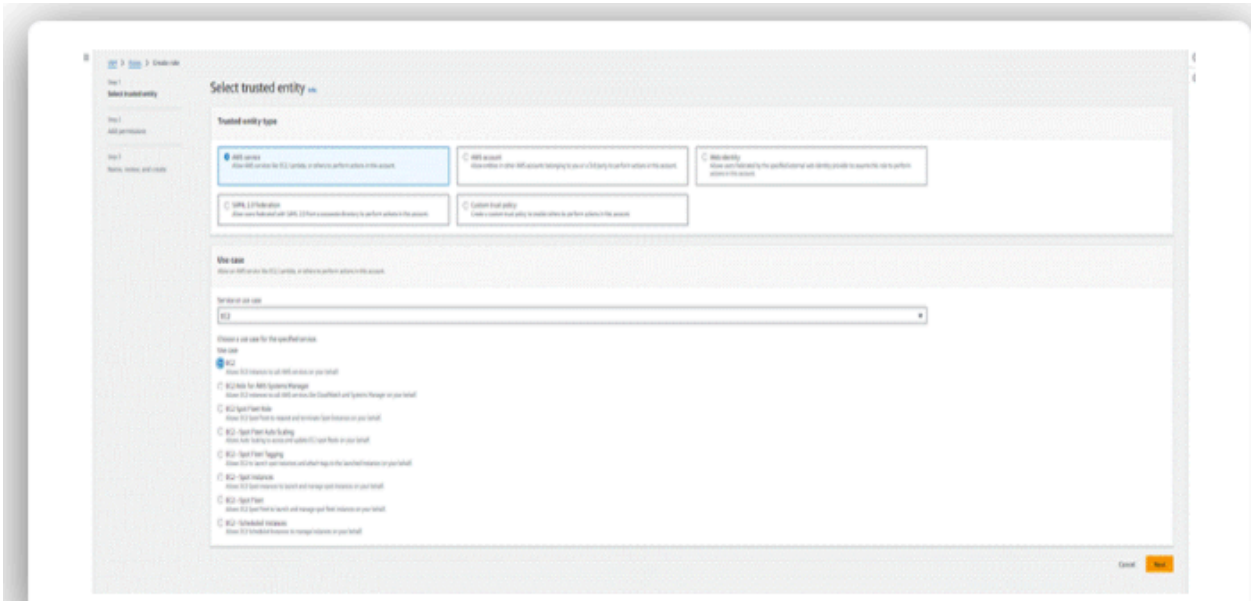
Milestone 4 : IAM Role Setup

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.

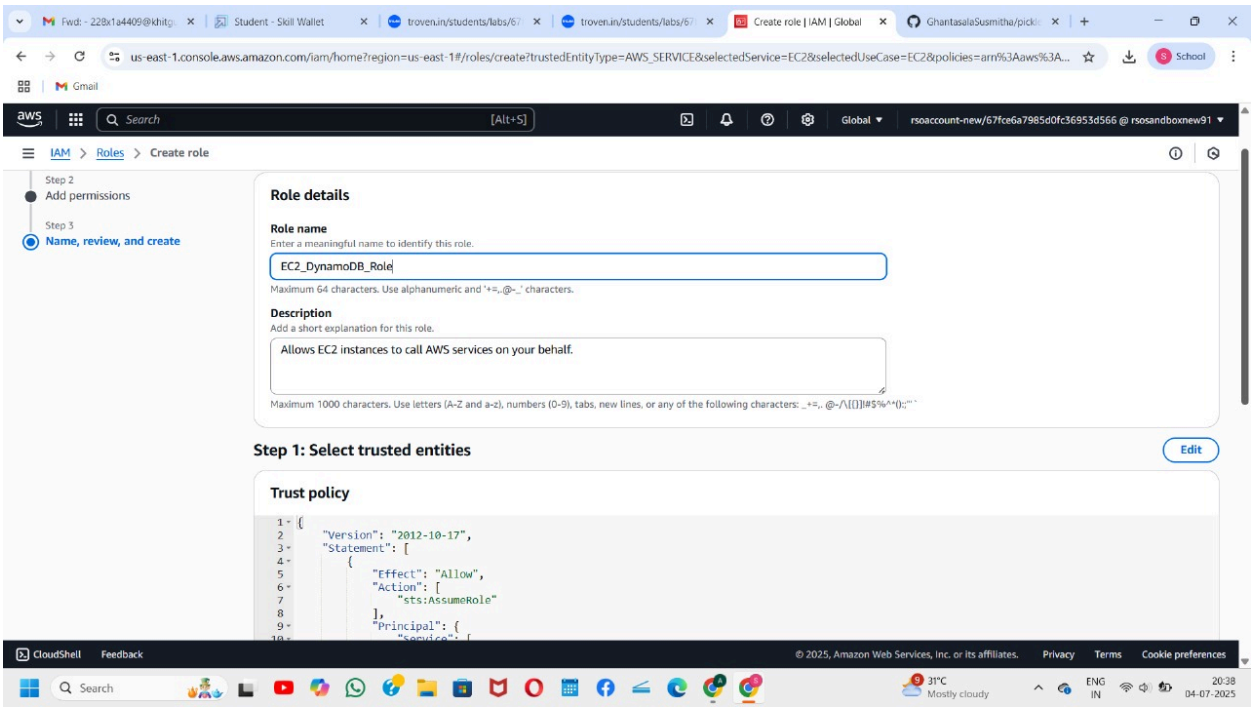


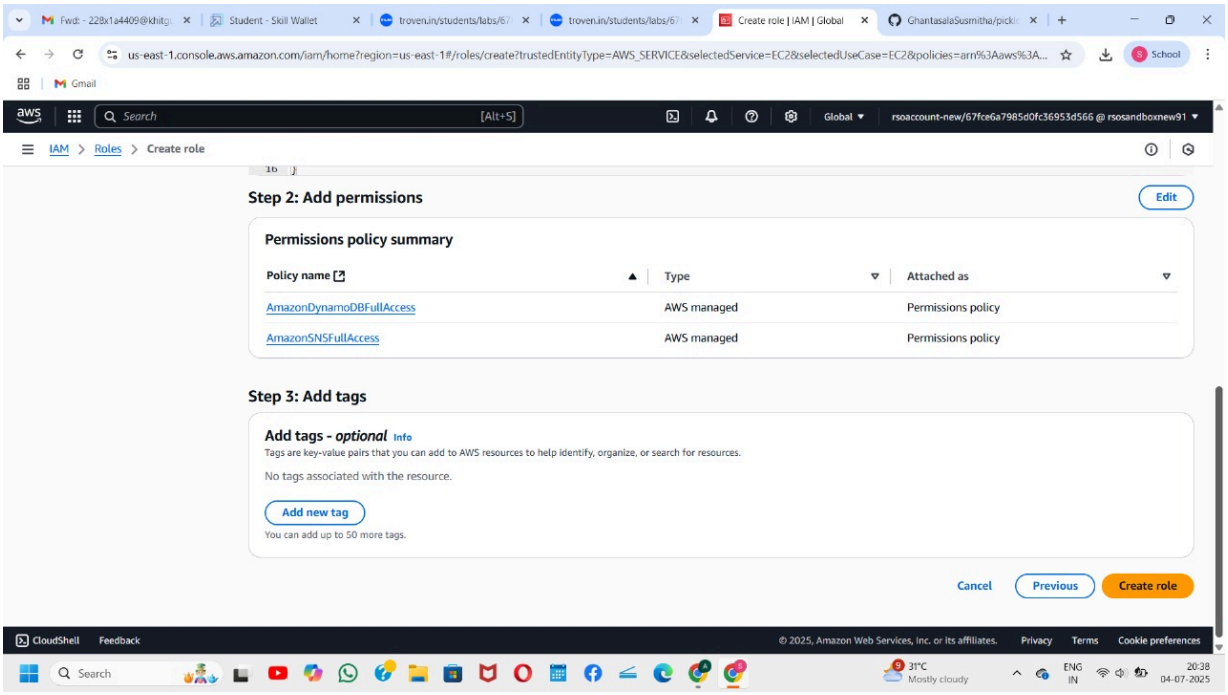


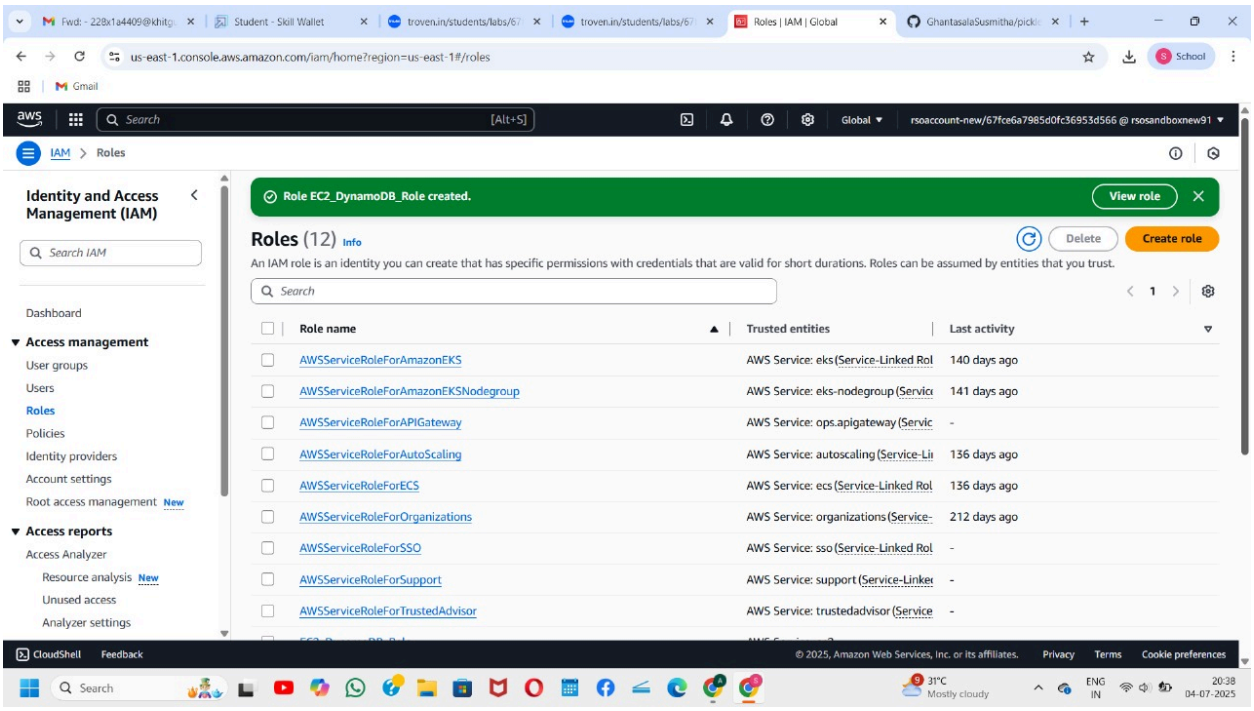
Attach Policies

Attach the following policies to the role:

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


Create role | IAM | Global
us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&selectedUseCase=EC2&policies=arn%3Aaws%3A...
Search [Alt+S]
Global
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
IAM > Roles > Create role
Step 2
Add permissions
Step 3
Name, review, and create
Role details
Role name
Enter a meaningful name to identify this role.
EC2_DynamoDB_Role
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.
Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`
Step 1: Select trusted entities
Edit
Trust policy
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"sts:AssumeRole"
],
"Principal": {
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
31°C
Mostly cloudy
ENG
IN
20:38
04-07-2025


Create role | IAM | Global
us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&selectedUseCase=EC2&policies=arn%3Aaws%3A...
Search [Alt+S]
Global
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
IAM > Roles > Create role
Step 2: Add permissions
Edit
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 Info
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
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
31°C
Mostly cloudy
ENG
IN
20:38
04-07-2025

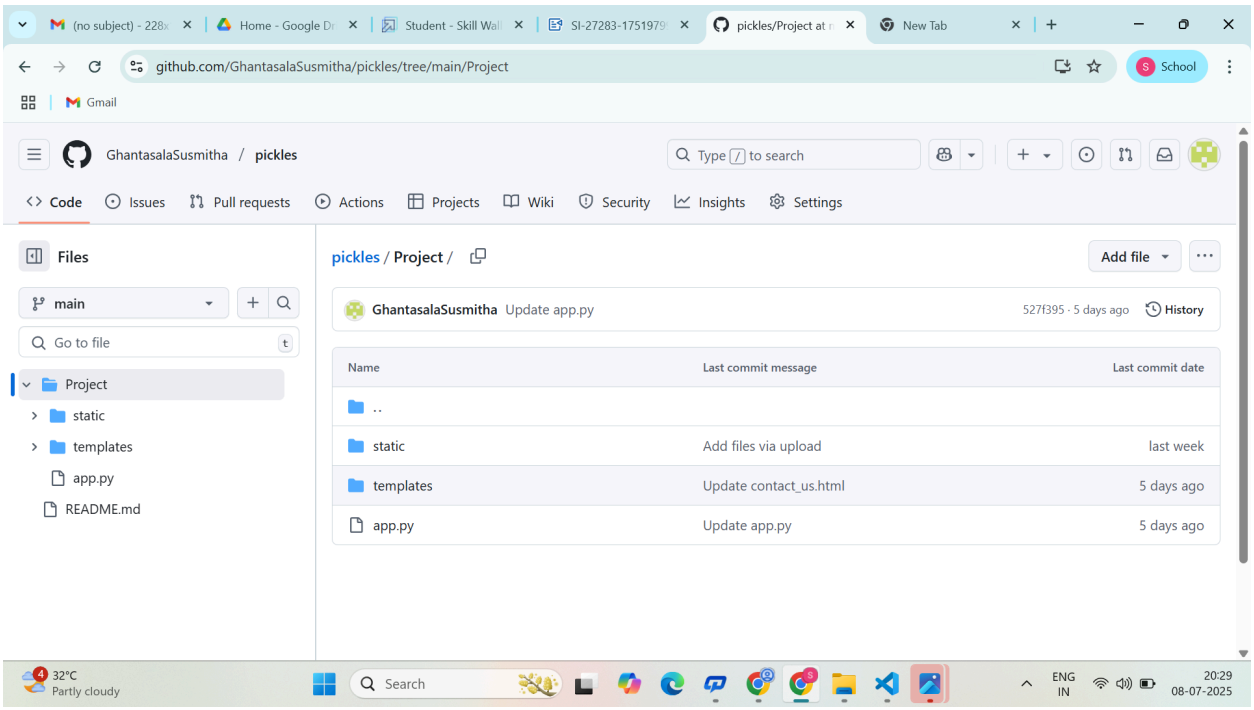


Milestone 5 : EC2 Instance Setup

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.

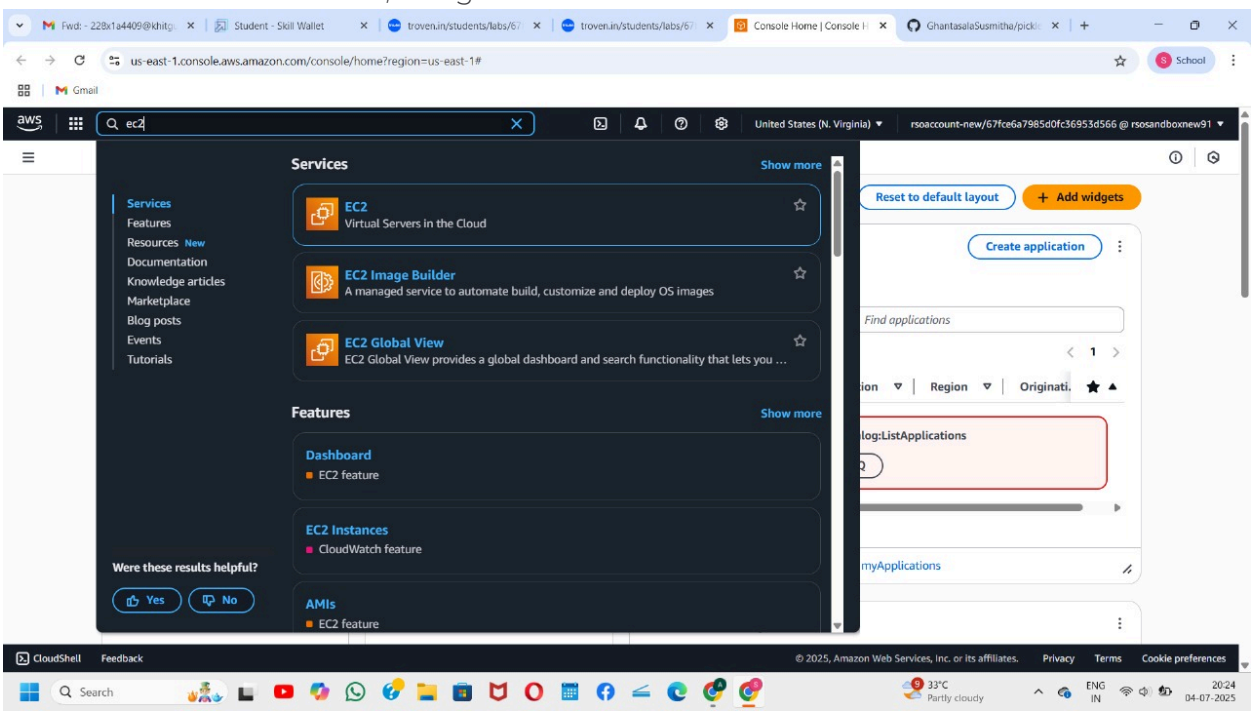
Load your Project Files to GitHub

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

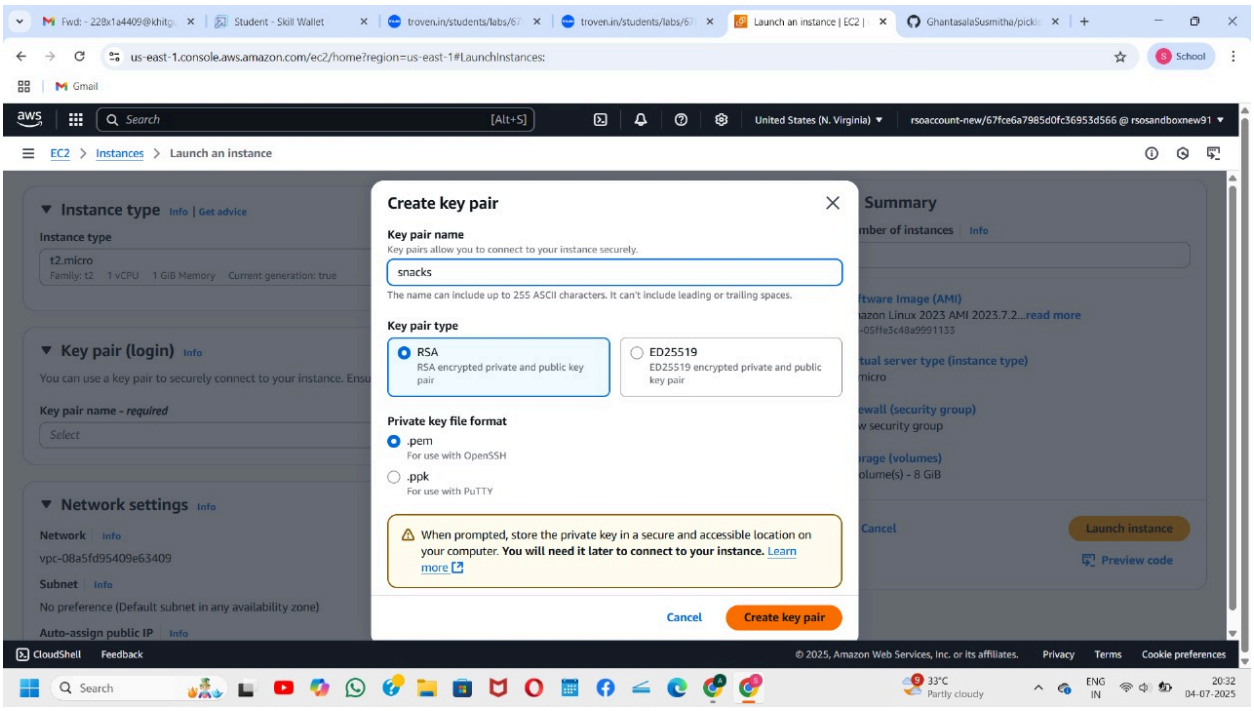


Launch an EC2 instance

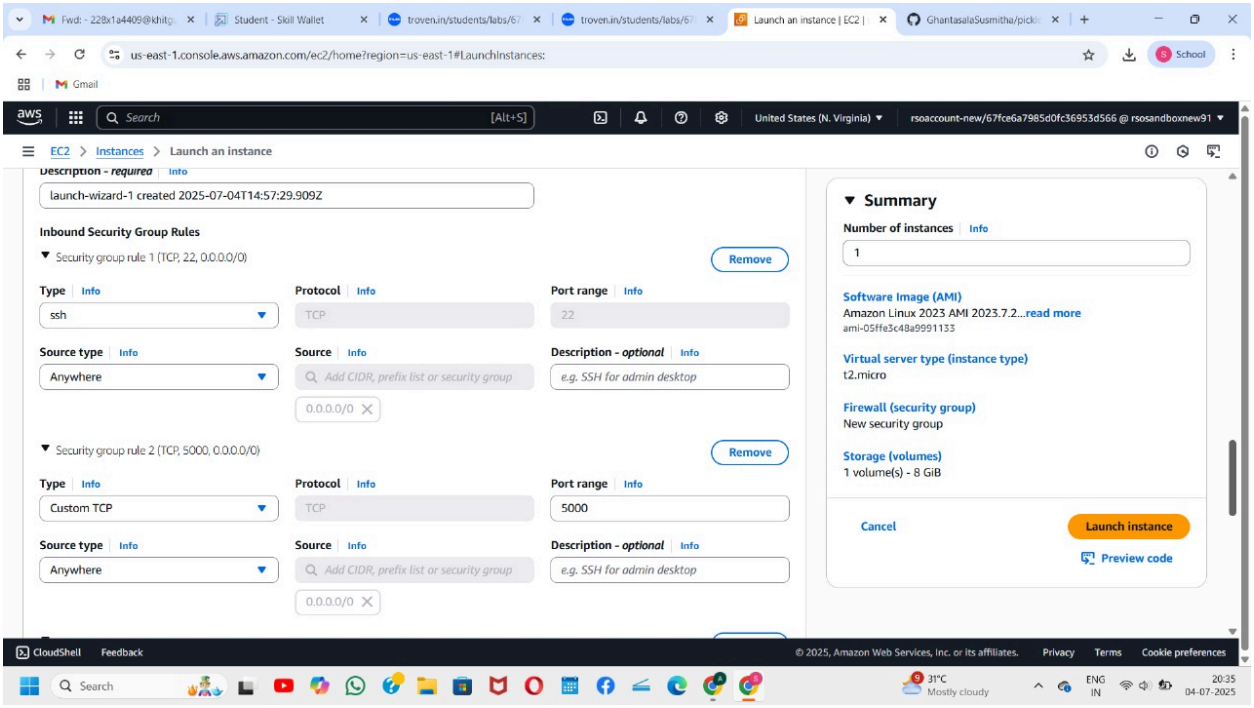
- In the AWS Console, navigate to EC2 and launch a new instance.

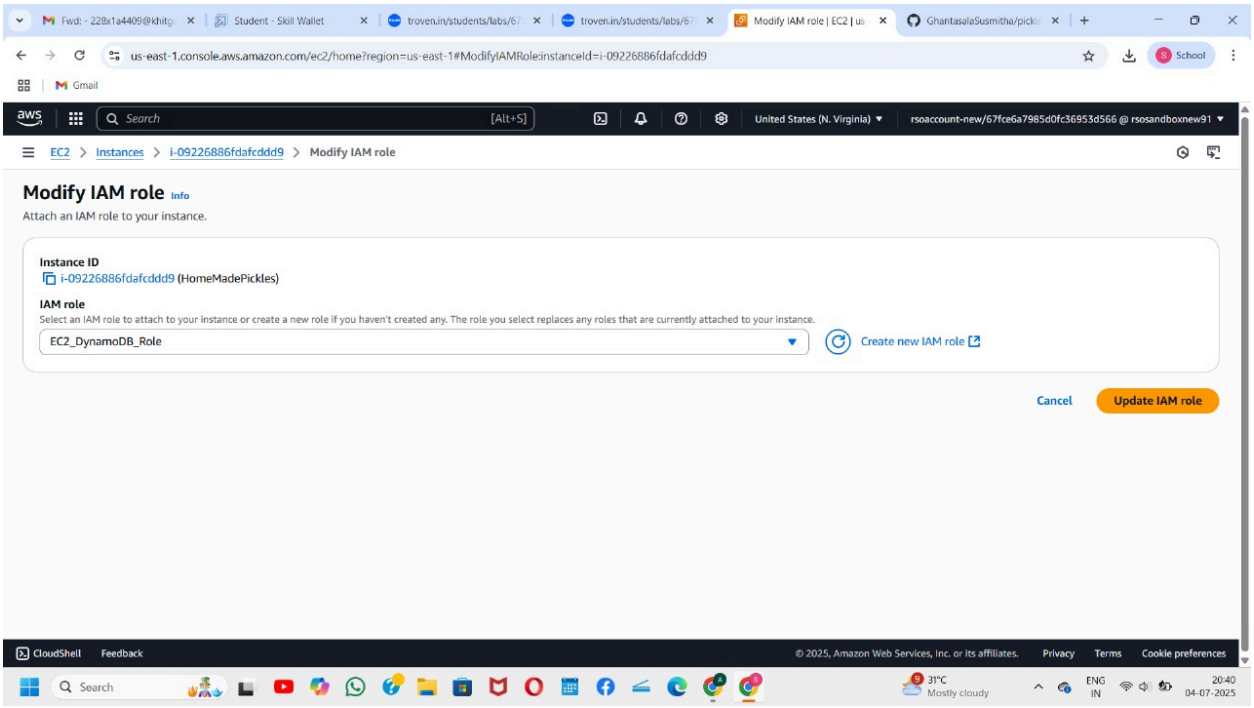


- Click on Launch instance to launch EC2 instance

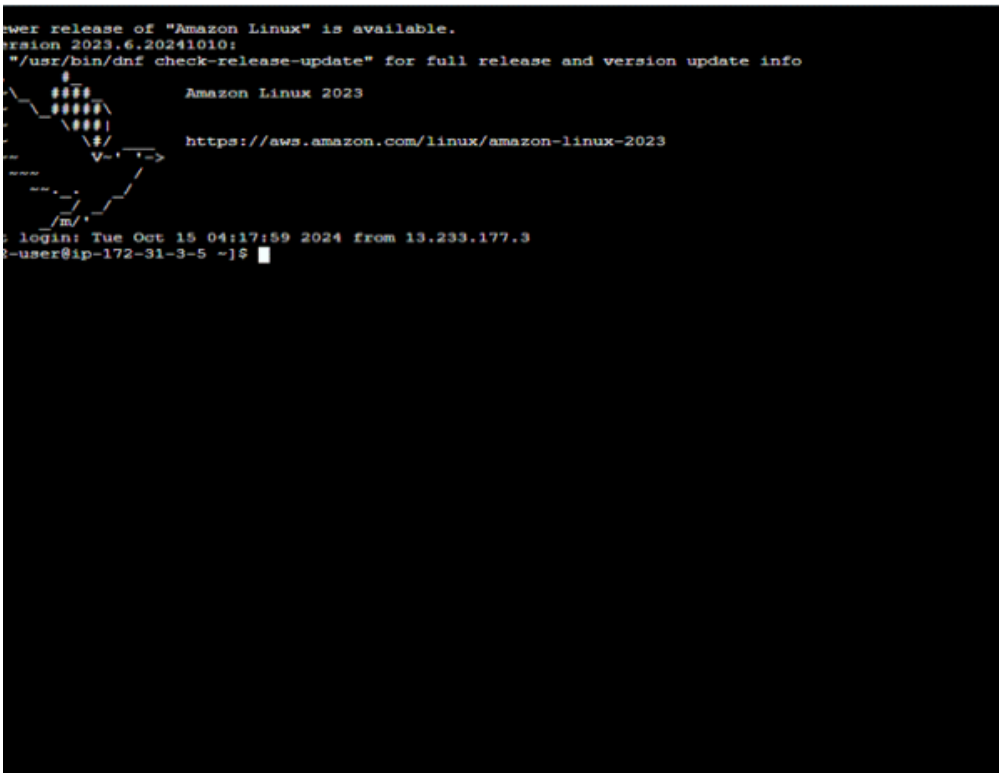
- 

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



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.

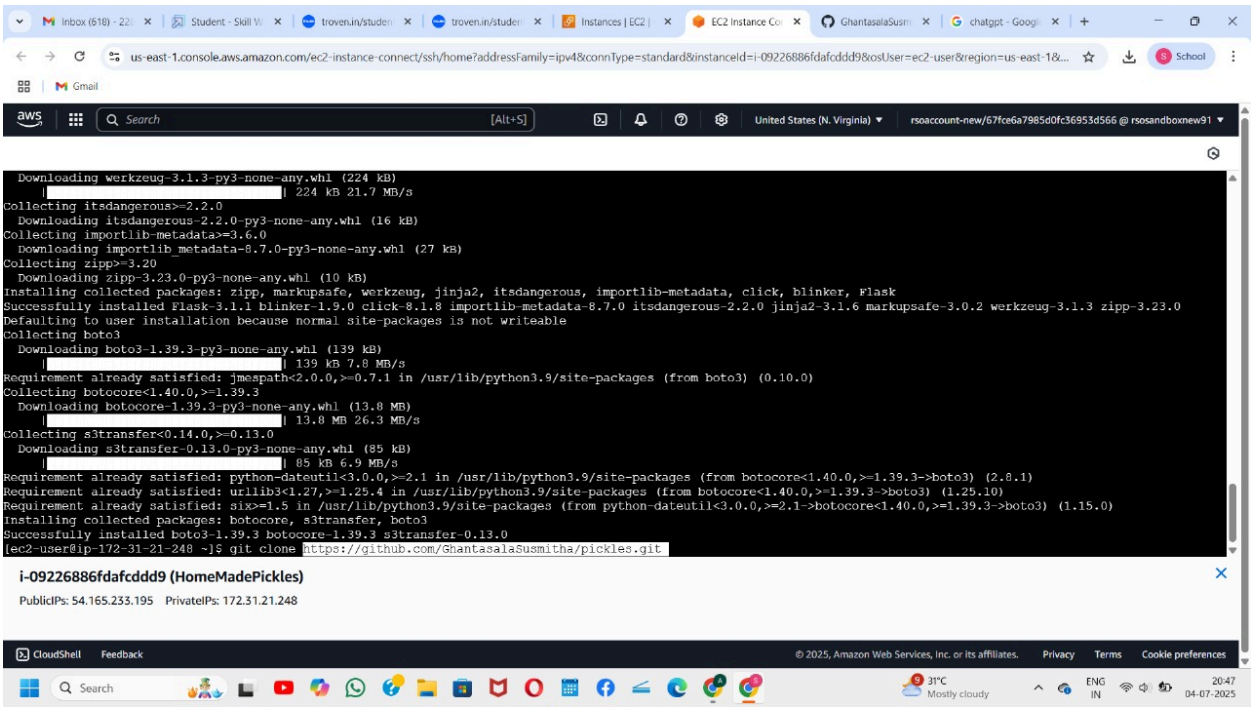
- Run: 'git clone:https://github.com/GhantasalaSusmitha/pickles.git

Create a Virtual Environment:

- python3 -m venv venv
- source venv/bin/activate
- sudo yum install python3 git
- sudo pip3 install flask boto3

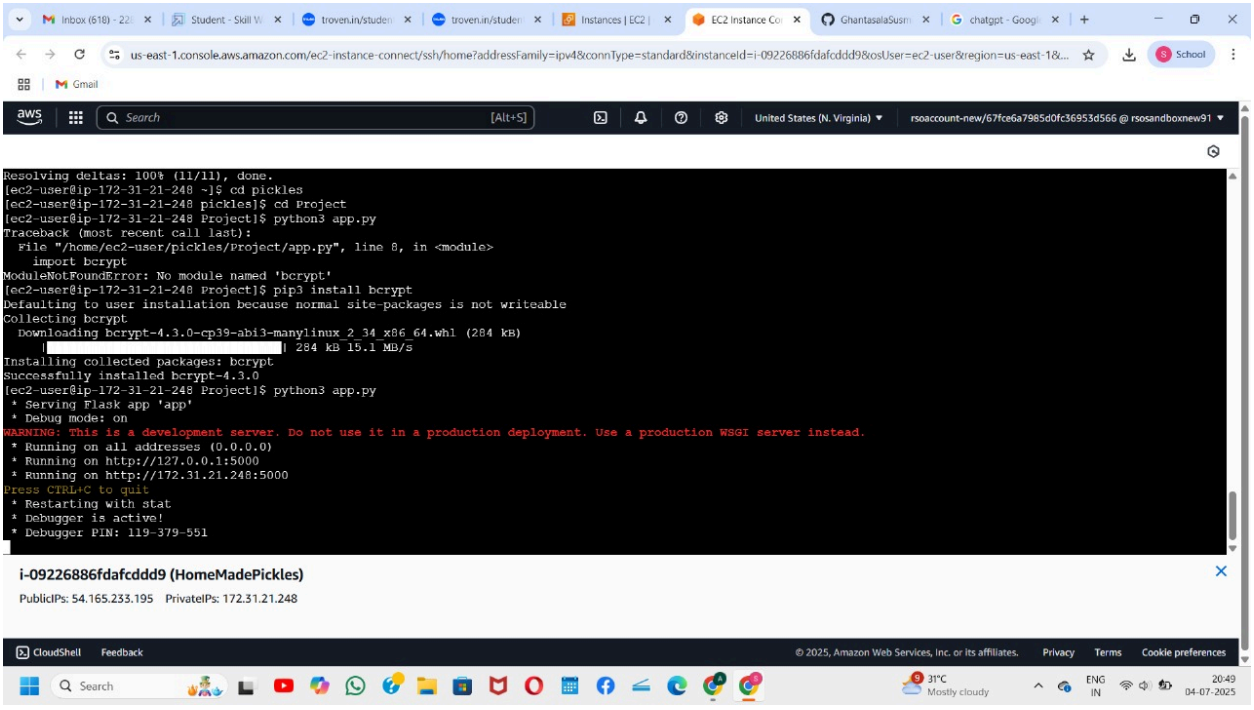
Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

- 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
- 

Access the website through:

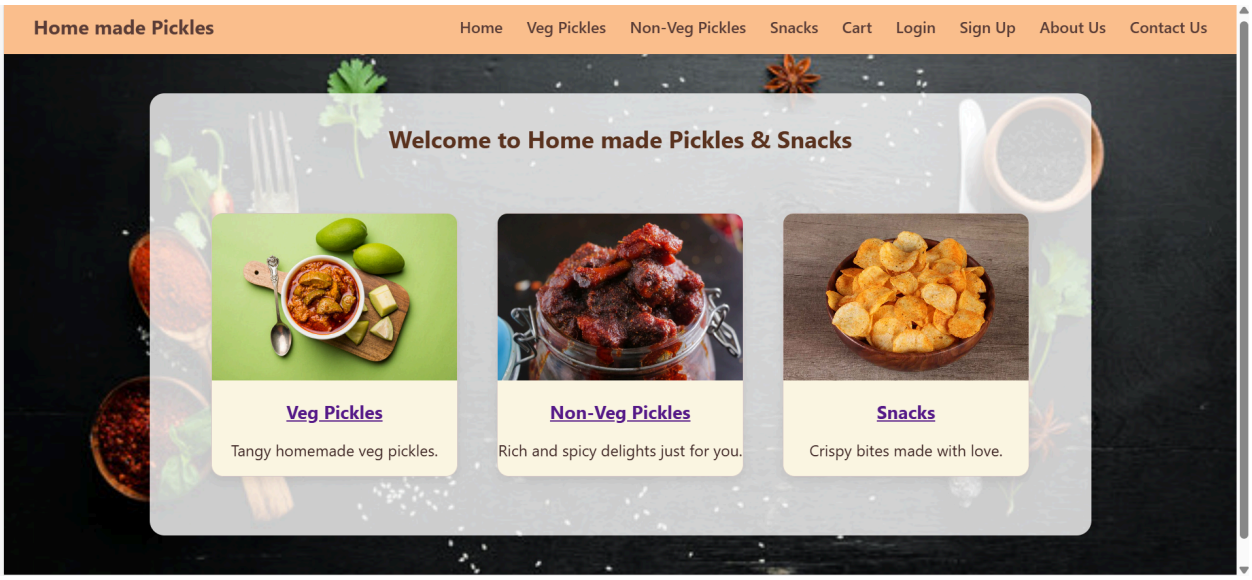
PublicIPs: http://54.165.233.195:5000/

Milestone 7 : Testing and Deployment

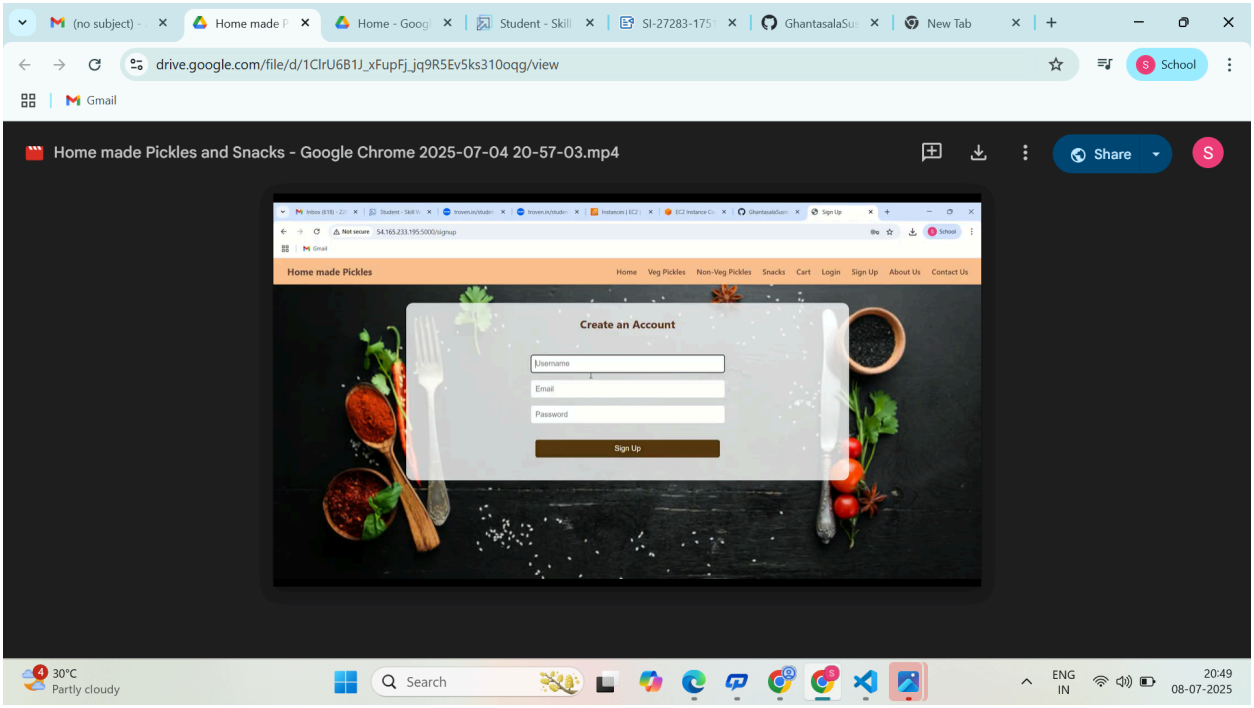
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and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

Functional testing to verify the Project

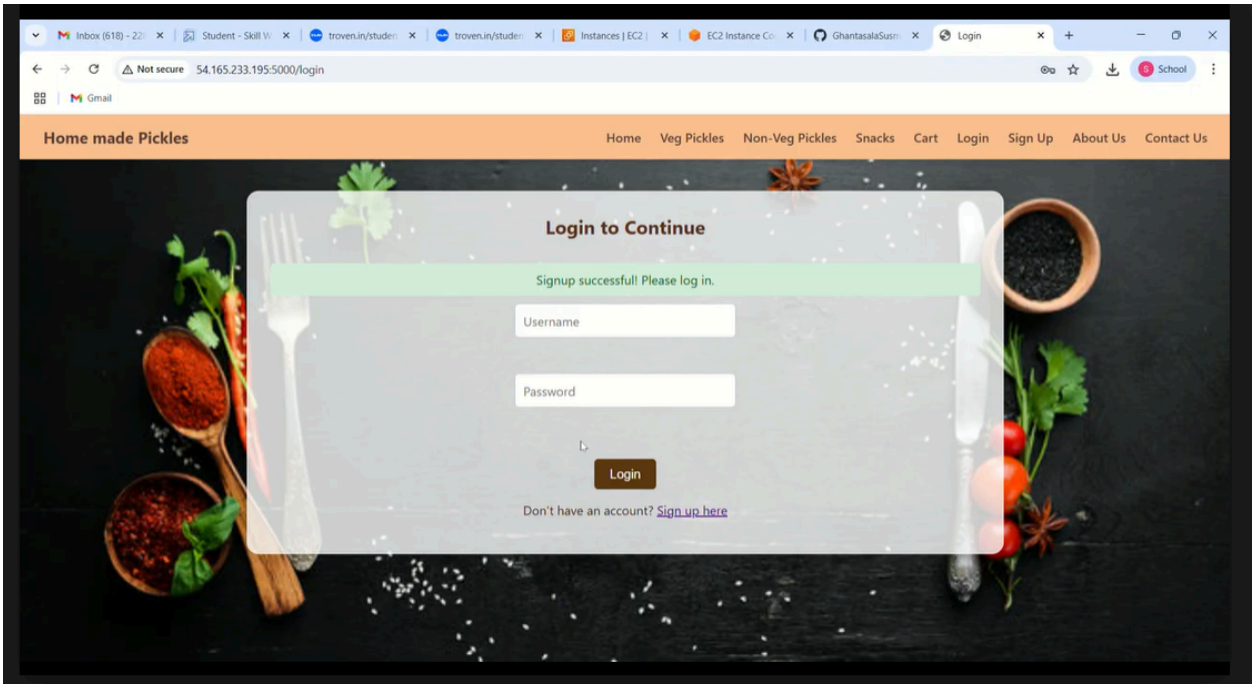
Welcome page:



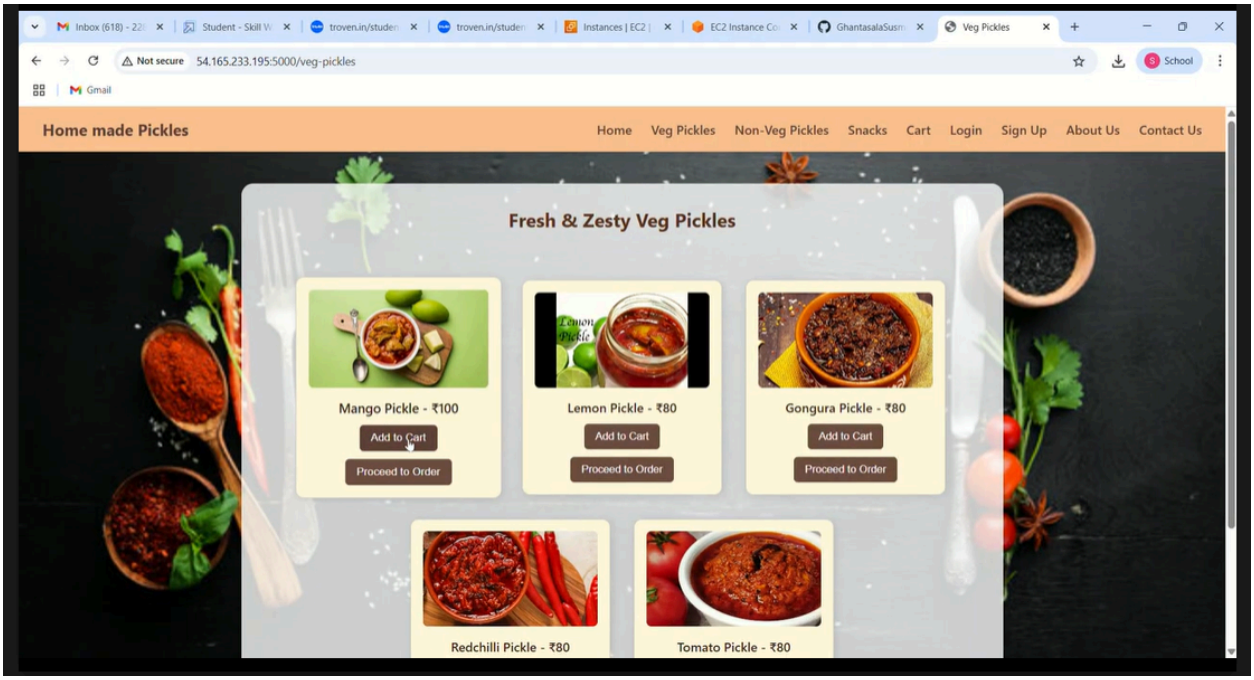
Sign up page:



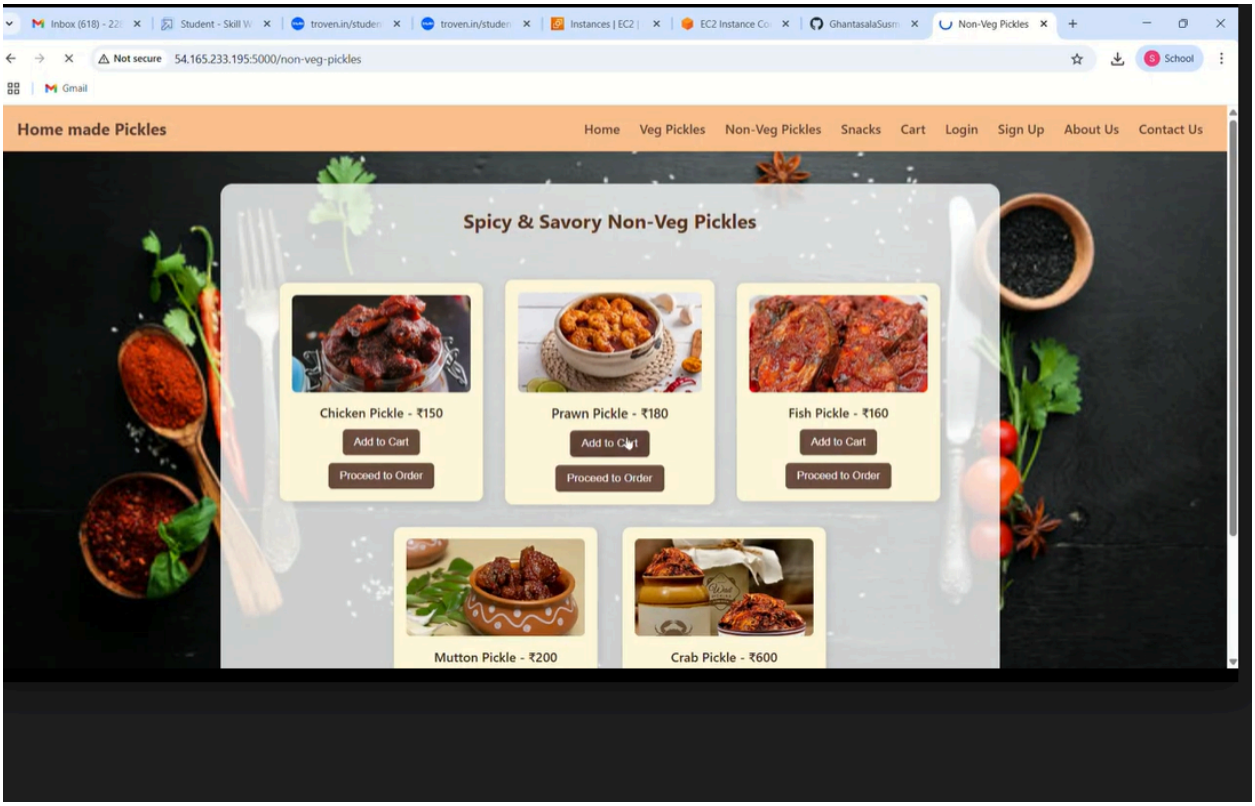
Login page:



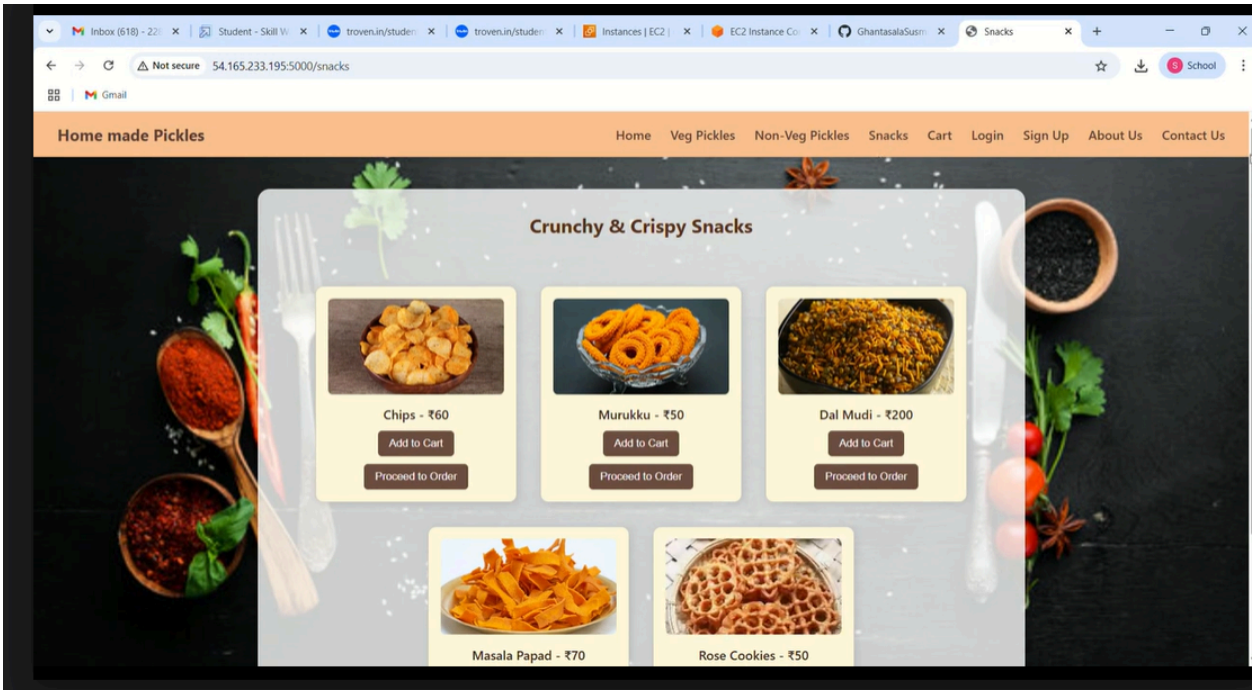
Veg Pickles:



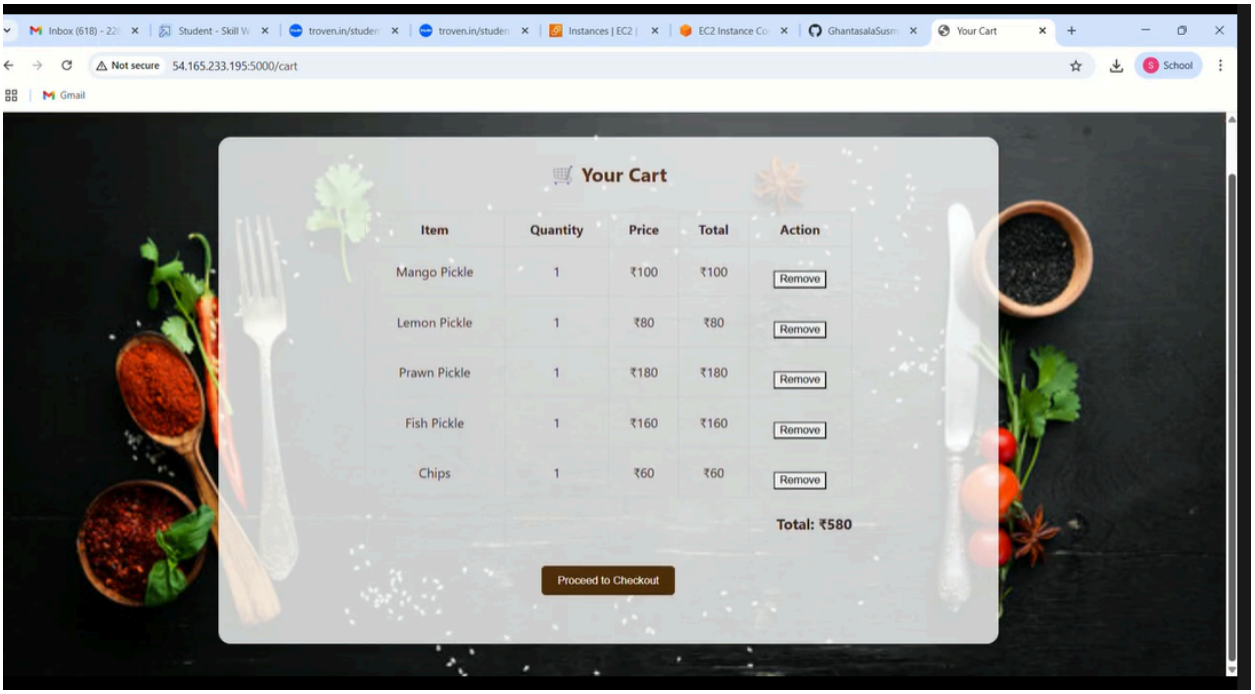
Non-veg pickles:



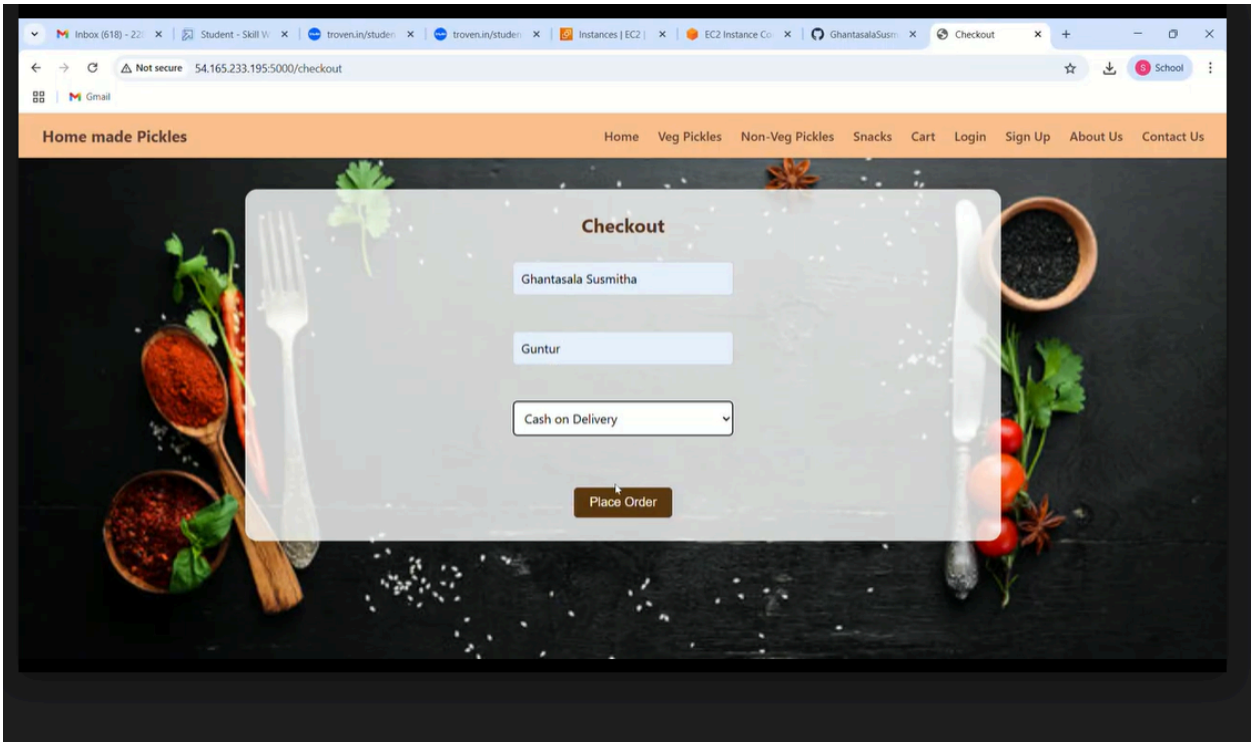
Snacks:



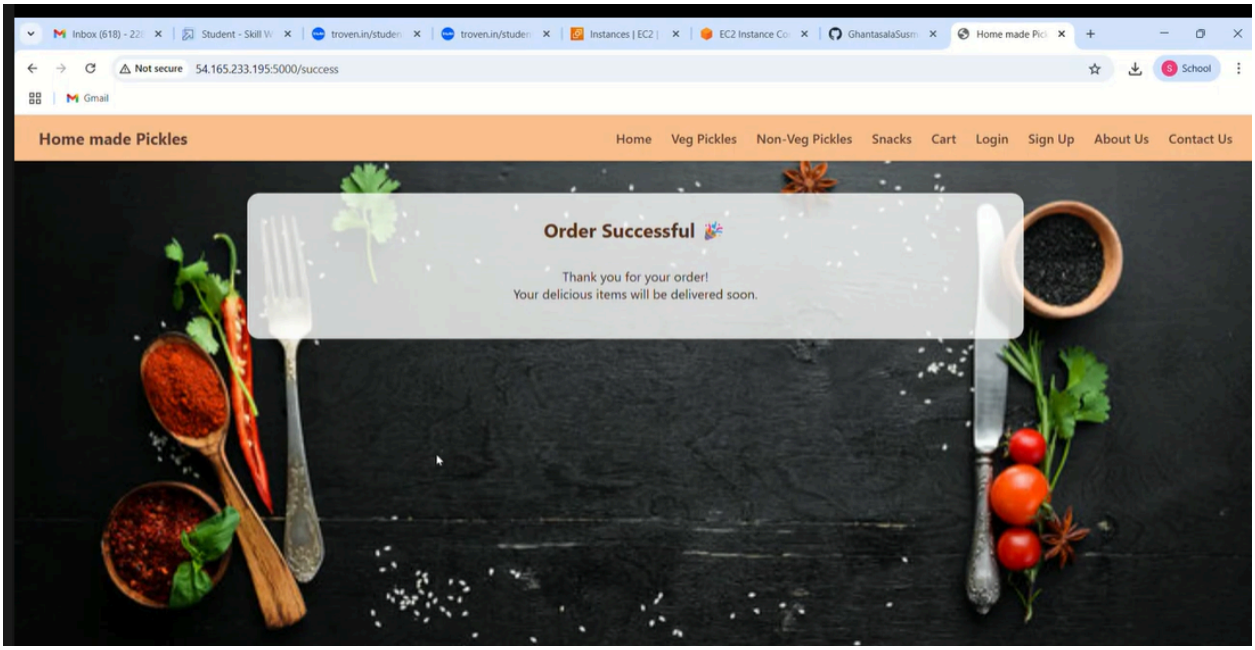
Cart page:



Checkout Page:

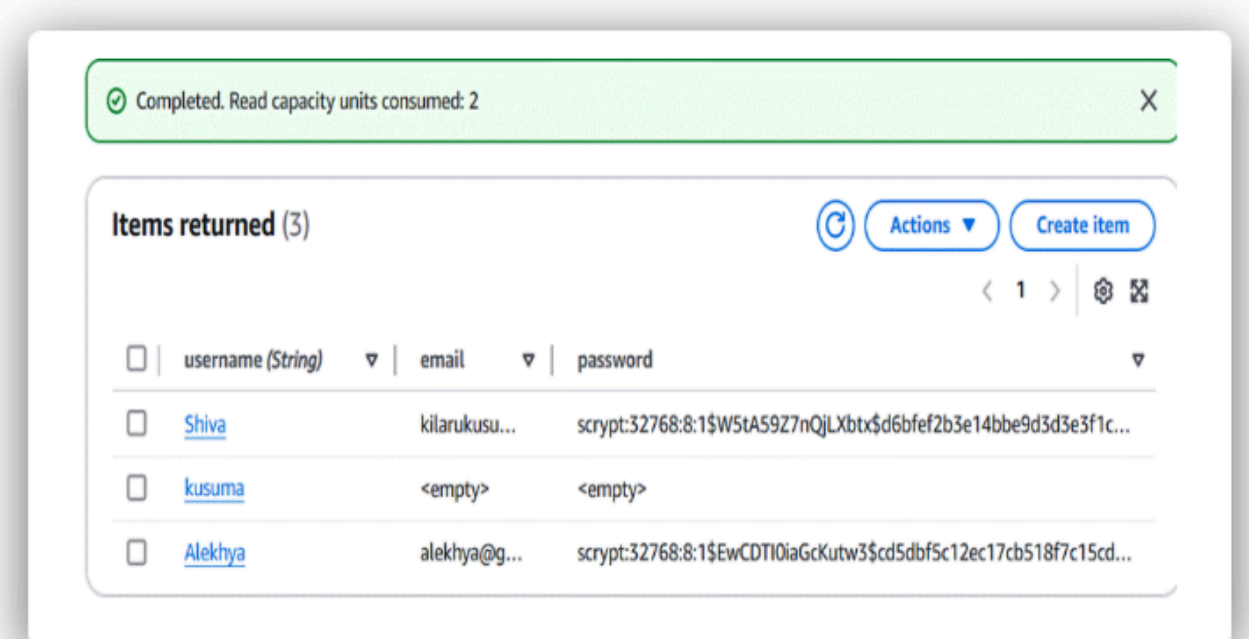


Success Page:



Dynamodb Database updations :

1. Users table :



2. Orders table :