

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

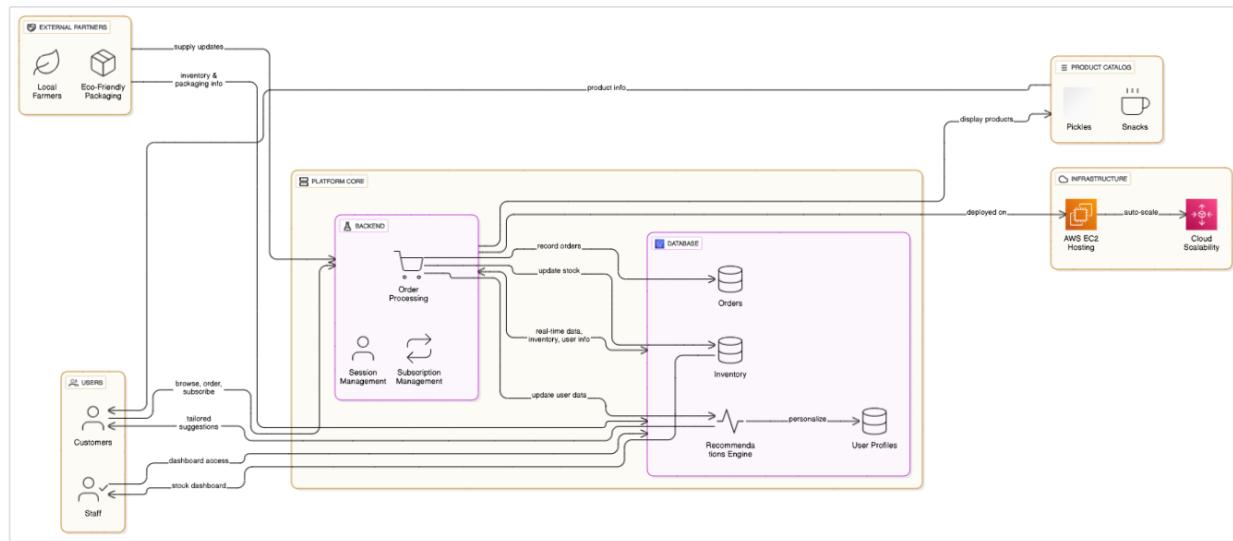
When a customer places an order for a product, the system instantly updates stock levels and records transaction details. For example, a user purchases an item, triggering automatic inventory deduction and order confirmation. Staff members receive updated dashboards to monitor stock availability and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

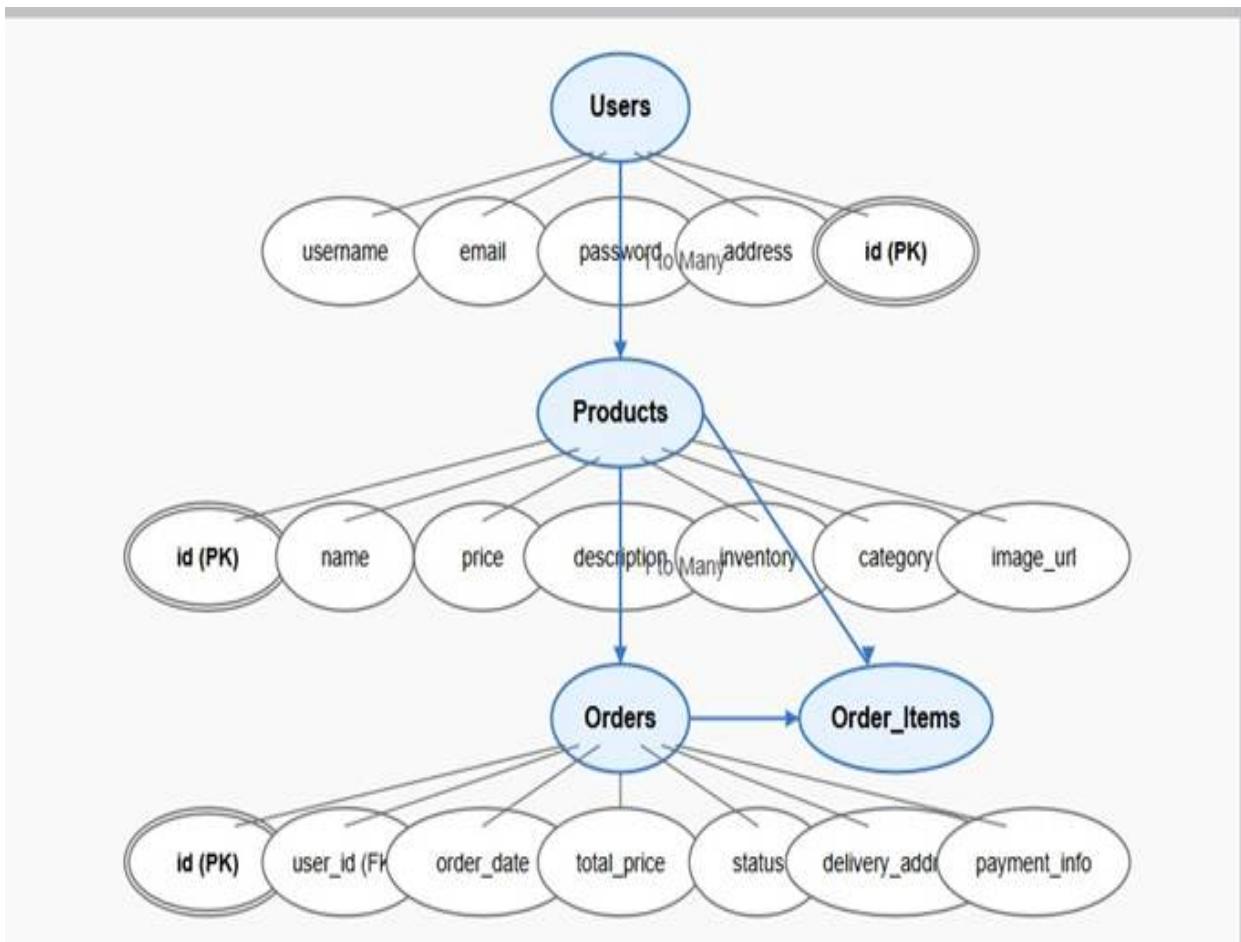
Architecture:

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



Entity Relationship (ER)Diagram:

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



Pre-requisites:

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

Project 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 : Web Application Development and Setup

LOCAL DEPLOYMENT

- File Explorer Structure

```
✓ home made
  ✓ pickle
    ✓ static
      ✓ css
        # animations.css
        # responsive.css
        # styles.css
      ✓ images
        📸 chicken pickle.jpeg
        📸 fish.jpeg
        📸 gifts.jpg
        📸 mango pickle.jpg
        📸 mango.webp
        📸 mutton.webp
        📸 non veg.webp
        📸 nuts.jpg
        📸 papad.webp
        📸 snacks.png
        📸 tomato.jpg
        📸 veg pic.jpg
    ✓ js
      JS cart.js
      JS main.js
      JS products.js
  ✓ template
    <> cart.html
    <> checkout.html
    <> index.html
    <> login.html
    <> product-detail.html
    <> product.html
    <> register.html
  🐍 app.py
```

Description of code:

Imports:

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import json
```

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.

```
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1') # e.g., 'us-east-1'
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table['Orders']
```

```
products = {
    'non_veg_pickles': [
        {'id': 1, 'name': 'Chicken Pickle', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 2, 'name': 'Fish Pickle', 'weights': {'250': 200, '500': 400, '1000': 800}},
        {'id': 3, 'name': 'Gongura Mutton', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 4, 'name': 'Mutton Pickle', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 5, 'name': 'Gongura Prawns', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 6, 'name': 'Chicken Pickle (Gongura)', 'weights': {'250': 350, '500': 700, '1000': 1050}}
    ],
    'veg_pickles': [
        {'id': 7, 'name': 'Traditional Mango Pickle', 'weights': {'250': 150, '500': 280, '1000': 500}},
        {'id': 8, 'name': 'Zesty Lemon Pickle', 'weights': {'250': 120, '500': 220, '1000': 400}},
        {'id': 9, 'name': 'Tomato Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 10, 'name': 'Kakarakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 11, 'name': 'Chintakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 12, 'name': 'Spicy Pandu Mirchi', 'weights': {'250': 130, '500': 240, '1000': 450}}
    ], # Add your veg pickle products here
    'snacks': [
        {'id': 7, 'name': 'Banana Chips', 'weights': {'250': 300, '500': 600, '1000': 800}},
        {'id': 8, 'name': 'Crispy Aam-Papad', 'weights': {'250': 150, '500': 300, '1000': 600}},
        {'id': 9, 'name': 'Crispy Chekka Pakodi', 'weights': {'250': 50, '500': 100, '1000': 200}},
        {'id': 10, 'name': 'Boondhi Acchu', 'weights': {'250': 300, '500': 600, '1000': 900}},
        {'id': 11, 'name': 'Chekkalu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 12, 'name': 'Ragi Laddu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 13, 'name': 'Dry Fruit Laddu', 'weights': {'250': 500, '500': 1000, '1000': 1500}},
        {'id': 14, 'name': 'Kara Boondi', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 15, 'name': 'Gavvalu', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 16, 'name': 'Kaju Chikki', 'weights': {'250': 250, '500': 500, '1000': 750}}
    ]
}
```

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

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

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

        try:
            # Fetch user from DynamoDB
            response = users_table.get_item(Key={'username': username})

            if 'Item' not in response:
                return render_template('login.html', error='User not found')

            user = response['Item']

            # Ensure password field exists in the DB
            if 'password' not in user:
                return render_template('login.html', error='Password not found in database')

            # Verify password
            if check_password_hash(user['password'], password):
                session['logged_in'] = True
                session['username'] = username
                session.setdefault('cart', []) # Initialize cart if not set
                return redirect(url_for('home'))
```

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

```

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

        try:
            # Check if username exists
            response = users_table.get_item(Key={'username': username})
            if 'Item' in response:
                return render_template('signup.html', error='Username already exists')

            # Hash password before storing
            hashed_password = generate_password_hash(password)

            # Store new user in DynamoDB
            users_table.put_item(
                Item={
                    'username': username,
                    'email': email,
                    'password': hashed_password # Store hashed password
                }
            )

            return redirect(url_for('login'))

        except Exception as e:
            app.logger.error(f"Signup error: {str(e)}")
    
```

```

except Exception as e:
    app.logger.error(f"Signup error: {str(e)}")
    return render_template('signup.html', error='Registration failed. Please try again.')

return render_template('signup.html')

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

```

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

```

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

```

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

```

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

@app.route('/non_veg_pickles')
def non_veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('non_veg_pickles.html', products=products['non_veg_pickles'])

@app.route('/veg_pickles')
def veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products['veg_pickles'])

@app.route('/snacks')
def snacks():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('snacks.html', products=products['snacks'])
    
```

(i) Restart Visual Studio

- Check out Route:

```

@app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    error_message = None # Variable to hold error messages

    if request.method == 'POST':
        try:
            # Extract form data safely
            name = request.form.get('name', '').strip()
            address = request.form.get('address', '').strip()
            phone = request.form.get('phone', '').strip()
            payment_method = request.form.get('payment', '').strip()

            # Validate inputs
            if not all([name, address, phone, payment_method]):
                return render_template('checkout.html', error="All fields are required.")

            if not phone.isdigit() or len(phone) != 10:
                return render_template('checkout.html', error="Phone number must be exactly 10 digits.")

            # Get cart data from hidden inputs
            cart_data = request.form.get('cart_data', '[]')
            total_amount = request.form.get('total_amount', '0')
            
```

(i) Restart Visual Studio Code to apply the late

```

try:
    cart_items = json.loads(cart_data)
    total_amount = float(total_amount)
except (json.JSONDecodeError, ValueError):
    return render_template('checkout.html', error="Invalid cart data format.")

# Ensure cart is not empty
if not cart_items:
    return render_template('checkout.html', error="Your cart is empty.")

# Store order in DynamoDB
try:
    orders_table.put_item(
        Item={
            'order_id': str(uuid.uuid4()),
            'username': session.get('username', 'Guest'),
            'name': name,
            'address': address,
            'phone': phone,
            'items': cart_items,
            'total_amount': total_amount,
            'payment_method': payment_method,
            'timestamp': datetime.now().isoformat()
        }
    )
except Exception as db_error:
    print(f"DynamoDB Error: {db_error}")
    return render_template('checkout.html', error="Failed to save order. Please try again later.")

# Redirect to success page with success message
return redirect(url_for('sucess', message="Your order has been placed successfully!"))

except Exception as e:
    print(f"Checkout error: {str(e)}")
    return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

return render_template('checkout.html') # Render checkout page for GET request

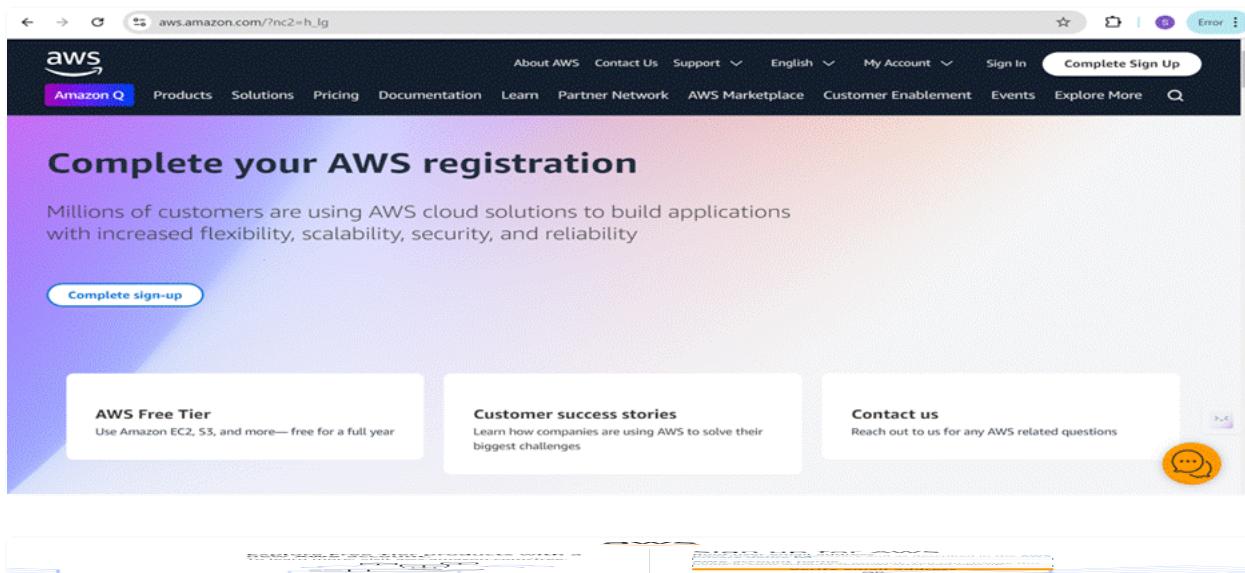
@app.route('/sucess')
def sucess():
    return render_template('sucess.html')

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

```

Milestone 2 : AWS Account Setup

- 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.
- 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.



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

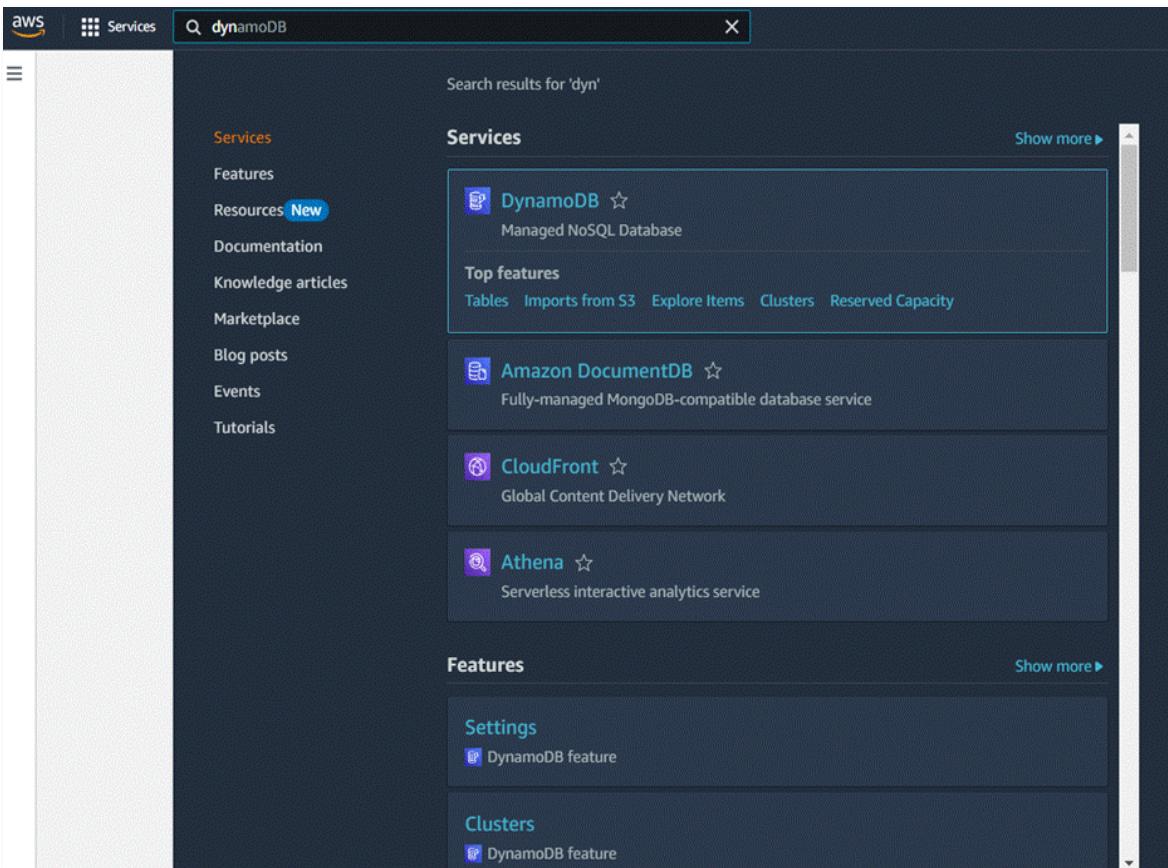
The screenshot shows the AWS sign-in page. The left side displays the "Sign in" form with two radio button options: "Root user" (selected) and "IAM user". Below the radio buttons is a "Root user email address" input field containing "username@example.com" and a "Next" button. At the bottom of the sign-in form, there is a small note about cookie consent. To the right of the sign-in form is a dark purple sidebar with the title "AI Use Case Explorer" in white. The sidebar text reads: "Discover AI use cases, customer success stories, and expert-curated implementation plans" and ends with a "Explore now >" link.

Milestone 3 : DynamoDB Database Creation and Setup

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

Navigate to the DynamoDB:

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





The screenshot shows two views of the AWS DynamoDB console. The top view is the 'Dashboard' page, which includes sections for 'Alarms', 'DAX clusters', and a 'Create resources' sidebar. The bottom view is the 'Tables' page, showing a table structure with columns for Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. It indicates 'You have no tables in this account in this AWS Region.' Both pages include a left sidebar with navigation links like 'Dashboard', 'Tables', 'Explore items', etc.

Create a DynamoDB table for storing data:

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

This screenshot shows the 'Create table' wizard in the AWS DynamoDB service. The top navigation bar includes the AWS logo, search bar, and [Alt+S] keyboard shortcut. The breadcrumb path is 'DynamoDB > Tables > Create table'. The main section is titled 'Create table' and contains three primary configuration sections: 'Table details', 'Partition key', and 'Sort key - optional'. Under 'Table details', the table name 'Users' is specified. Under 'Partition key', the key name 'Username' is set to type 'String'. Under 'Sort key - optional', the key name 'Enter the sort key name' is also set to type 'String'. Each section includes descriptive text and character count limits (1 to 255 characters).

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

Tags

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

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#) [Create table](#)

The Users table was created successfully.

DynamoDB X

Tables

DynamoDB > Tables

Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
Users	Active	email (S)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes

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



aws | ⚙️ Search [Alt+S]

DynamoDB > Tables > Create table

Create table

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

Table name
This will be used to identify your table.
 Orders
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and consistency.
 Order_id String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 Enter the sort key name String
1 to 255 characters and case sensitive.

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

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

No tags are associated with the resource.

Add new tag
You can add 50 more tags.

Cancel **Create table**

Find tables Any tag key Any tag value

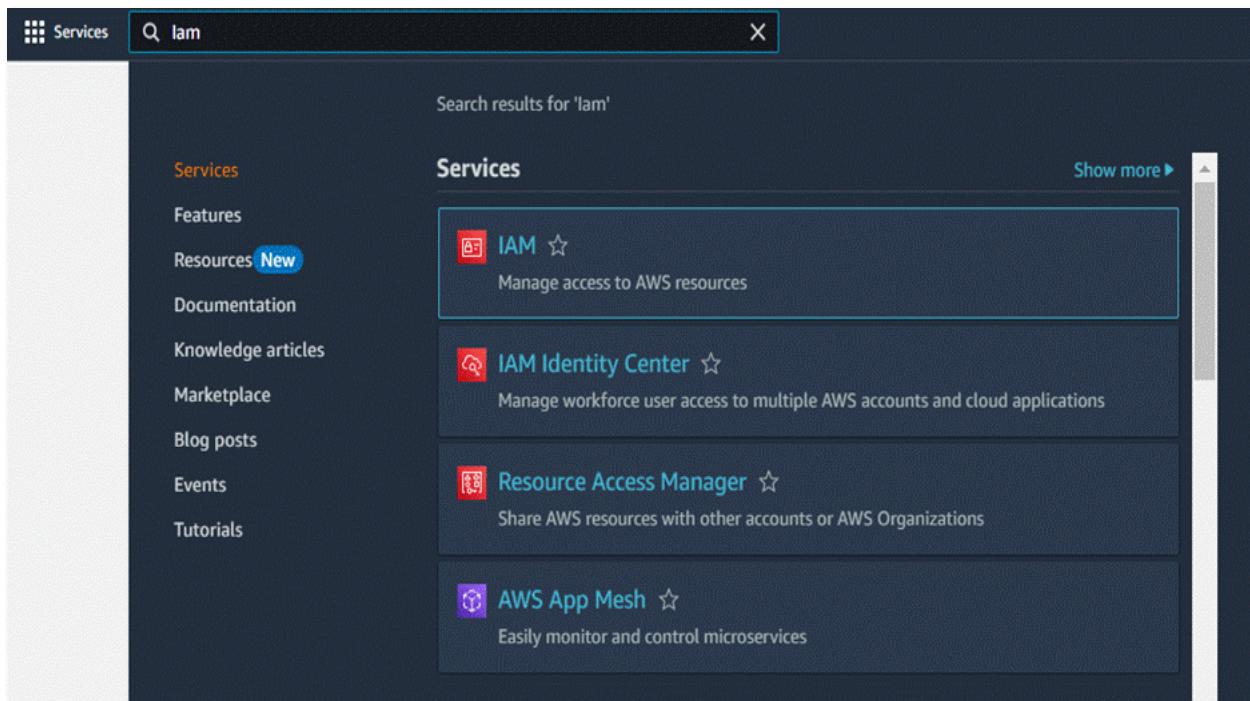
<input type="checkbox"/>	Name ▲	Status ▼	Partition key ▼	Sort key ▼	Indexes ▼	Replication Regions ▼	Deletion protect
<input type="checkbox"/>	Orders	Active	order_id (S)	-	0	0	<input type="checkbox"/> Off
<input type="checkbox"/>	Users	Active	username (S)	-	0	0	<input type="checkbox"/> Off

Milestone 4 : IAM Role Setup

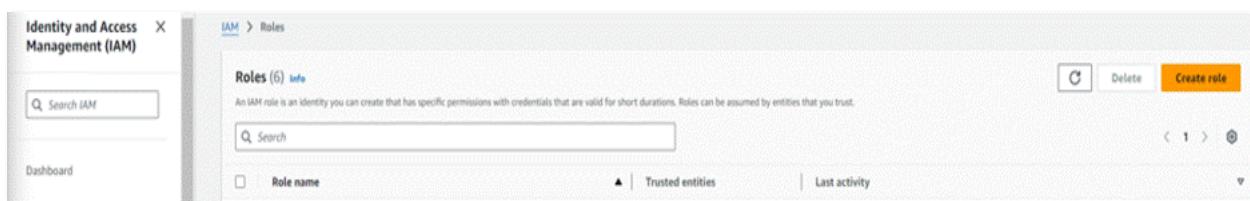
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows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

Create IAM Role:

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



The screenshot shows the AWS search results for the term 'iam'. The search bar at the top contains 'iam'. On the left, there is a sidebar with links for Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main area displays search results for 'iam'. The first result is 'IAM' with the subtext 'Manage access to AWS resources'. Below it are 'IAM Identity Center' (Manage workforce user access to multiple AWS accounts and cloud applications), 'Resource Access Manager' (Share AWS resources with other accounts or AWS Organizations), and 'AWS App Mesh' (Easily monitor and control microservices). A 'Show more ▶' link is visible on the right side of the results.



The screenshot shows the 'Roles' page within the AWS IAM service. The left sidebar has 'Identity and Access Management (IAM)' selected. The main content area shows a table with 6 rows under the heading 'Roles (6) Info'. The table includes columns for 'Role name', 'Trusted entities', and 'Last activity'. At the bottom of the table, there are buttons for 'Create role' and navigation arrows. The overall interface is the standard AWS dark-themed design.



Step 1 > Step 2 > Step 3 > Create role

Select trusted entity [info](#)

Trusted entity type

AWS service Allow AWS or AWS Lambda, or others to perform actions in this account.

AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAP 2.0 Federation Allows users federated with SAP 2.0 from a corporate directory to perform actions in this account.

Custom trust policy Create a custom trust policy to enable others to perform actions in this account.

Use case Allow an AWS service (EC2, Lambda, or others) to perform actions in this account.

Service or use case

Choose a use case for the specified service.

User Case EC2 Allows EC2 instances to call AWS services on your behalf.

EC2 Role for AWS Systems Manager Allows EC2 instances to assume an IAM role for AWS Systems Manager on your behalf.

EC2 Spot Fleet Role Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

EC2 - Spot Fleet Auto Scaling Allows Auto Scaling to access and update EC2 spot fleets on your behalf.

EC2 - Spot Fleet Tagging Allows EC2 to search spot instance and attach tags to the launched instances on your behalf.

EC2 - Spot Instances Allows EC2 Spot Fleet to launch and manage spot instances on your behalf.

EC2 - Scheduled Instances Allows EC2 scheduled instances to manage instances on your behalf.

[Cancel](#) [Next](#)

Step 1 > Step 2 > Step 3 > Create role

Add permissions [info](#)

Permissions policies (1/955) [info](#)

Choose one or more policies to attach to your new role.

Filter by Type All types

Policy name	Type
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed

Set permissions boundary - optional

[Cancel](#) [Previous](#) [Next](#)



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 New Role' wizard - Step 1: Select trusted entities.

Name details

Role name: sns_dynamodb_role

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

Step 1: Select trusted entities

Trust policy (JSON code shown):

```
1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Effect": "Allow",
5         "Principal": "*",
6         "Action": "sts:AssumeRole"
7     }
8 ]
```

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): No tags associated with the resources.

Add new tag: You can add up to 50 more tags.

Cancel | Preview | Create role

Screenshot of the AWS IAM 'Roles' page for the 'sns_Dynamodb_role'.

Summary

Creation date: October 13, 2024, 23:06 (UTC+05:30)	ARN: arn:aws:iam::5576900616836:role/sns_Dynamodb_role	Instance profile ARN: arn:aws:iam::5576900616836:instance-profile/sns_Dynamodb_role
Last activity: 6 days ago	Maximum session duration: 1 hour	

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (2)

You can attach up to 10 managed policies.

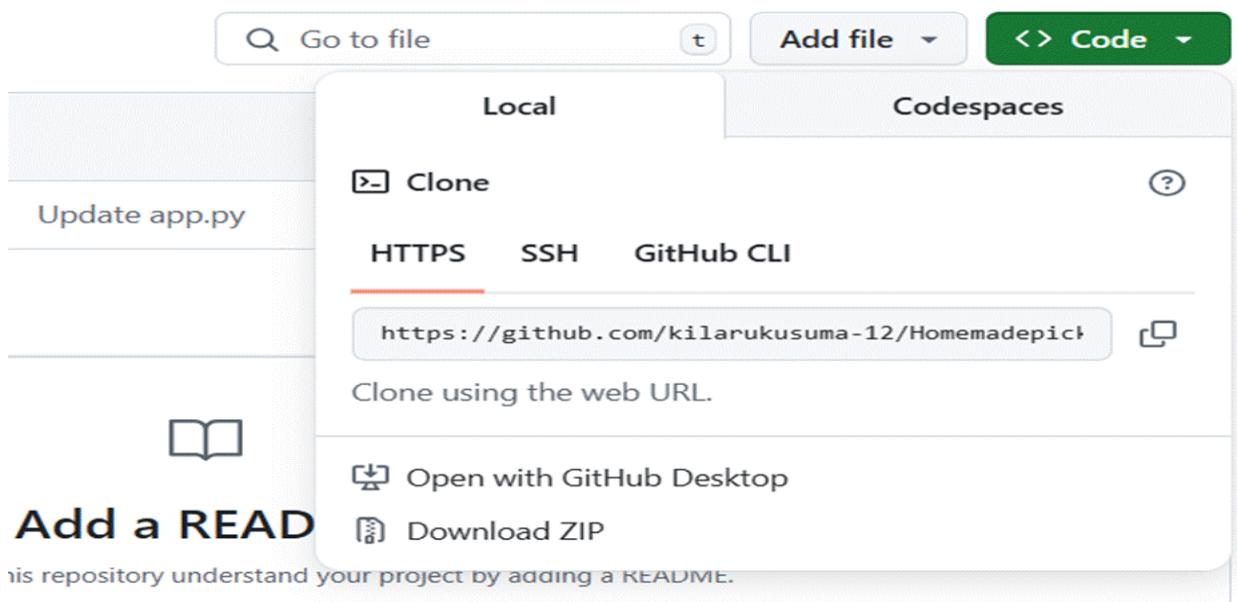
Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	4
AmazonSNSFullAccess	AWS managed	2

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 Flask app and Html files into GitHub repository.

 static	Initial commit
 templates	Update statistics.html
 app.py	Update app.py



The screenshot shows a GitHub repository interface. A context menu is open over the file 'Update app.py'. The menu includes options for cloning the repository via HTTPS, SSH, or GitHub CLI. The HTTPS URL is displayed as <https://github.com/kilarukusuma-12/Homemadepickle>. Other options include opening with GitHub Desktop or downloading as a ZIP file. At the bottom of the menu, there is a note: 'This repository understand your project by adding a README.'

Load your Project Files to GitHub

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

 static	Initial commit
 templates	Update statistics.html
 app.py	Update app.py

Local

Codespaces

 Clone



[HTTPS](#) [SSH](#) [GitHub CLI](#)

<https://github.com/AlekhyaPenubakula/InstantLit>



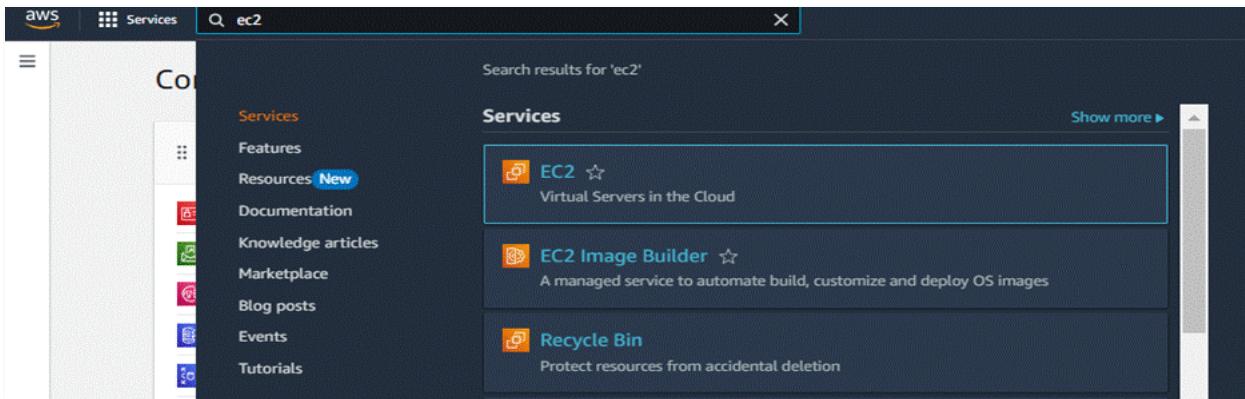
Clone using the web URL.

 Open with GitHub Desktop

 Download ZIP

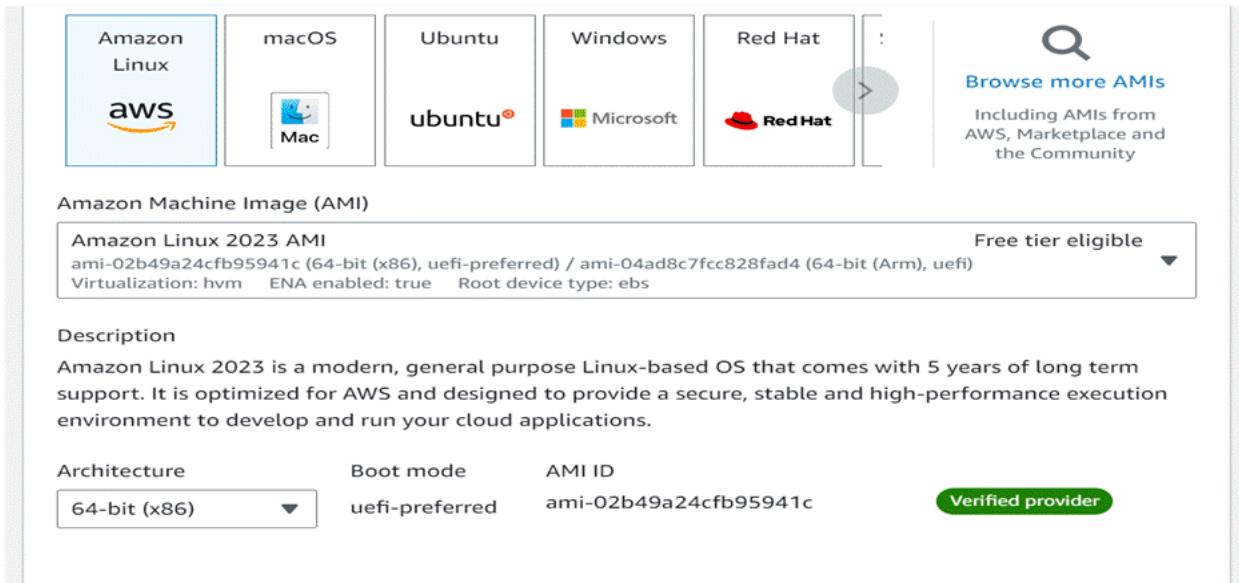
Launch an EC2 instance to host the Flask

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



- Click on Launch instance to launch EC2 instance

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



Amazon Machine Image (AMI)

Amazon Linux 2023 AMI

Free tier eligible

ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

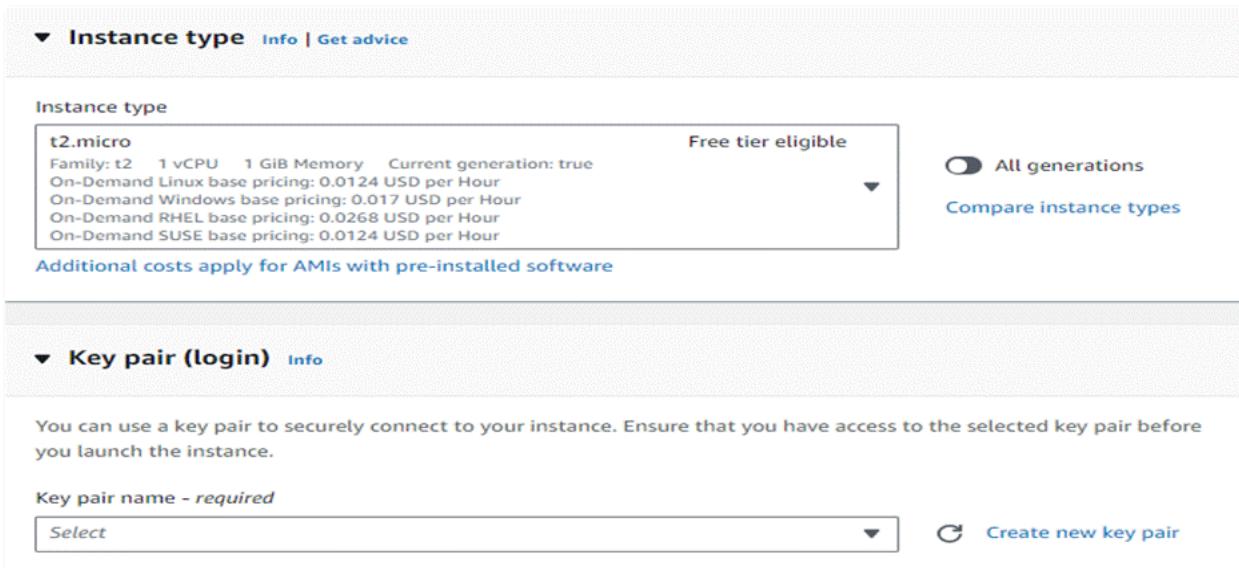
Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID
64-bit (x86)	uefi-preferred	ami-02b49a24cfb95941c

Verified provider

- Create and download the key pair for Server access.



Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro	Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true	
On-Demand Linux base pricing: 0.0124 USD per Hour	
On-Demand Windows base pricing: 0.017 USD per Hour	
On-Demand RHEL base pricing: 0.0268 USD per Hour	
On-Demand SUSE base pricing: 0.0124 USD per Hour	

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select [Create new key pair](#)



EC2 > Instances > Launch an instance

It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices [Do not show me](#)

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name [Add additional tags](#)

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

Recent **Quick Start**

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian [Browse more AMIs](#) Including AMIs from [Cancel](#)

InstantLibrary.pem

Description
Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID	Username
64-bit (x86)	uefi-preferred	ami-078264b8ba71b c45e	ec2-user Verified provider

Instance type Info | Get advice

Instance type **t2.micro** Free tier eligible

Family: t2	1 vCPU	1 GB Memory	Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour			
On-Demand Windows base pricing: 0.017 USD per Hour			
On-Demand RHEL base pricing: 0.0268 USD per Hour			
On-Demand SUSE base pricing: 0.0124 USD per Hour			

All generations [Compare instance types](#)

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required [Create new key pair](#)

Summary

Number of instances [Info](#) 1

Software Image (AMI)
Amazon Linux 2023 AMI 2023.5.2...read more
ami-078264b8ba71b
c45e

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

[Cancel](#) [Preview code](#) [Launch instance](#)

Configure security groups for HTTP and SSH access:

▼ Network settings [Info](#)

VPC - required [Info](#)
 vpc-03cdc7b6f19dd7211 (default) ▾
 172.31.0.0/16

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

Auto-assign public IP [Info](#)
 Enable ▾
 Additional charges apply when outside of **free tier** allowance

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

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

Security group name - required
 launch-wizard

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

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

Inbound Security Group Rules

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

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

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

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

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

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

[Add security group rule](#)



EC2 > ... > Launch an instance

Success
Successfully initiated launch of instance i-001861022fbac290

Launch log

Next Steps

Q. What would you like to do next with this instance, for example "create alarm" or "create backup"?

1 2 3 4 >

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

[View all instances](#)

- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

Instances (1/2) [Info](#)

Last updated less than a minute ago [C](#) Connect [Instance state](#) [Actions](#) [Launch instances](#)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs	Monitoring	Security
InstantLibrary...	i-001861022fbac290	Stopped	t2.micro	-	View alarms	ap-south-1b	-	-	-	-	disabled	launch-wi

EC2 > Instances > i-001861022fbac290

Instance summary for i-001861022fbac290 (InstantLibraryApp) [Info](#)

Updated less than a minute ago

Instance ID i-001861022fbac290	Public IPv4 address -	Private IPv4 addresses 172.31.5.5
IPv6 address -	Instance state Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Auto-assigned IP address -	VPC ID vpc-03cd7b6f19dd7211	Auto Scaling Group name -
IAM Role arn:aws:iam::123456789012:role/DynamoDBRole	Subnet ID subnet-0df9fa5144480cc9a9	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbac290
Details Status and alarms Monitoring Security Networking Storage Tags		



EC2 > Instances > i-001861022fbcac290

Instance summary for i-001861022fbcac290 (InstantLibraryApp) [Info](#)

Updated less than a minute ago

Instance ID i-001861022fbcac290	Public IPv4 address -	Private IPv4 addresses 172.31.3.5
IPv6 address -	Instance state Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-3-5.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-3-5.ap-south-1.compute.internal	Change security groups
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Get Windows password
Auto-assigned IP address -	VPC ID vpc-05cc0c7b6f19d7211	Modify IAM role
IAM Role sns_Dynamodb_role	Subnet ID subnet-0d9fa3144480cc9u9	Elastic IP addresses -
IMDsv2 Required	Instance ARN arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Auto Scaling Group name -		

Connect [Actions ▾](#)

EC2 > Instances > i-001861022fbcac290 > Modify IAM role

Modify IAM role [Info](#)

Attach an IAM role to your instance.

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

IAM role

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

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

[Cancel](#) [Update IAM role](#)



- Now connect the EC2 with the files

Connect to instance

Info

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

[EC2 Instance Connect](#) | [Session Manager](#) | [SSH client](#) | [EC2 serial console](#)

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

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

Instance ID

Connection Type

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

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

X

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

[Cancel](#) [Connect](#)

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

i-001861022fbcac290 (InstantLibraryApp)

Public IPs: 13.201.74.42 Private IPs: 172.31.3.5

Milestone 6 : Deployment on EC2

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

Install Software on the EC2 Instance:

Install Python3, Flask, and Git:

On Amazon Linux 2:

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

Verify Installations:

- flask --version
- git --version

Clone Your Flask Project from GitHub:

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

- Run
- change your-github-username and your-repository-name with your credential
- This will download your project to the EC2 instance.

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

- cd Homemadepicklesandsnacks
- cd "Home Made Pickles1"

Create a Virtual Environment:

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



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

- Run the Flask Application
 - `sudo flask run --host=0.0.0.0 --port=5000`

```
A newer release of "Amazon Linux" is available.
 Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
      #
      ##
      Amazon Linux 2023
      \##
      https://aws.amazon.com/linux/amazon-linux-2023
      '-
      /_
      /m.

Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$ git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git
fatal: destination path 'InstantLibrary' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-5 ~]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ flask run --host=0.0.0.0 --port=80
 * Debug mode: off
Permission denied
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
^C[ec2-user@ip-172-31-3-5 InstantLibrary]$ 
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```



Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

```
NodeModuleNotFoundError: No module named 'boto3'

[ec2-user@ip-172-31-15-149 Home made pickles]$ sudo pip3 install boto3
Collecting boto3
  Downloading boto3-1.37.23-py3-none-any.whl (139 kB)
    [██████████] 139 kB 8.1 MB/s
Collecting s3transfer<0.12.0,>=0.11.0
  Downloading s3transfer-0.11.4-py3-none-any.whl (84 kB)
    [██████████] 84 kB 6.7 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.38.0,>=1.37.23
  Downloading botocore-1.37.23-py3-none-any.whl (13.4 MB)
    [██████████] 13.4 MB 38.3 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<2.1> in /usr/lib/python3.9/site-packages (from botocore<1.38.0,>=1.37.23->boto3) (2.8.1)
Requirement already satisfied: six!=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.38.0,>=1.37.23->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.37.23 botocore-1.37.23 s3transfer-0.11.4
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-15-149 Home made pickles]$ sudo flask run --host=0.0.0.0 --port=5000
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.15.149:5000
Press CTRL+C to quit
```

Access the website through:

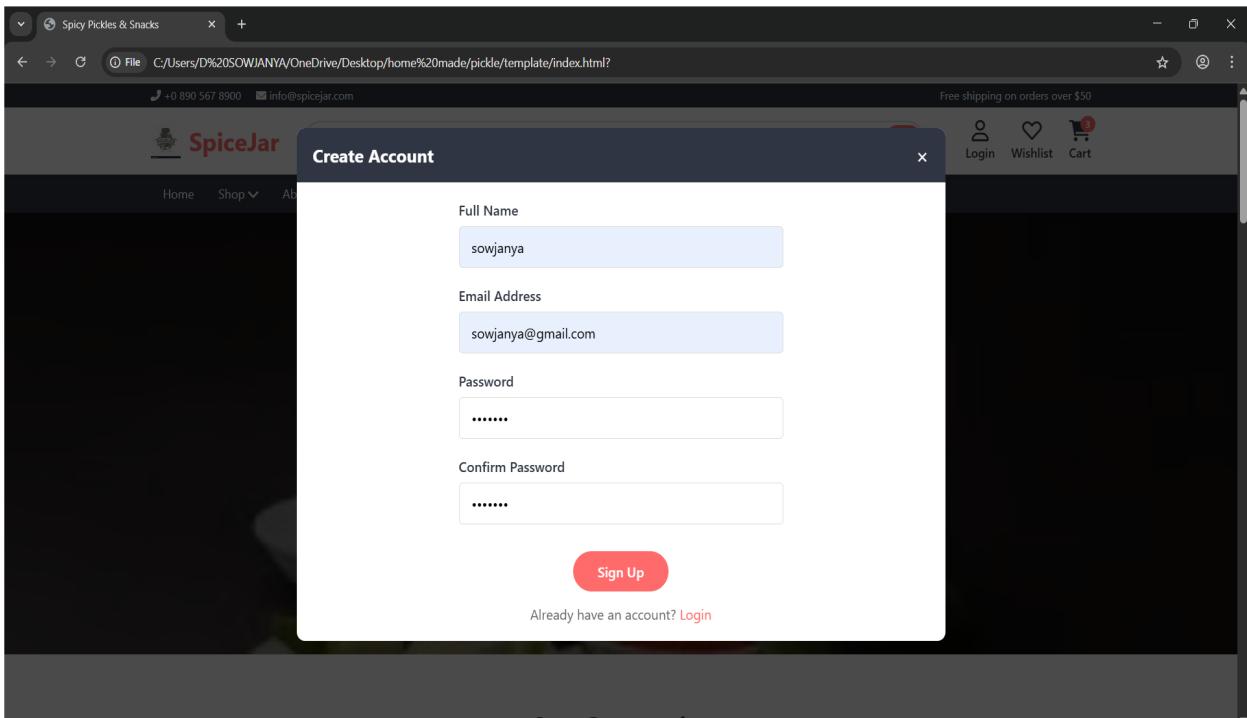
Public IPs: <http://192.168.137.120:5000/>

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:

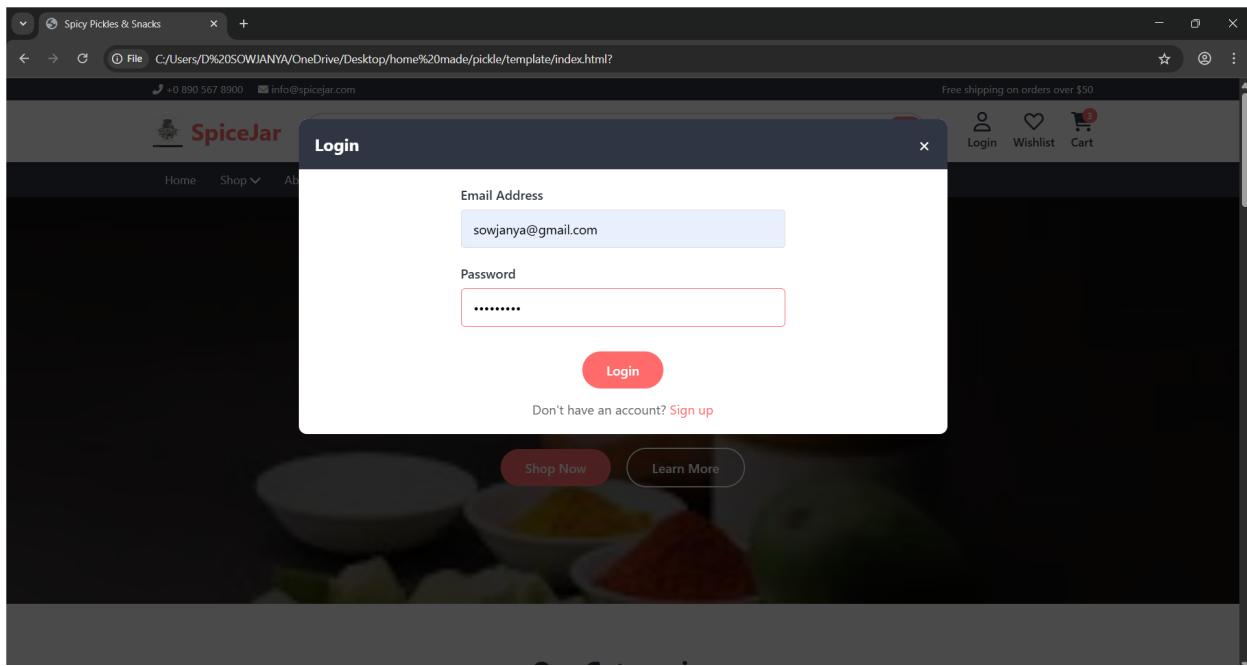
Signup page:



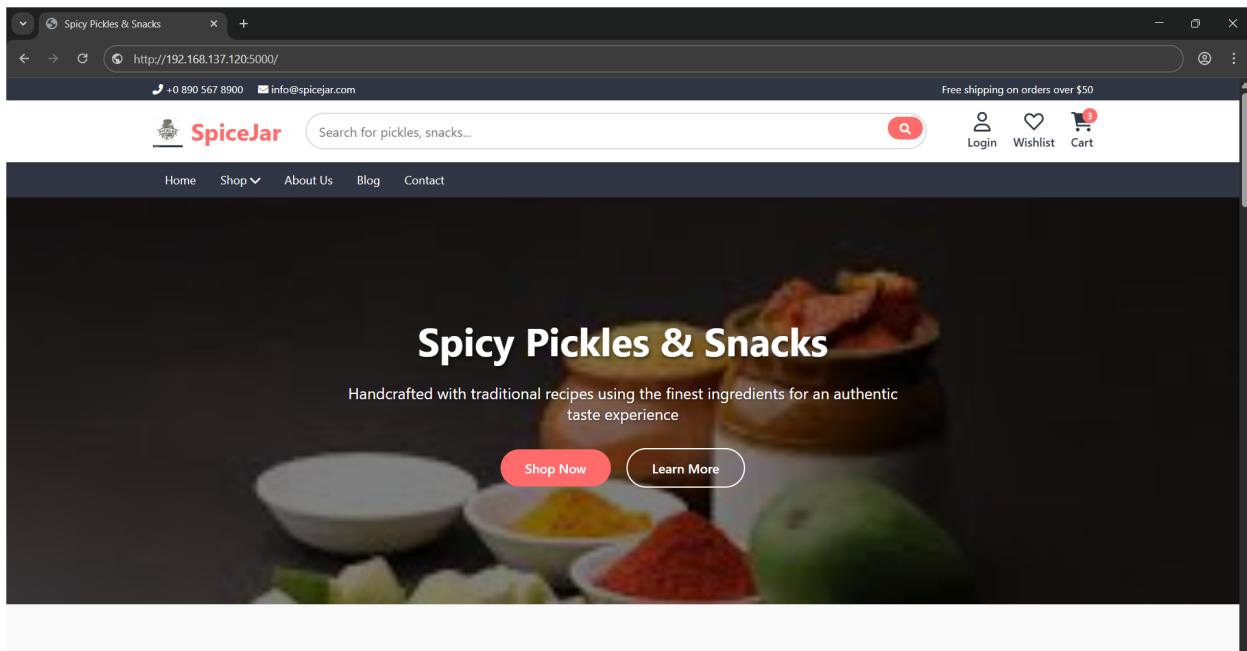
The screenshot shows a web browser window with a dark-themed user interface. At the top, the address bar displays the URL: C:/Users/D%20SOWJANYA/OneDrive/Desktop/home%20made/pickle/template/index.html?. Below the address bar, there are navigation icons and contact information: +0 890 567 8900 and info@spicejar.com. A banner at the top right states "Free shipping on orders over \$50". The main content area features a "Create Account" modal window. The modal has a dark header with the text "Create Account". Inside the modal, there are four input fields: "Full Name" containing "sowjanya", "Email Address" containing "sowjanya@gmail.com", "Password" containing "*****", and "Confirm Password" containing "*****". At the bottom of the modal is a red "Sign Up" button. Below the modal, a link says "Already have an account? [Login](#)". The background of the browser shows a dark-themed website for "SpiceJar" with sections for Home, Shop, About, and Contact.



Login Page:



Home page:



Spicy Pickles & Snacks http://192.168.137.120:5000/

 Search for pickles, snacks... Login Wishlist Cart

Our Categories



Vegetarian Pickles
Traditional recipes with fresh vegetables



Non-Veg Pickles
Rich and flavorful meat-based pickles



Snacks
Perfect accompaniments for your pickles

Spicy Pickles & Snacks http://192.168.137.120:5000/

 Search for pickles, snacks... Login Wishlist Cart

Featured Products



Bestseller
Vegetarian Pickles
Traditional Mango Pickle
\$12.99 ~~\$15.99~~
★ ★ ★ ★ ★ (42)
[Add to Cart](#)



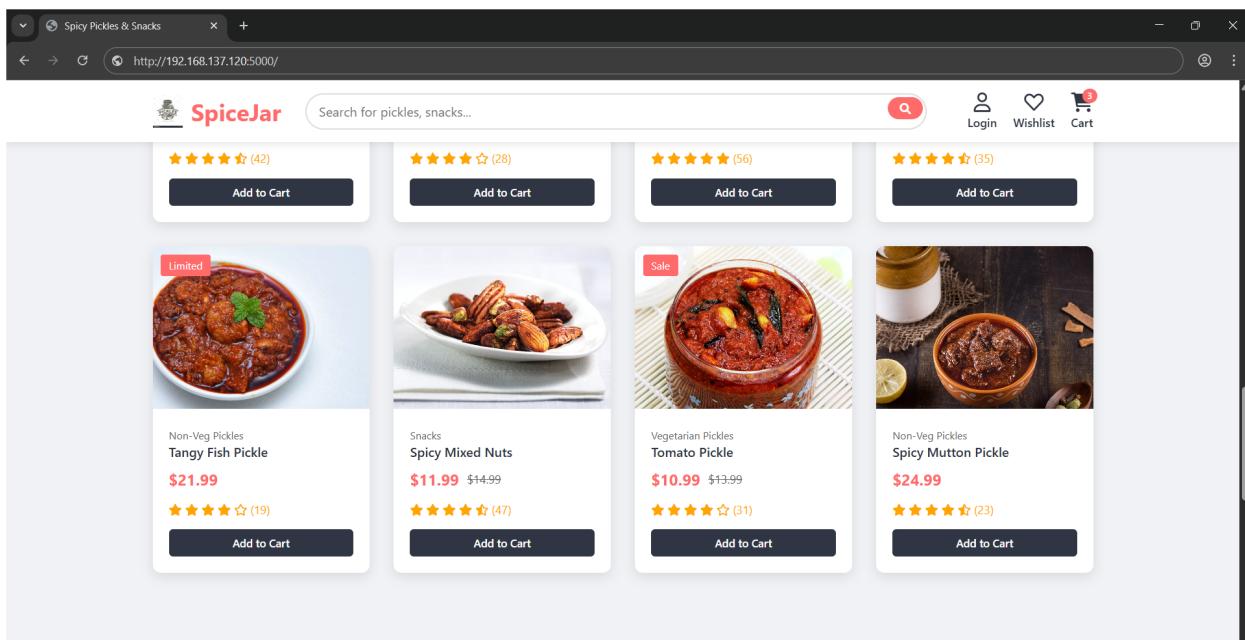
Non-Veg Pickles
Spicy Chicken Pickle
\$18.99
★ ★ ★ ★ ★ (28)
[Add to Cart](#)



New
Snacks
Crispy Papadum
\$8.99 ~~\$10.99~~
★ ★ ★ ★ ★ (56)
[Add to Cart](#)



Vegetarian Pickles
Mixed Vegetable Pickle
\$14.99
★ ★ ★ ★ ★ (35)
[Add to Cart](#)



SpiceJar

Search for pickles, snacks...

Add to Cart Add to Cart Add to Cart Add to Cart

Limited

Non-Veg Pickles
Tangy Fish Pickle
\$21.99

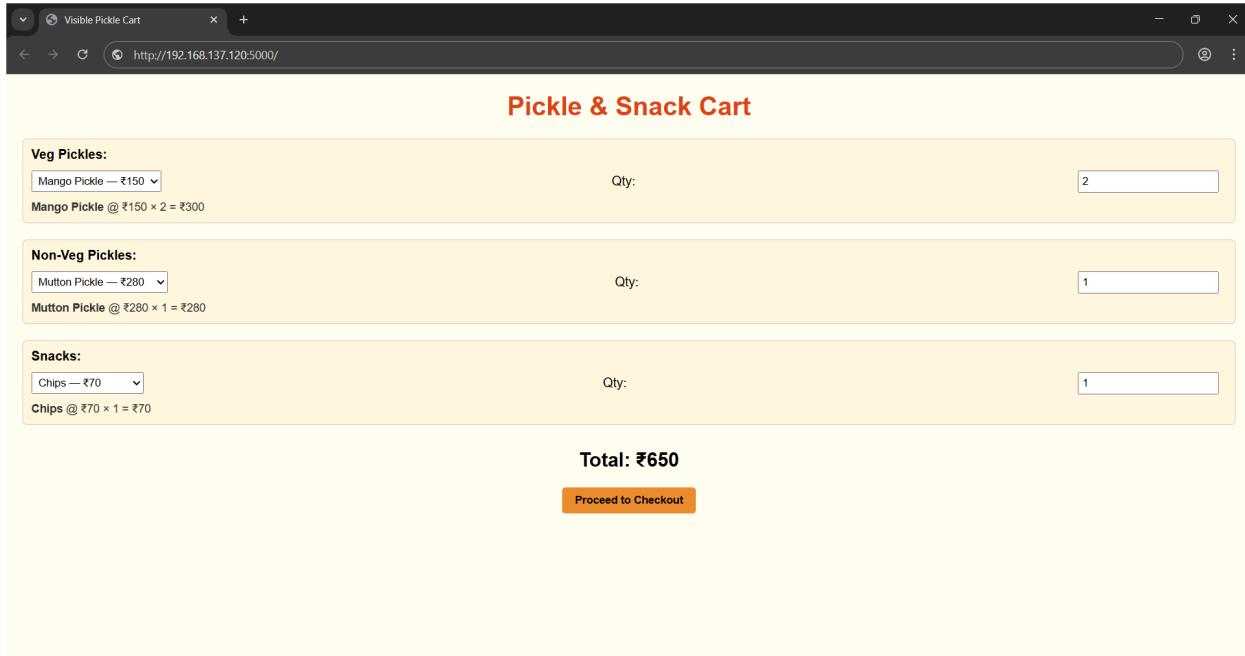
Snacks
Spicy Mixed Nuts
\$11.99

Sale

Vegetarian Pickles
Tomato Pickle
\$10.99

Non-Veg Pickles
Spicy Mutton Pickle
\$24.99

Cart Page:



Pickle & Snack Cart

Veg Pickles:
Mango Pickle — ₹150 Qty: 2

Non-Veg Pickles:
Mutton Pickle — ₹280 Qty: 1

Snacks:
Chips — ₹70 Qty: 1

Total: ₹650

Proceed to Checkout



Check out Page:

The screenshot shows a web browser window with the URL <http://192.168.137.120:5000/>. The main content is a 'Secure Checkout' form. It has two sections: 'Shipping Info' and 'Payment Details'. Under 'Shipping Info', there are four input fields containing 'sowjanya', 'srivilliputhur', '97876533456', and 'sowjanya@gmail.com'. Under 'Payment Details', there are three input fields containing '65436as', '09/12', and 'cash'. At the bottom is a dark green 'Place Order' button.

Success page:

The screenshot shows a web browser window with the URL <http://192.168.137.120:5000/>. The main content is a 'Secure Checkout' page with a success message. It features a large green banner with the text 'Order Successful!' and an emoji of a person with arms raised. Below the banner, it says 'Thanks for shopping with us.' and 'Your Order ID is #ORD725310'.

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"/>	Shiva	kilarukusu...	scrypt:32768:8:1\$W5tA59Z7nQjLXbtx\$d6bfef2b3e14bbe9d3d3e3f1c...
<input type="checkbox"/>	kusuma	<empty>	<empty>
<input type="checkbox"/>	Alekhya	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"/>	7c6bd84e-f2c7-4fe0...	Kothur	[{"M": {"n..."}]	Siri	cod	8187810
<input type="checkbox"/>	3de0fe0c-9539-4fb6...	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849889
<input type="checkbox"/>	fbc41d6d-d6f2-4158...	chatanpally	[{"M": {"n..."}]	KILARU KU...	cod	9849889
<input type="checkbox"/>	1g	<empty>	[]	<empty>	<empty>	0



Conclusion:

The Homemade Pickles and Snacks site was carefully conceived to honour the depth of traditional flavours uncompromisingly with quality. All products—from spicy pickles to crunchy snacks—are made in small batches with ancient recipes and preservative-free ingredients to give an authentic taste that speaks to food enthusiasts.

Not only does the platform make it easier to shop online with dynamic cart management, secure payment processes, and speedy checkout but it also maximizes visibility for thoroughly crafted, handmade products. Every listing showcases the attention to detail and handiwork that went into creating the product, with weight-based pricing and category-based browsing making discovery simple and tailored.

Flexible cloud infrastructure, developed using Flask and AWS, allows the platform to scale with demand while keeping orders, inventory, and customer preferences current in real time. Sellers enjoy easy-to-use dashboards for product management, and buyers have quick access to original, high-quality homemade snacks.

This project does more than digitize the culinary experience—it reimagines it, putting artisanal products front and center where they belong. With each click, consumers move one step closer to a richly flavoured experience based in heritage, reliability, and superior taste