# Fresh Basket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS

A PROJECT REPORT

Submitted by


| | |
|---|---|
| BRINDHA SELVA VANAJA .V. T | 113322243014 |
| GOPIKA.S | 113322243026 |
| POOJA.G | 113322243072 |
| PRIYA DARSHINI.M | 113322243079 |


**BACHELOR OF TECHNOLOGY**

***ARTIFICIAL INTELLIGENCE AND DATA SCIENCE VELAMMAL INSTITUTE OF TECHNOLOGY CHENNAI 601 204***

**ANNA UNIVERSITY: CHENNAI 600 025**

**DECEMBER 2024 - 2025**

# BONAFIDE CERTIFICATE

Certified that this project report "**Fresh Basket: Scalable E-commerce Platform Deployment with Flask on AWS EC2 and RDS**" is the Bonafide work of "**BRINDHA SELVA VANAJA V T -113322243014, GOPIKA.S -113322243026, POOJA G -113322243072, PRIYA DARSHINI.M - 113322243079**" who carried out the project work under my supervision.

SIGNATURE

**DR.S.PADMAPRIYA,M.E,Ph.D**

**PROFESSOR,**

**HEAD OF THE DEPARTMENT,**

Artificial Intelligence & Data Science,
Velammal Institute of Technology,
Velammal Gardens, Panchetti,
Chennai-601 204.

SIGNATURE

**MR. DINESH KUMAR K**

**ASSISSTANT PROFESSOR ,**

**NM COORDINATOR ,**

Artificial Intelligence & Data Science,
Velammal Institute of Technology,
Velammal Gardens, Panchetti,
Chennai-601 204

# ACKNOWLEDGEMENT

We are personally indebted to many who had helped us during the course of this project work. Our deepest gratitude to the **God Almighty.**

We are greatly and profoundly thankful to our beloved Chairman **Thiru.M.V.Muthuramalingam** for facilitating us with this opportunity. Our sincere thanks to our respected Director **Thiru.M.V.M Sasi Kumar** for his consent to take up this project work and make it a great success.

We are also thankful to our Advisors **Shri.K.Razak**, **Shri.M.Vaasu**, our Principal **Dr.N.Balaji** and our Vice Principal **Dr.S.Soundararajan** for their never ending encouragement that drives us towards innovation.

We are extremely thankful to our Head of the Department **Dr.S.PadmaPriya and** Naan Mudhalvan Coordinator **Mr. Dinesh Kumar K**, for their valuable teachings and suggestions.

The Acknowledgment would be incomplete if we would not mention word of thanks to our Parents, Teaching and Non-Teaching Staffs, Administrative Staffs and Friends for their motivation and support throughout the project.

Finally, we thank all those who directly or indirectly contributed to the successful completion of this project. Your contributions have been a vital part of our success

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

E-commerce platforms have revolutionized the retail industry, offering customers convenience and a wide range of products at their fingertips. FreshBasket is designed as a modern e-commerce solution with features like product browsing, cart management, and order placement. This project emphasizes scalability, security, and high availability by leveraging AWS services.

The "FreshBasket" project involves developing and deploying a scalable e-commerce platform for selling vegetables and fruits. The platform uses Flask for backend development, AWS EC2 for hosting, and Amazon RDS for database management. This cloud-native solution ensures high availability, scalability, and efficient management of the platform's operations. The project demonstrates how leveraging AWS services can create a robust infrastructure for managing user interactions, product catalogs, and order processing in a seamless manner.

E-commerce has emerged as one of the most transformative aspects of modern business, redefining the way goods and services are exchanged. It has broken down traditional barriers, enabling businesses to reach a global audience while offering consumers the convenience of shopping from anywhere at any time. This project is centered on developing a comprehensive e-commerce platform designed to provide a seamless and engaging experience for users. The platform aims to bridge the gap between buyers and sellers by creating a digital marketplace that is intuitive, reliable, and secure.

At its core, the project focuses on user-centric design, ensuring that the interface is easy to navigate, visually appealing, and optimized for both desktop and mobile devices. The goal is to make the purchasing process effortless for consumers while equipping sellers with powerful tools for managing their online storefronts. Features such as product listing management, inventory tracking, and sales analytics will empower sellers to make informed decisions and grow their businesses. For buyers, the platform will offer advanced search functionality, personalized recommendations based on preferences and behavior, and a streamlined checkout process.

# CHAPTER 2

# PROJECT OVERVIEW

## Objective:

The *FreshBasket* project aims to develop and deploy a scalable, cloud-based e-commerce platform specifically designed for online grocery shopping. The platform serves as a model for small to medium-sized businesses looking to establish a reliable and scalable online presence. By leveraging Flask for backend development and AWS services for deployment, the project focuses on ensuring high availability, performance, and data security.

The project aims to create a digital marketplace that enhances the shopping experience by offering intuitive navigation, personalized recommendations, and efficient transaction processes. It seeks to empower sellers with robust tools for product management, sales tracking, and inventory control while ensuring data security and trust through advanced encryption and payment systems. Additionally, the project aspires to leverage cutting-edge technologies such as artificial intelligence and data analytics to optimize user engagement, improve business outcomes, and drive the scalability of the platform to meet evolving market demands.

## Key Features:

1. **User Authentication System:**
    - Secure login and registration.
    - Session management and password encryption for user data protection.
2. **Product Management:**
    - Dynamic product catalog with search and filter functionality.
    - Displays product details such as name, price, category, and availability.
3. **Shopping Cart and Checkout System:**
    - Allows users to add/remove items from the cart dynamically.

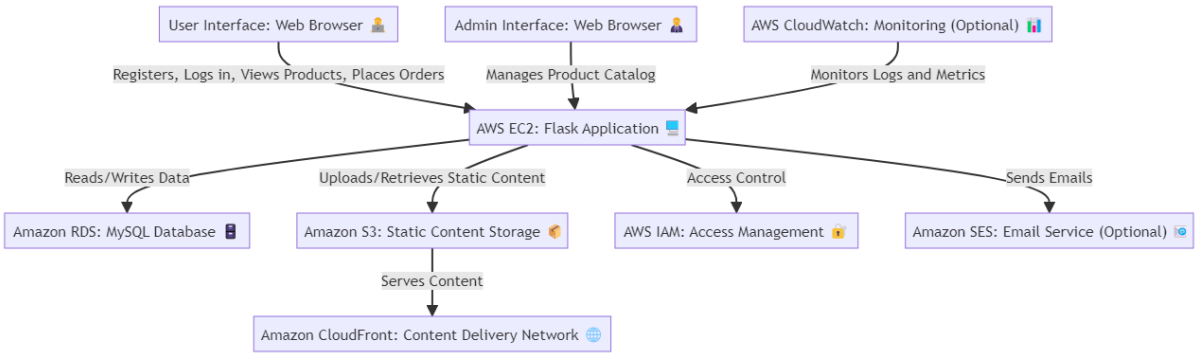
- Provides real-time updates on cart total and stock availability.
- Simulates order confirmation with a simple checkout process.

4. **Order Management:**
    - Stores order history for users.
    - Enables users to track the status of their orders.
5. **Responsive Design:**
    - Ensures seamless user experience across desktop, tablet, and mobile devices.
6. **Scalable Deployment:**
    - Hosted on AWS EC2 with auto-scaling enabled to manage high traffic loads.
    - Database hosted on AWS RDS, supporting automatic backups and multi-AZ deployments for high availability.
7. **Performance Optimization:**
    - Use of Gunicorn as a WSGI server to efficiently handle multiple requests.
    - Nginx as a reverse proxy for load distribution and serving static files.
8. **Monitoring and Logging:**
    - AWS CloudWatch for real-time monitoring of system performance.
    - Error logs and access logs for debugging and analytics.

# CHAPTER 3

# ARCHITECTURE

The architecture for the FreshBasket e-commerce platform is built on a cloud-based, modular system that ensures scalability, reliability, and security. The system is hosted primarily on AWS services, utilizing various components to manage different aspects of the application efficiently.




## Key Components and Their Roles:

1. **User Interface (Web Browser):**
    - **Purpose:** Allows customers to interact with the FreshBasket platform.
    - **Functions:** Users can register, log in, view products, manage their shopping cart, and place orders.
    - **Admin Interface:** Used by administrators to manage the product catalog, view orders, and perform other administrative tasks.
2. **AWS EC2: Flask Application Server**
    - **Purpose:** Hosts the Flask web application, acting as the core of the e-commerce platform.
    - **Functions:**
        - Manages user requests, product catalog, shopping cart, and orders.

        - Interfaces with the database, cloud storage, and other AWS services.
    - **Deployment:** Supports auto-scaling to handle traffic spikes efficiently.

3. **Amazon RDS (MySQL Database):**
    - **Purpose:** Stores application data, including user information, product details, orders, and cart data.
    - **Functions:** Provides a reliable and scalable relational database solution.
    - **Integration:** Flask application reads and writes data to RDS for seamless operation.
4. **Amazon S3 (Static Content Storage):**
    - **Purpose:** Stores static content such as product images, user uploads, and other media.
    - **Functions:** Enables efficient retrieval and storage of static resources.
    - **Integration:** The Flask application uploads/retrieves content from S3.
5. **Amazon CloudFront (Content Delivery Network):**
    - **Purpose:** Distributes static and dynamic content globally with low latency.
    - **Functions:** Improves the performance of content delivery by caching it in multiple geographic locations.
    - **Integration:** Serves static content from S3 via CloudFront for faster user experiences.
6. **AWS IAM (Access Management):**
    - **Purpose:** Provides secure access control to AWS resources.
    - **Functions:** Manages roles, policies, and permissions to ensure only authorized access to the AWS infrastructure.
    - **Integration:** Ensures secure communication between the Flask app and other AWS services.**3. Knowledge of Amazon EC2 (Elastic Compute Cloud)**

# CHAPTER 4
# PROJECT WORKFLOW

The workflow of an e-commerce project outlines the structured process for designing, developing, and deploying a comprehensive platform. Below is a detailed workflow highlighting each phase of the project:

1. Project Initialization:

- Define objectives, scope, and KPIs for deploying the FreshBasket e-commerce platform.
- Set up the AWS environment, including EC2 instance configuration and RDS setup.
- Outline the use of Flask for backend development and integration of AWS services.

2. EC2 Instance Creation:

- Launch an EC2 instance to host the FreshBasket application.
- Select the appropriate instance type based on expected traffic and resource requirements.

3. RDS Configuration:

- Set up Amazon RDS for database management with MySQL.
- Configure database instances, including security settings and access controls.

4. Flask Application Deployment:

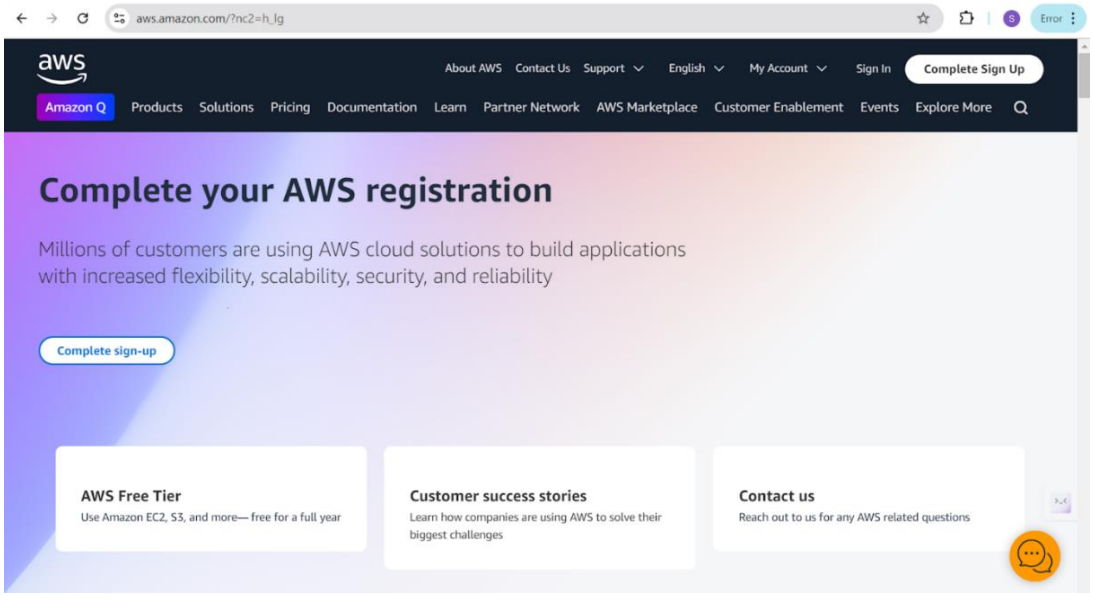
- Develop and deploy the FreshBasket application using Flask.
- Transfer application files to the EC2 instance and configure the environment.

# CHAPTER 5

# Project Implementation Process

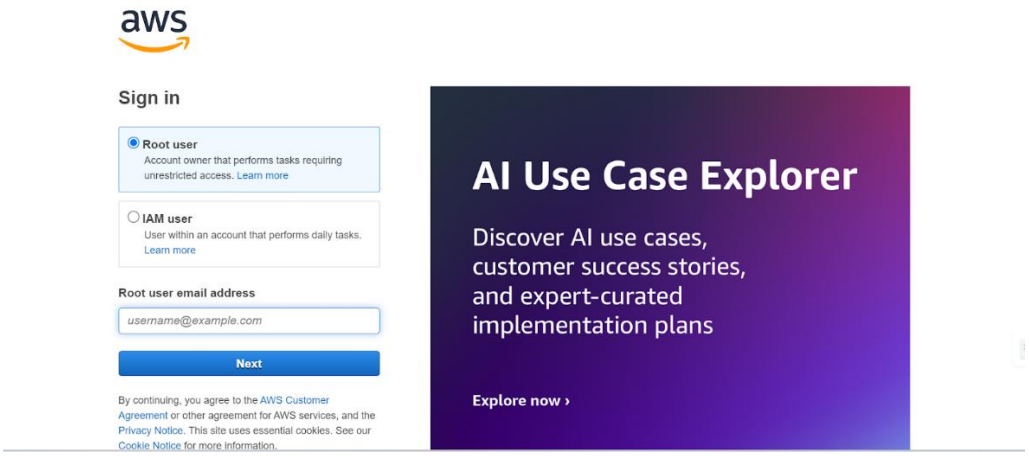
Create AWS Account

- Sign up for an AWS account and configure billing settings.




## Log in to AWS Management Console

- Access the AWS Management Console using your login credentials.

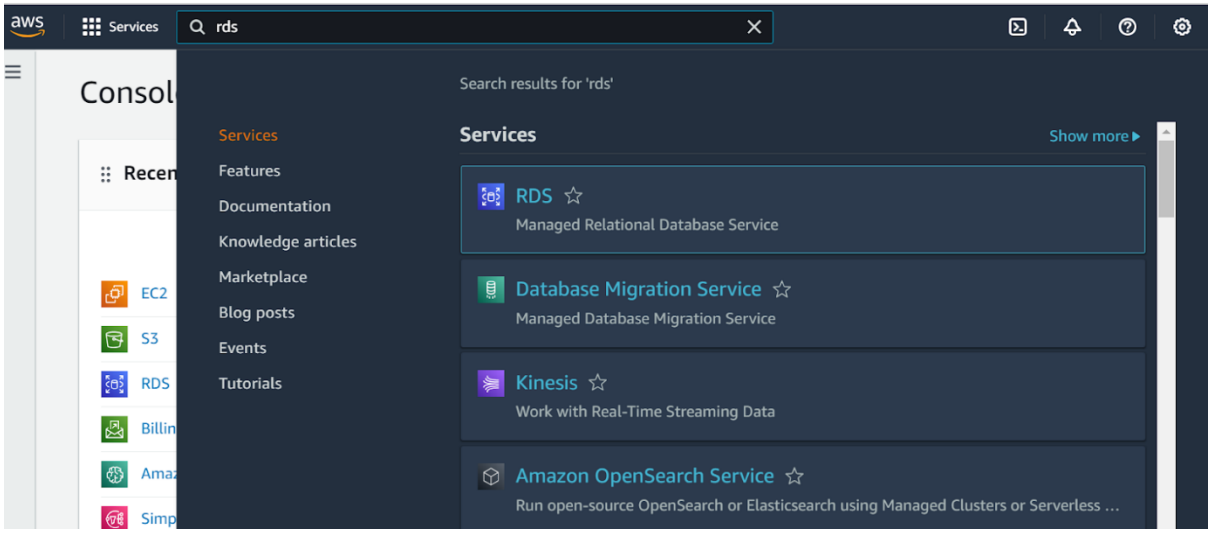



## AWS Account Setup and Login

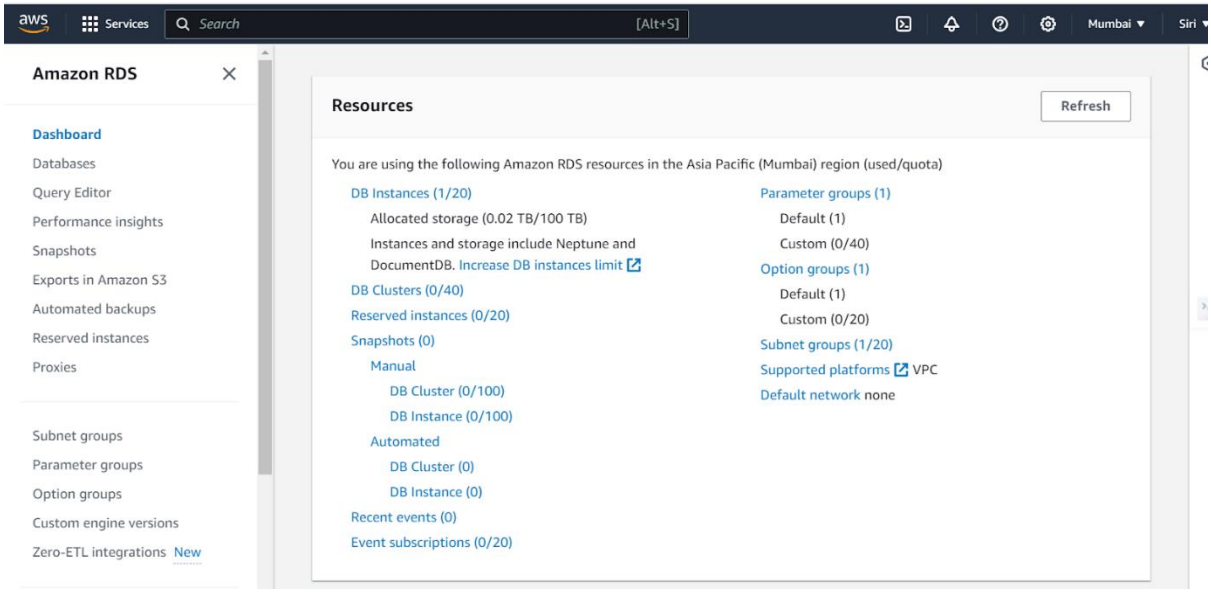
## RDS Database Creation and Setup

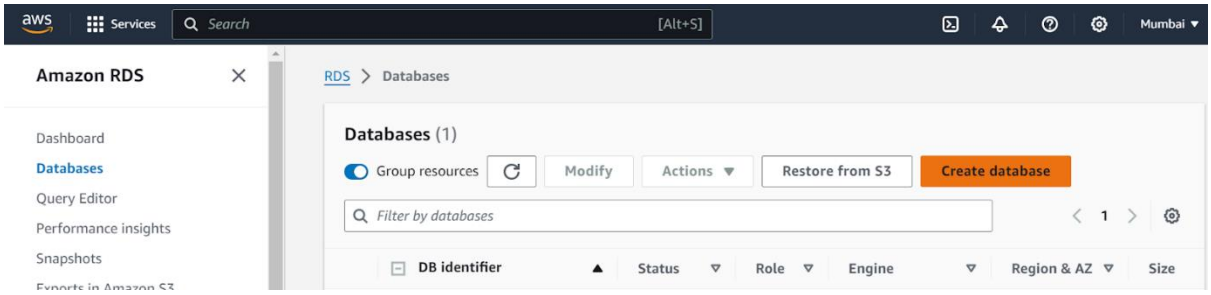
### Create an RDS Instance

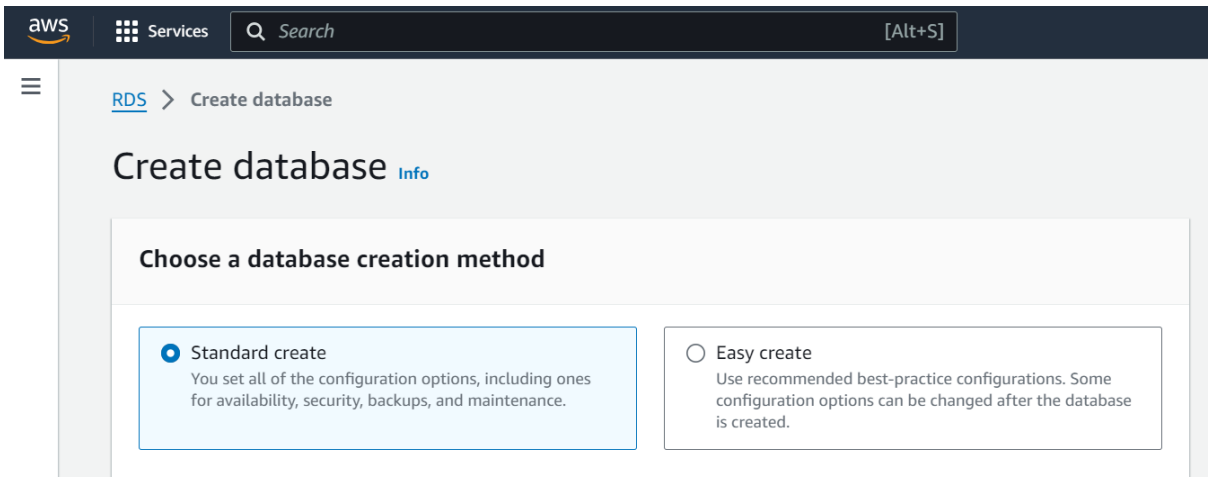
- Choose the RDS service from the AWS Management Console.

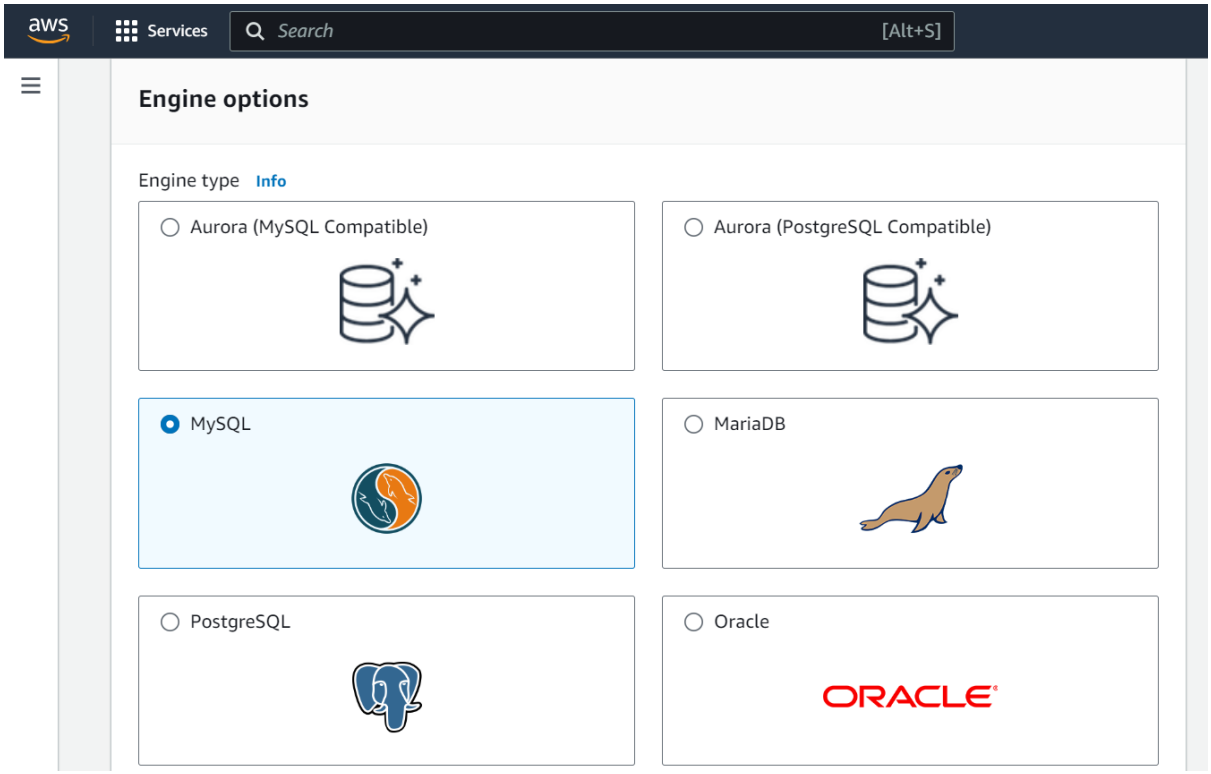



- Select MySQL as the database engine, configure the instance settings (e.g., storage, instance class), and launch the RDS instance.




aws
Services
Search
[Alt+S]
RDS
Create database
Create database
Info
Choose a database creation method
Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.
Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.


aws
Services
Search
[Alt+S]
Engine options
Engine type
Info
Aurora (MySQL Compatible)
Aurora (PostgreSQL Compatible)
MySQL
MariaDB
PostgreSQL
Oracle
ORACLE

Edition

MySQL Community

Engine version Info

View the engine versions that support the following database features.

Hide filters

Show versions that support the Multi-AZ DB cluster Info

Create a A Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ DB clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

Show versions that support the Amazon RDS Optimized Writes Info

Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Engine Version

MySQL 8.0.35

Enable RDS Extended Support Info

Amazon RDS Extended Support is a paid offering. By selecting this option, you consent to being charged for this offering if you are running your database major version past the RDS end of standard support date for that version. Check the end of standard support date for your major version in the RDS for MySQL documentation.

## Templates

Choose a sample template to meet your use case.

Production

Use defaults for high availability and fast, consistent performance.

Dev/Test

This instance is intended for development use outside of a production environment.

Free tier

Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. Info

## Availability and durability

Deployment options Info

The deployment options below are limited to those supported by the engine you selected above.

Multi-AZ DB Cluster

Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.

Multi-AZ DB instance (not supported for Multi-AZ DB cluster snapshot)

Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.

Single DB instance (not supported for Multi-AZ DB cluster snapshot)

Creates a single DB instance with no standby DB instances.

## Settings

DB instance identifier Info

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

database-1

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username Info

Type a login ID for the master user of your DB instance.

admin

1 to 16 alphanumeric characters. The first character must be a letter.

Credentials management

You can use AWS Secrets Manager or manage your master user credentials.

- ○ Managed in AWS Secrets Manager - *most secure*
  RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.
- ◉ Self managed
  Create your own password or have RDS create a password that you manage.

☐ Auto generate password

Amazon RDS can generate a password for you, or you can specify your own password.

Master password Info

••••••••

Password strength Neutral

Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @

Confirm master password Info

••••••••

### Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class Info

▼ Hide filters

- ○ Show instance classes that support Amazon RDS Optimized Writes Info
  Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.
- ○ Include previous generation classes

- ○ Standard classes (includes m classes)
- ○ Memory optimized classes (includes r and x classes)
- ◉ Burstable classes (includes t classes)

## Storage

Storage type Info

Provisioned IOPS SSD (io2) storage volumes are now available.

General Purpose SSD (gp3)
Performance scales independently from storage

Allocated storage Info

20 GiB

Minimum: 20 GiB. Maximum: 6,144 GiB

After you modify the storage for a DB instance, the status of the DB instance will be in storage-optimization. Your instance will remain available as the storage-optimization operation completes. Learn more

▶ **Advanced settings**

Baseline IOPS of 3,000 IOPS and storage throughput of 125 MiBps are included for allocated storage less than 400 GiB.

▶ **Storage autoscaling**

Compute resource

Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

- (●) Don't connect to an EC2 compute resource
  Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.
- ( ) Connect to an EC2 compute resource
  Set up a connection to an EC2 compute resource for this database.

Network type Info

To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

- (●) IPv4
  Your resources can communicate only over the IPv4 addressing protocol.
- ( ) Dual-stack mode
  Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC) Info

Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

vpc-0c9a09cc4c0ef9dd5
3 Subnets, 3 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

After a database is created, you can't change its VPC.

Public access Info

- Yes
  RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.
- No
  RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

VPC security group (firewall) Info

Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.

- Choose existing
  Choose existing VPC security groups
- Create new
  Create new VPC security group

New VPC security group name

new security group

Availability Zone Info

No preference

RDS Proxy

RDS Proxy is a fully managed, highly available database proxy that improves application scalability, resiliency, and security.

- Create an RDS Proxy Info
  RDS automatically creates an IAM role and a Secrets Manager secret for the proxy. RDS Proxy has additional costs. For more information, see Amazon RDS Proxy pricing.

## Database authentication

Database authentication options Info

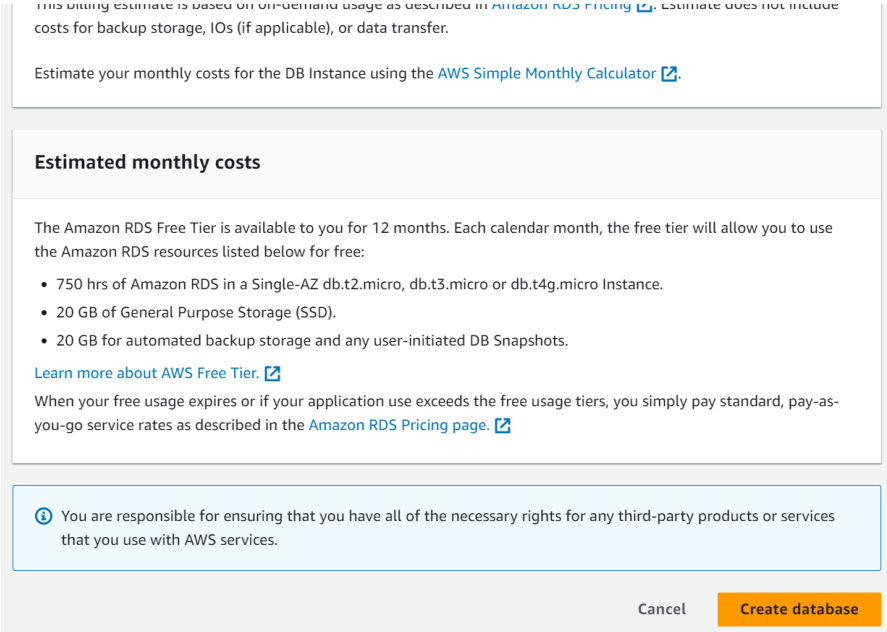
- Password authentication
  Authenticates using database passwords.
- Password and IAM database authentication
  Authenticates using the database password and user credentials through AWS IAM users and roles.
- Password and Kerberos authentication
  Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

## Monitoring

- Enable Enhanced Monitoring
  Enabling Enhanced Monitoring metrics are useful when you want to see how different processes or threads use the CPU.

## Additional configuration

Database options, encryption turned on, backup turned on, backtrack turned off, maintenance, CloudWatch Logs, delete protection turned off.

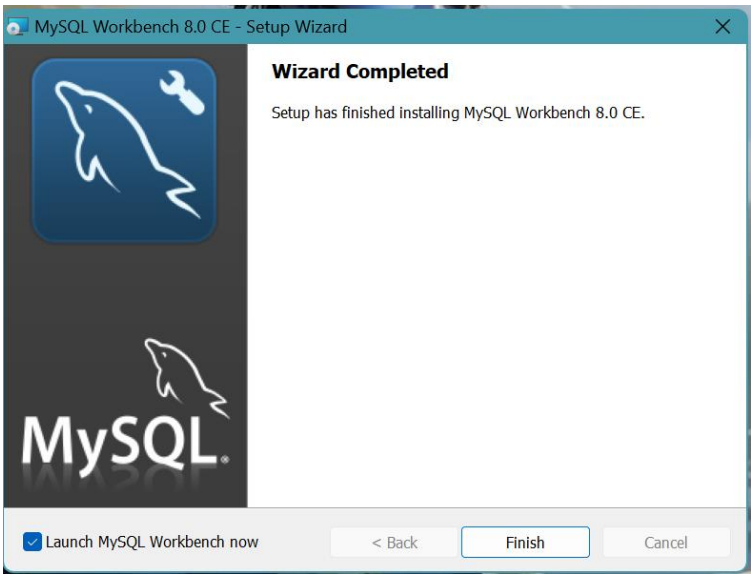


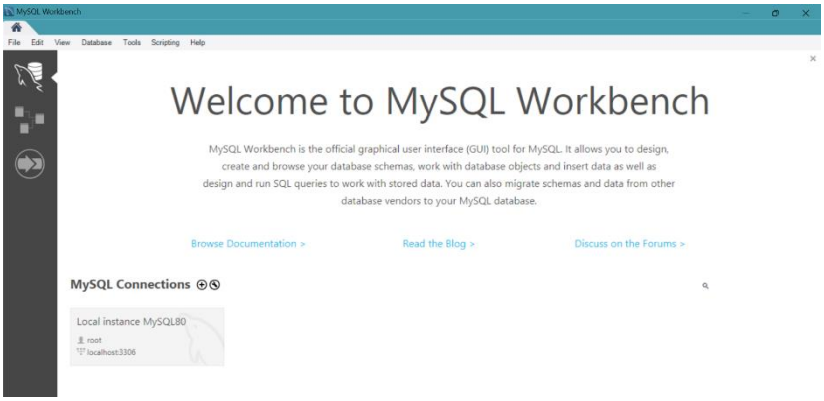

## Configure Database Access

Set up security groups, create database credentials, and configure access policies to ensure secure connectivity to the database.

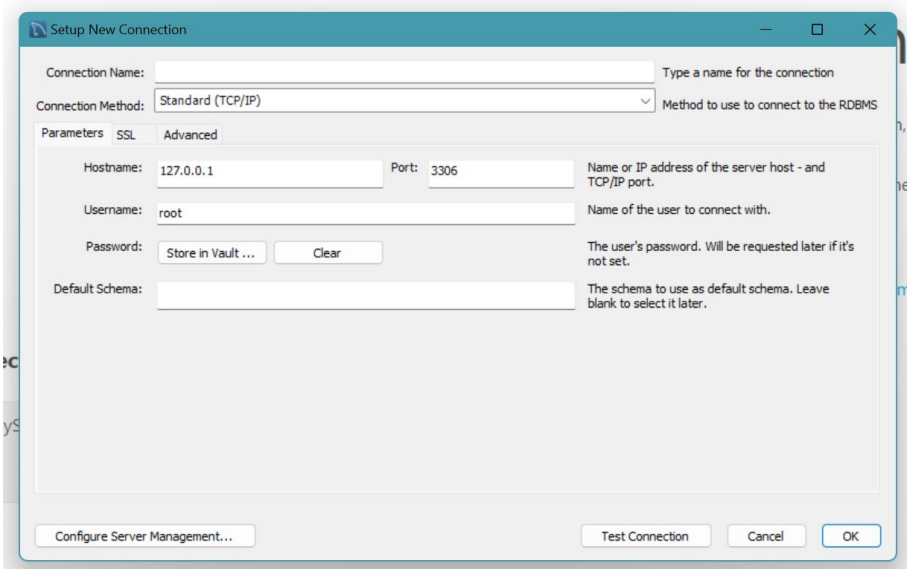
## Install MySQL Workbench

- Download and install MySQL Workbench on your local machine for database management.

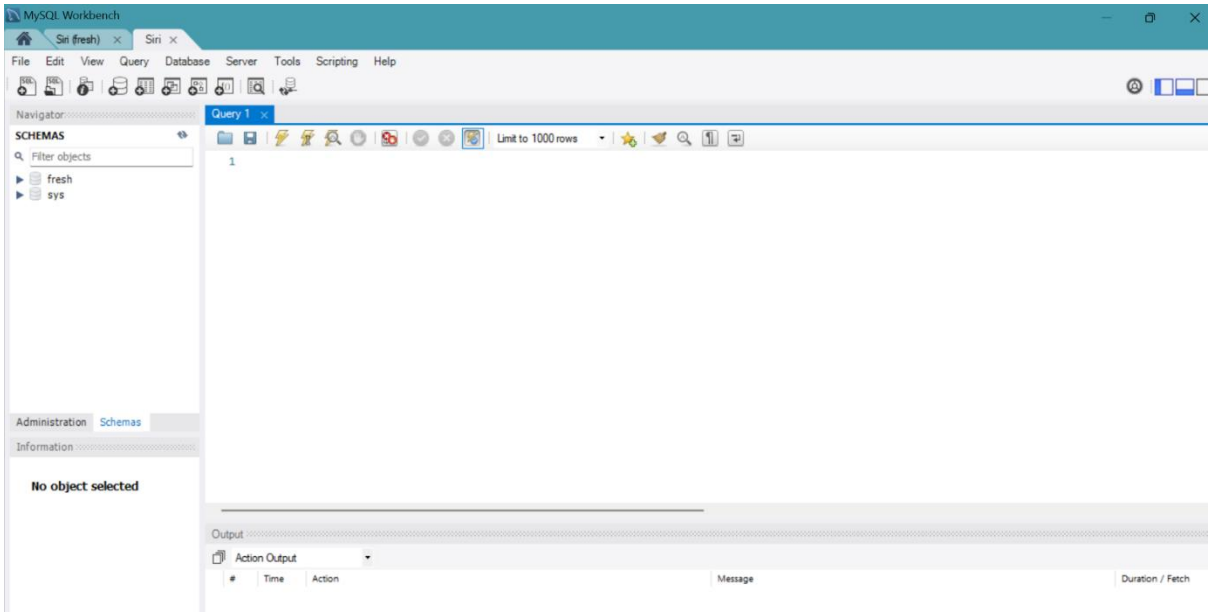





- Connect to the RDS instance via MySQL Workbench using the endpoint and credentials from AW

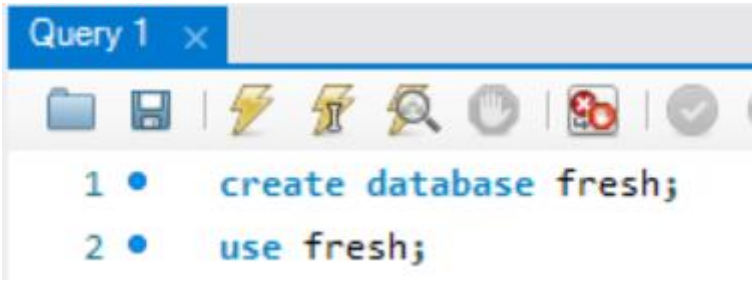



- Give a connection name.
- Copy the endpoint from the RDS database that is created in AWS and paste it in **Hostname.**
- Write the username and enter the password , then click on **Test Connection**.
- Once the connection is successful, you'll be welcomed with this interface




**Create the Database and the tables which are required**

- Create  a basic database schema for an e-commerce platform




Tables Created :

1. users:

- Stores user information such as name, mobile, email, password, and address.
- Each user has a unique ID (id), which is the primary key.

Columns:

- id (Primary Key)

- name, mobile, email, password, address

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    mobile VARCHAR(20),
    email VARCHAR(255),
    password VARCHAR(255),
    address TEXT
);
```

2. items

- Contains available products with fields like item_name and price.
- Each item has a unique identifier item_id and the auto-increment starts from 111.

Columns:

- item_id (Primary Key)
- item_name, price

```sql
CREATE TABLE items (
    item_id INT AUTO_INCREMENT PRIMARY KEY,
    item_name VARCHAR(255) NOT NULL,
    price DECIMAL(10, 2) NOT NULL
) AUTO_INCREMENT = 111;
```

3. orders

- Holds order details including the user_id (foreign key referencing the users table), delivery information, payment method, and total price.
- Each order is assigned a status (e.g., "Yet to Ship", "Shipped") and an order_date.

Columns:

- id (Primary Key)
- user_id (Foreign Key), delivery_address, payment_method, total_price, status, order_date

```sql
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    delivery_address TEXT,
    payment_method VARCHAR(50),
    total_price DECIMAL(10, 2),
    status VARCHAR(50),
    order_date DATETIME,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

4.order_items:

- Stores details of each item in an order. For each order (order_id), the table records the item_name, item_price, and item_quantity.
- The order_id references the orders table.

Columns:

- id (Primary Key)
- order_id (Foreign Key), item_name, item_price, item_quantity

```sql
CREATE TABLE order_items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    item_name VARCHAR(255),
    item_price DECIMAL(10, 2),
    item_quantity INT,
    FOREIGN KEY (order_id) REFERENCES orders(id) ON DELETE CASCADE
);
```

Data Operations:

1. Data Insertion into items: We inserted multiple rows into the items table, representing products available in the shop (like Apple, Banana, etc.) along with their prices.

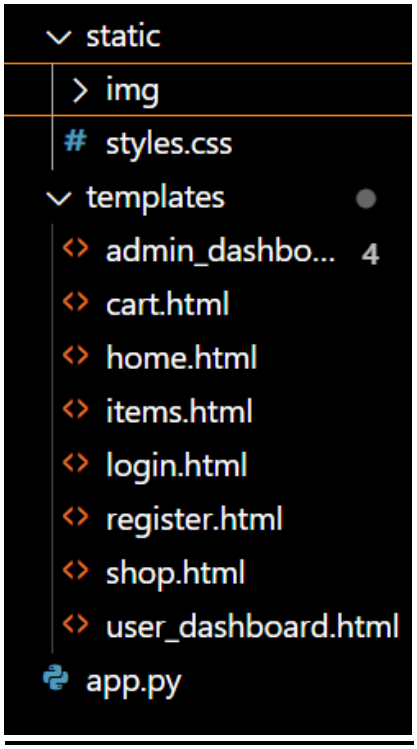
```sql
INSERT INTO items (item_name, price)
VALUES
('Apple', 2.00),
('Banana', 1.50),
('Orange', 1.80),
('Tomato', 1.20),
('Potato', 1.00),
('Onion', 0.80),
('Grapes', 3.00),
('Carrot', 1.10),
('Pepper', 2.20),
('Cucumber', 1.00),
```

# CHAPTER 6

# Frontend Development and Application Setup

## Build the Frontend

- Develop HTML, CSS, and Python-based Flask application files for FreshBasket's frontend interface.




### Integrate Application with RDS

- Connect app.py (Flask application) to the MySQL RDS database by configuring database connection settings and verifying connectivity.

Description of the code :

1. Flask App Initialization: Initializes a Flask application with secret key for sessions.

```
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
import mysql.connector
from datetime import datetime
import mysql.connector.pooling

app = Flask(__name__)
app.secret_key = "your_secret_key"  # Needed for flash messages
```

2. Database Configuration: Configures MySQL RDS with connection pooling for efficient database access.

```
db_config = {
    'host': 'db-instance-commerce.cfwim48wwhiq.ap-south-1.rds.amazonaws.com',  # Your RDS endpoint
    'user': 'admin',  # Your DB username
    'password': 'Siri#123456',  # Your DB password
    'database': 'fresh'  # Your DB name
}
```

3. Connection Pool: Uses MySQL connection pooling to handle multiple database connections.

```
# Connection pool setup
cnxpool = mysql.connector.pooling.MySQLConnectionPool(pool_name="mypool",
                                                      pool_size=5,
                                                      **db_config)

# Function to establish a database connection
def get_db_connection():
    try:
        return cnxpool.get_connection()
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None
```

4. Home Route: Renders the home page template when the root URL is accessed.
5. Register Route (GET/POST): Handles user registration, inserts user data into the database.

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

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form.get('name')
        mobile = request.form.get('mobile')
        email = request.form.get('email')
        password = request.form.get('password')
        default_address = request.form.get('default_address')

        if not default_address:
            flash('Default address is required!')
            return redirect(url_for('register'))

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
            'INSERT INTO users (name, mobile, email, password, address) VALUES (%s, %s, %s, %s, %s)',
            (name, mobile, email, password, default_address))
        conn.commit()
        cursor.close()
        conn.close()

        flash('Thank you for registering!')
        return redirect(url_for('login'))

    return render_template('register.html')
```

6. Login Route (GET/POST): Authenticates user with email and password, creates session on success.

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

        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute('SELECT * FROM users WHERE email = %s AND password = %s', (email, password))
        user = cursor.fetchone()
        cursor.close()
        conn.close()

        if user:
            session['user_id'] = user['id']
            session['user_name'] = user['name']
            flash('Login successful!')
            return redirect(url_for('shop'))
        else:
            flash('Invalid email or password!')

    return render_template('login.html')
```

7. Shop Route: Renders the shop page, showing available products to logged-in users.
8. Add to Cart (POST): Adds selected items to the user's session-based shopping cart.

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

@app.route('/add_to_cart', methods=['POST'])
def add_to_cart():
    item_data = request.get_json()
    item_name = item_data['name']
    item_price = item_data['price']
    item_quantity = item_data['quantity']

    cart_items = session.get('cart_items', [])

    # Check if the item is already in the cart
    item_found = False
    for item in cart_items:
        if item['name'] == item_name:
            item['quantity'] += item_quantity
            item_found = True
            break

    if not item_found:
        cart_items.append({
            'name': item_name,
            'price': item_price,
            'quantity': item_quantity
        })

    session['cart_items'] = cart_items
    return jsonify(success=True)
```

**9.** Items Route (GET/POST): Displays available items and allows adding them to the shopping cart.

```python
@app.route('/items', methods=['GET', 'POST'])
def items():
    if request.method == 'POST':
        # Handle adding items to the cart in session
        item_name = request.form.get('name')
        item_price = float(request.form.get('price'))
        item_quantity = int(request.form.get('quantity'))

        cart_items = session.get('cart_items', [])

        # Check if item already exists in the cart
        for item in cart_items:
            if item['name'] == item_name:
                item['quantity'] += item_quantity
                break
        else:
            cart_items.append({'name': item_name, 'price': item_price, 'quantity': item_quantity})

        session['cart_items'] = cart_items
        flash(f'{item_name} added to your cart!')
        return redirect(url_for('items'))

    # Fetch items from the database for display
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT item_id, item_name, price FROM items')
    items = cursor.fetchall()
    cursor.close()
    conn.close()

    cart_items = session.get('cart_items', [])
    return render_template('items.html', items=items, cart_items=cart_items)
```

**10.** Place Order Route (POST): Places an order, inserts order and item data into the database.

```python
@app.route('/place_order', methods=['POST'])
def place_order():
    if 'user_id' not in session:
        return jsonify(success=False, message="User not logged in")
    data = request.get_json()
    delivery_address = data.get('address', 'Default Address')
    payment_method = data['payment_method']
    items = data['items']
    total_price = data['total_price']
# Insert order into the database with error handling
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
            'INSERT INTO orders (user_id, delivery_address, payment_method, status, order_date, total_price) '
            'VALUES (%s, %s, %s, %s, %s, %s)',
            (session['user_id'], delivery_address, payment_method, 'Yet to Ship', datetime.now(), total_price)
        )
        order_id = cursor.lastrowid
        for item in items:
            cursor.execute(
                'INSERT INTO order_items (order_id, item_name, quantity, price) '
                'VALUES (%s, %s, %s, %s)',
                (order_id, item['name'], item['quantity'], item['price'])
            )
        conn.commit()  # Commit the transaction after all queries are successful
        cursor.close()
        conn.close()
        return jsonify(success=True)
    except Exception as e:
        conn.rollback()  # Rollback the transaction in case of error
        return jsonify(success=False, message=str(e))
```

**11.** User Dashboard Route: Displays user-specific order history with item details and statuses**.**

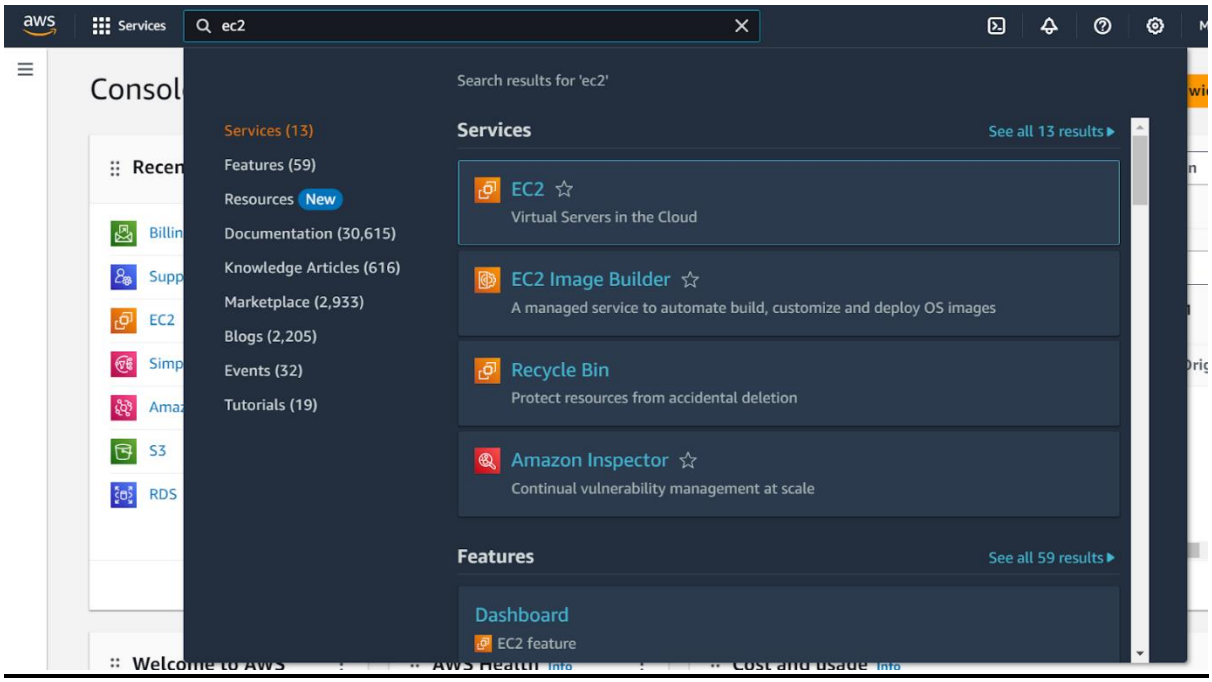
```python
@app.route('/user_dashboard')
def user_dashboard():
    if 'user_id' not in session:
        flash('You need to log in to access your dashboard!')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch the orders and order items for the current user
    cursor.execute('''
        SELECT o.id, o.total_price, o.status, o.order_date,
               GROUP_CONCAT(CONCAT(oi.item_name, ' (x', oi.item_quantity, ')')) AS items
        FROM orders o
        JOIN order_items oi ON o.id = oi.order_id
        WHERE o.user_id = %s
        GROUP BY o.id
    ''', (session['user_id'],))
    orders = cursor.fetchall()  # Fetch all orders for the user

    cursor.close()
    conn.close()

    return render_template('user_dashboard.html', orders=orders)
```

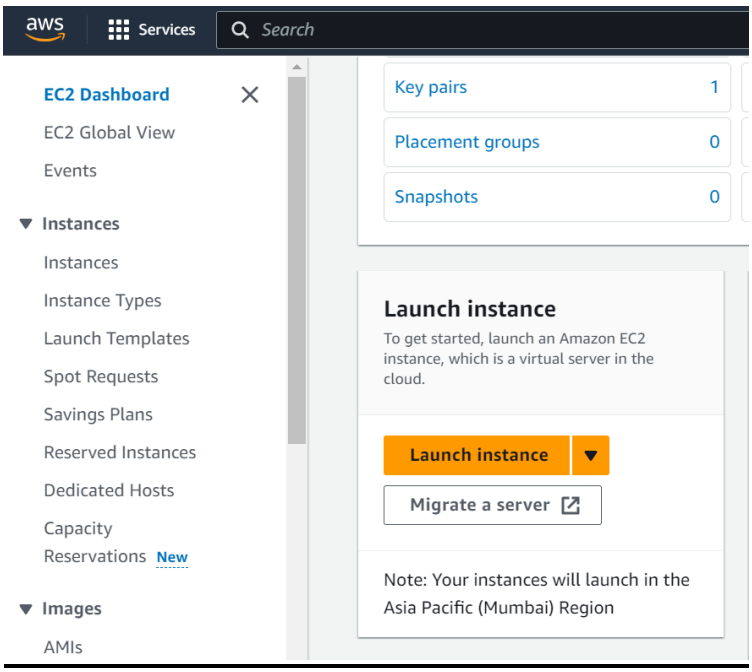
# CHAPTER 7

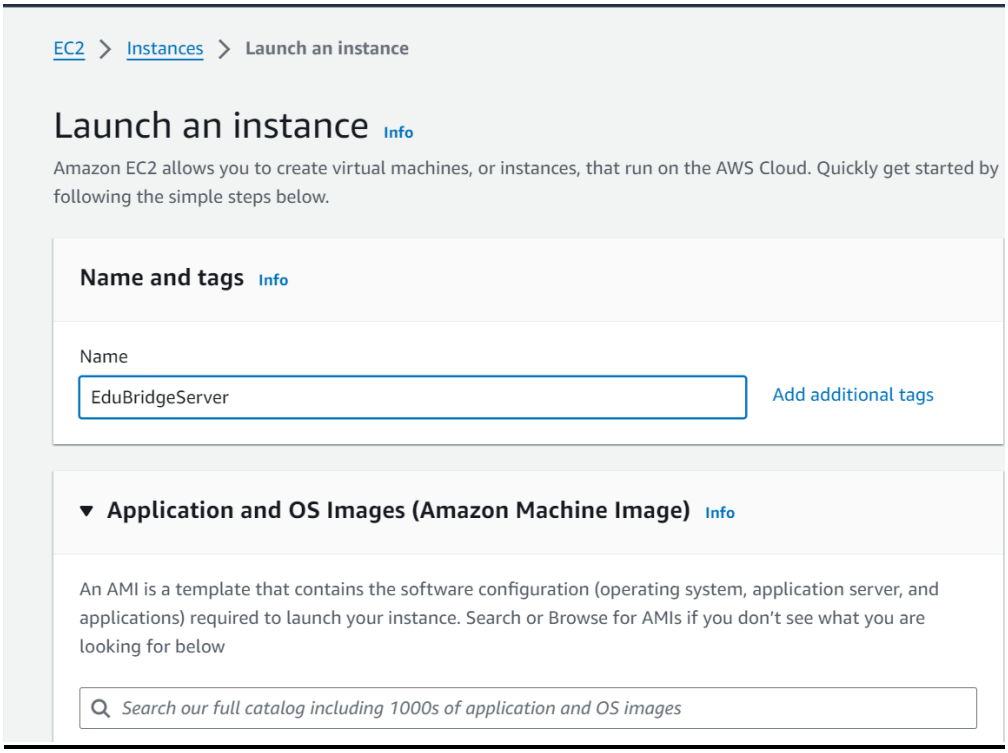
# EC2 Instance Setup

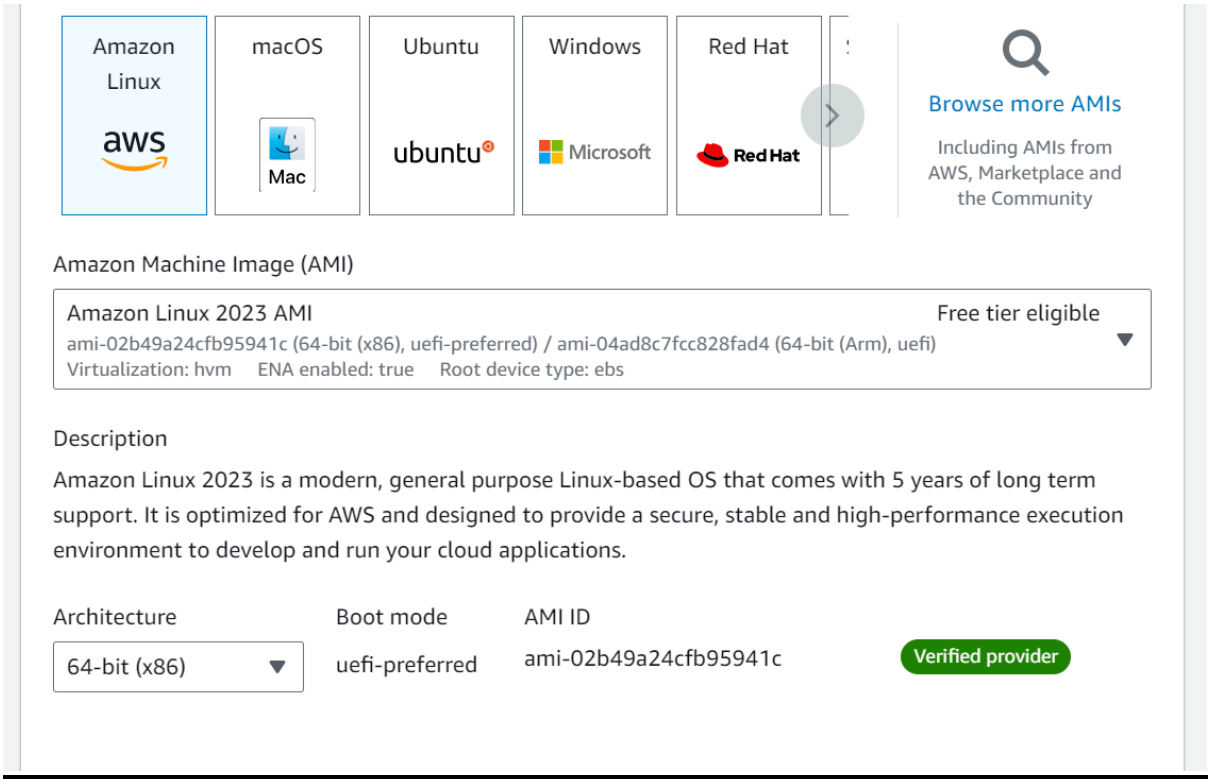
## Launch EC2 Instance

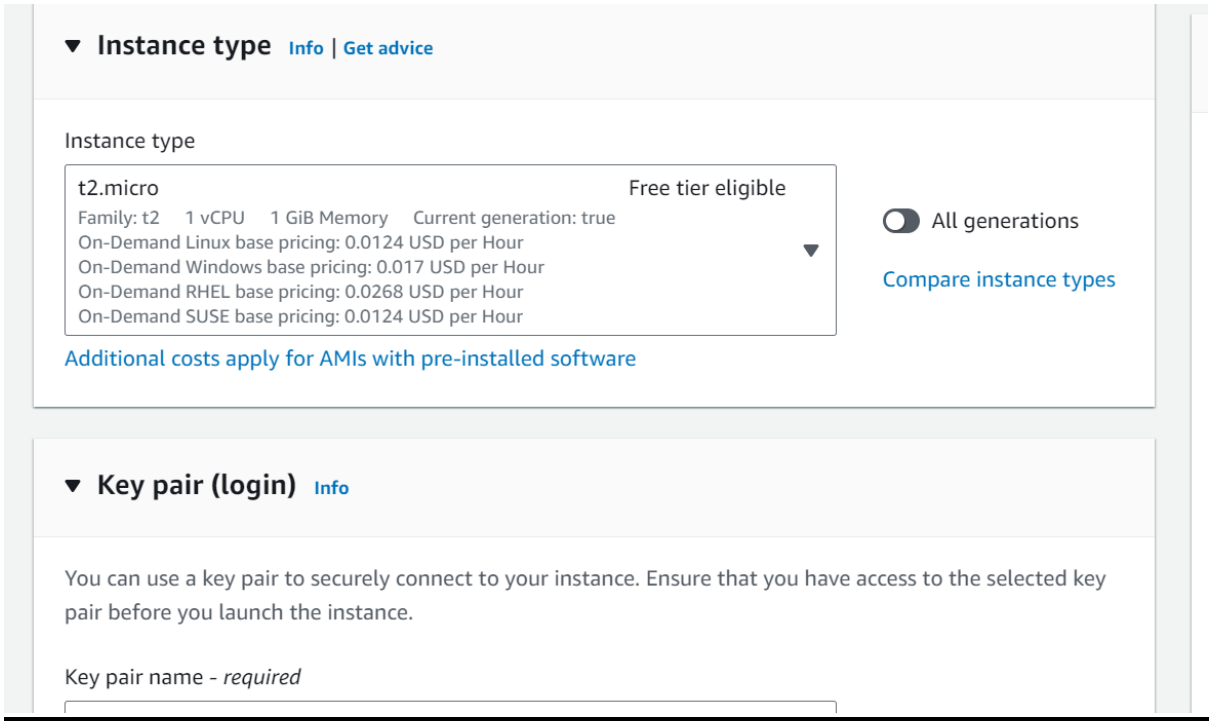
- Choose a Linux-based EC2 instance from the AWS Console to host the FreshBasket application.

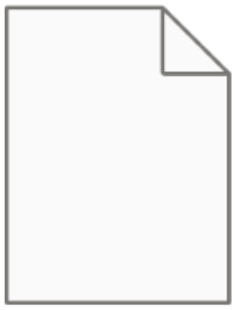



EC2 > Instances > Launch an instance
Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
EduBridgeServer
Add additional tags
Application and OS Images (Amazon 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
Search our full catalog including 1000s of application and OS images


Amazon Linux
macOS
Mac
Ubuntu
Windows
Microsoft
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
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

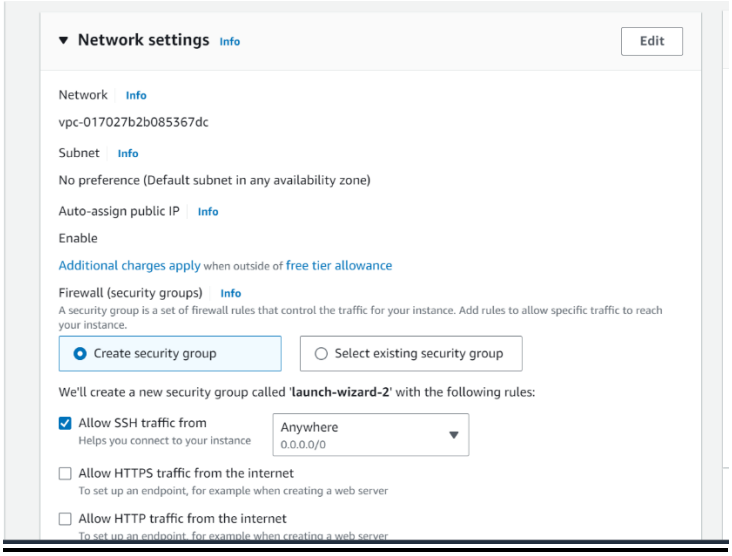


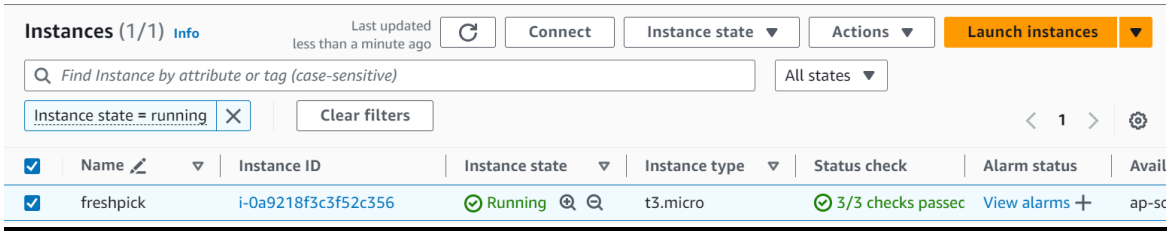
freshpick.pem

## Configure Network Settings

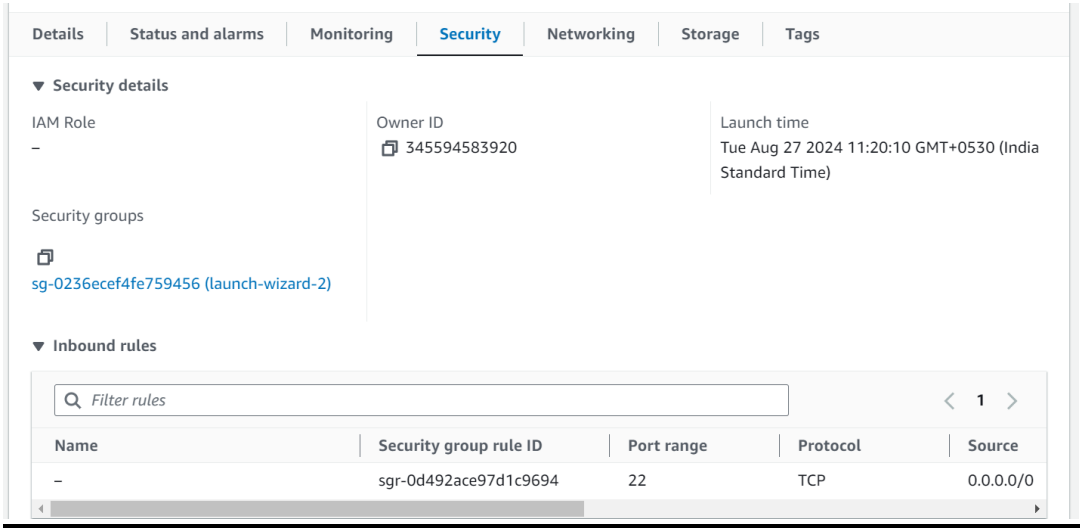
- Set up the security group to allow HTTP, HTTPS, and SSH traffic.

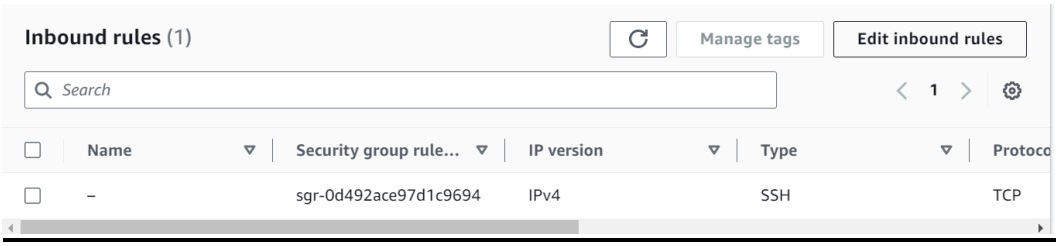


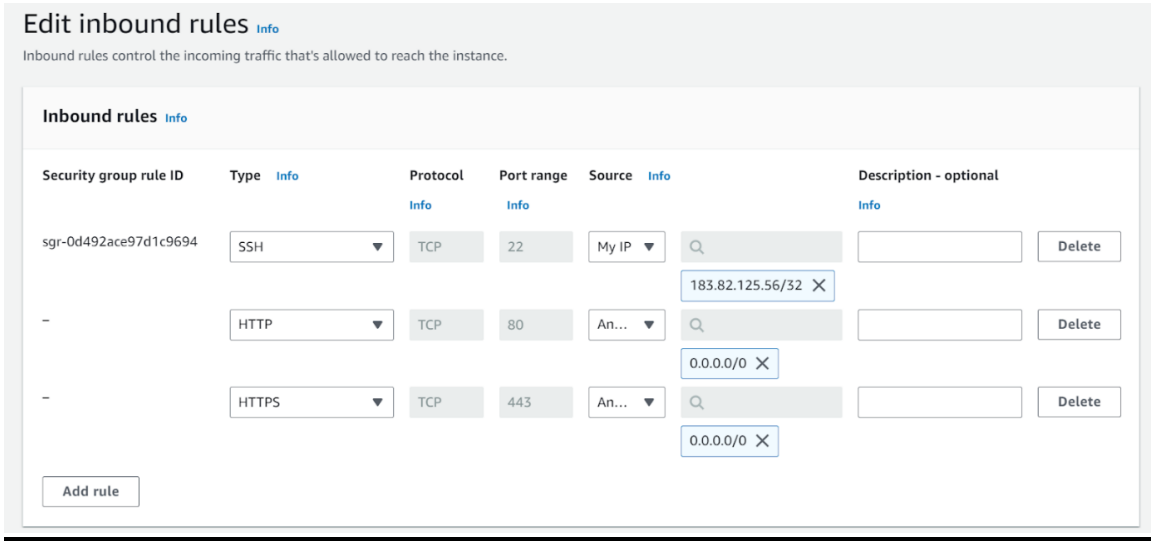
- Create and download the key pair for SSH access.



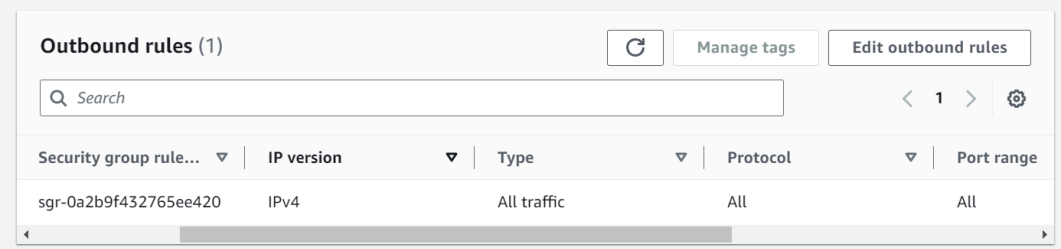

## Setting up Inbound and Outbound rules



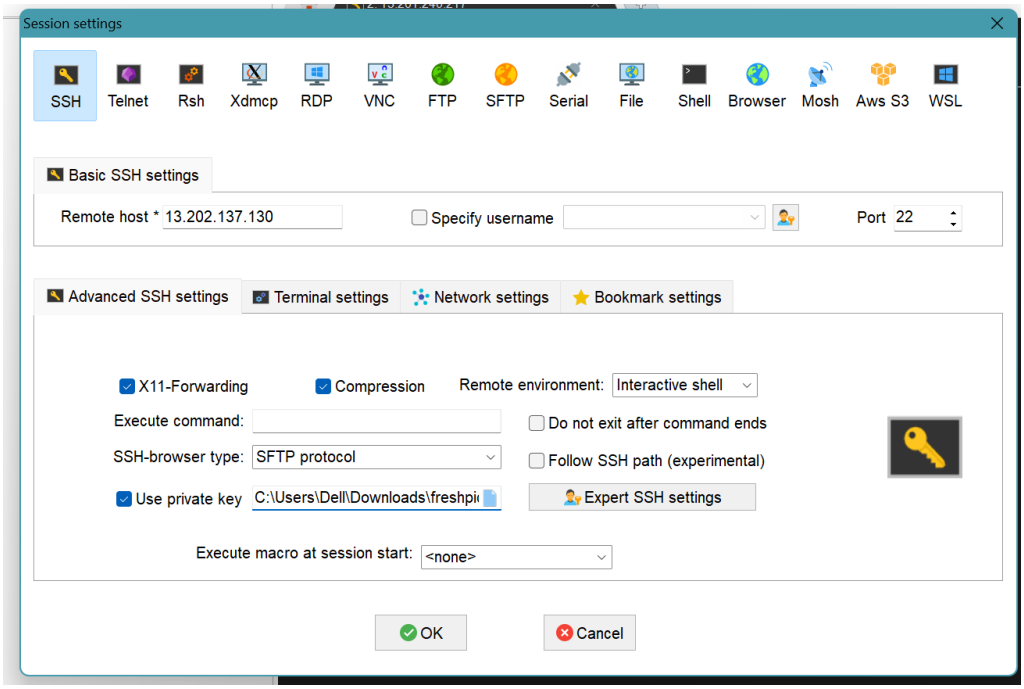



- Add Type : HTTP > Source : Anywhere
- Add Type : HTTPS > Source : Anywhere

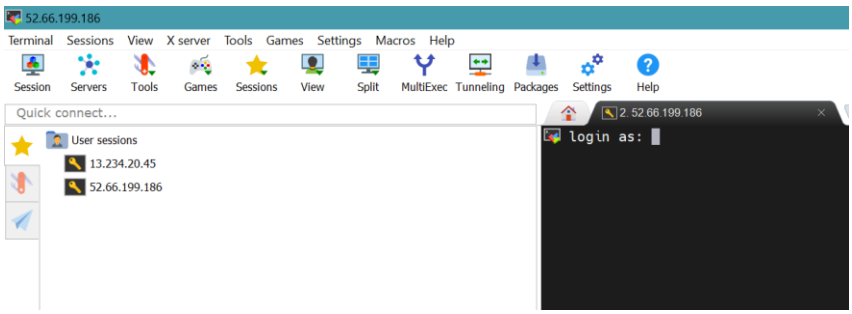



## MobaXterm Setup and SSH Access

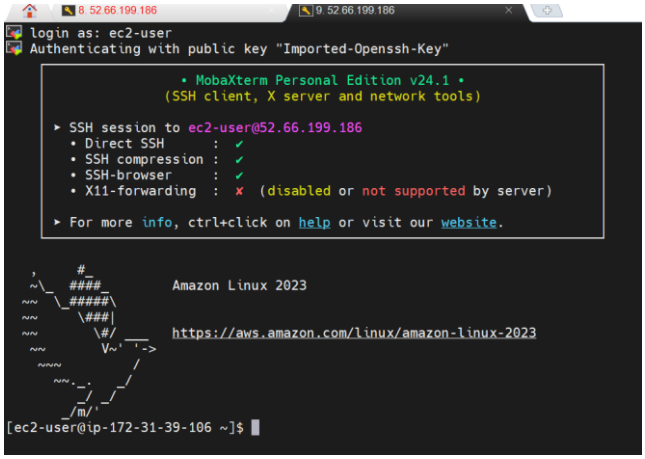
### Install and Configure MobaXterm




### Login into the SSH Session




- Login as : ec2-user




Update and Install Web Server Software

- Update package lists with sudo apt-get update (Ubuntu) or sudo yum update (Amazon Linux).

Install Apache or Nginx:

- For Apache: sudo apt-get install apache2 (Ubuntu) or sudo yum install httpd (Amazon Linux).
- For Nginx: sudo apt-get install nginx (Ubuntu) or sudo yum install nginx (Amazon Linux).

```
[ec2-user@ip-172-31-39-106 ~]$ sudo yum install httpd -y
Last metadata expiration check: 0:23:00 ago on Tue Aug 27 05:50:54 2024.
Dependencies resolved.
================================================================================
 Package                Arch        Version                    Repository      Siz
================================================================================
Installing:
 httpd                  x86_64      2.4.62-1.amzn2023          amazonlinux      48
Installing dependencies:
 apr                    x86_64      1.7.2-2.amzn2023.0.2       amazonlinux     129
 apr-util               x86_64      1.6.3-1.amzn2023.0.1       amazonlinux      98
 generic-logos-httpd    noarch      18.0.0-12.amzn2023.0.3     amazonlinux      19
 httpd-core             x86_64      2.4.62-1.amzn2023          amazonlinux     1.4
 httpd-filesystem       noarch      2.4.62-1.amzn2023          amazonlinux      14
 httpd-tools            x86_64      2.4.62-1.amzn2023          amazonlinux      81
 libbrotli              x86_64      1.0.9-4.amzn2023.0.2       amazonlinux     315
 mailcap                noarch      2.1.49-3.amzn2023.0.3      amazonlinux      33
Installing weak dependencies:
 apr-util-openssl       x86_64      1.6.3-1.amzn2023.0.1       amazonlinux      17
 mod_http2              x86_64      2.0.27-1.amzn2023.0.3      amazonlinux     166
 mod_lua                x86_64      2.4.62-1.amzn2023          amazonlinux      61

Transaction Summary
================================================================================
Install  12 Packages

Total download size: 2.3 M
Installed size: 6.9 M
Downloading Packages:
(1/12): apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64.rpm    258 kB/s |  17 kB     00:00
(2/12): apr-1.7.2-2.amzn2023.0.2.x86_64.rpm                 1.8 MB/s | 129 kB     00:00
(3/12): apr-util-1.6.3-1.amzn2023.0.1.x86_64.rpm            1.3 MB/s |  98 kB     00:00
(4/12): generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch.r 1.0 MB/s |  19 kB     00:00
(5/12): httpd-2.4.62-1.amzn2023.x86_64.rpm                  2.6 MB/s |  48 kB     00:00
(6/12): httpd-core-2.4.62-1.amzn2023.x86_64.rpm              35 MB/s | 1.4 MB     00:00
```

```
Complete!
[ec2-user@ip-172-31-39-106 ~]$ sudo service httpd start
Redirecting to /bin/systemctl start httpd.service
[ec2-user@ip-172-31-39-106 ~]$
```

```
[ec2-user@ip-172-31-39-106 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-39-106 ~]$
```

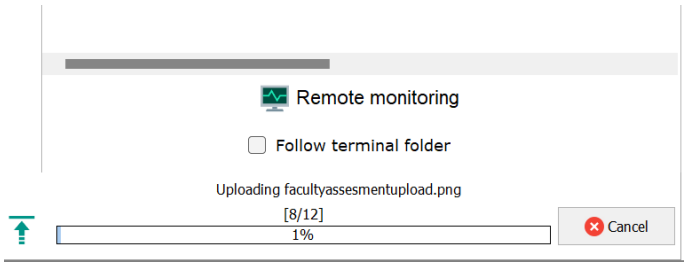
```
[ec2-user@ip-172-31-39-106 ~]$ cd /var/www/html
[ec2-user@ip-172-31-39-106 html]$ pwd
/var/www/html
[ec2-user@ip-172-31-39-106 html]$
```

Upload Website Files Using MobaXterm

- Use MobaXterm's SFTP functionality to transfer website files to the EC2 instance.
- Navigate to the /var/www/html directory (or the relevant directory for Nginx) and upload the files.

```
[ec2-user@ip-172-31-39-106 ~]$ cd /var/www/html
[ec2-user@ip-172-31-39-106 html]$ pwd
/var/www/html
[ec2-user@ip-172-31-39-106 html]$
```

```
/var/www/html
[ec2-user@ip-172-31-39-106 html]$ sudo chmod 777 /var/www/html
[ec2-user@ip-172-31-39-106 html]$
```




Launch Flask Application

- Run the Flask app on the EC2 instance through the SSH session to serve the FreshBasket fronte

# CHAPTER 8

# Testing and Deployment

## Functional Testing

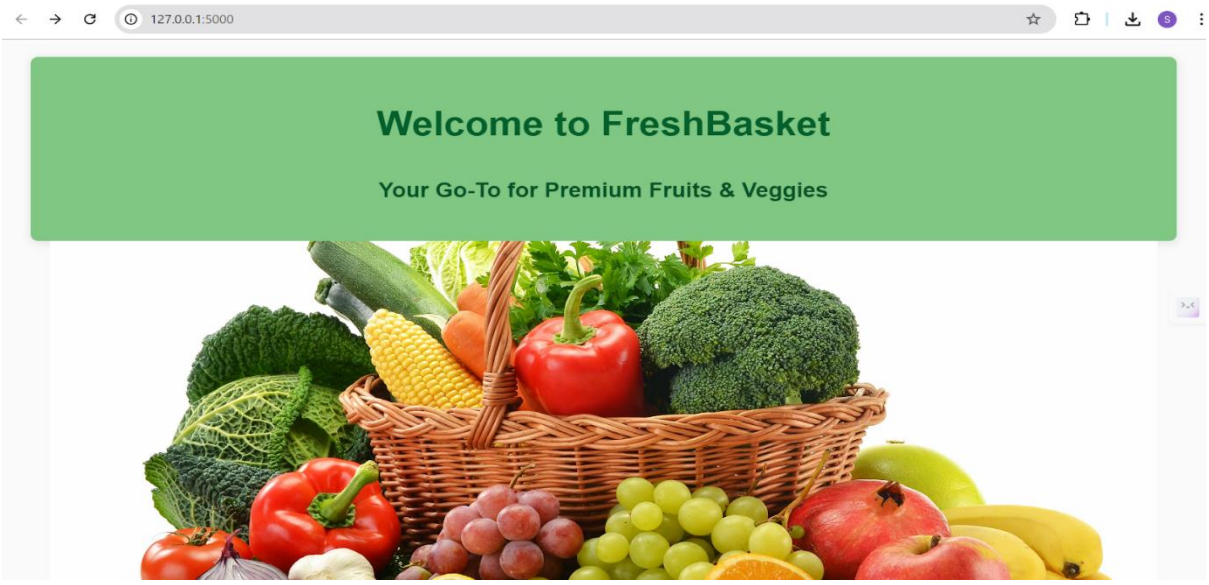
- Test the FreshBasket application for functionality, including database interactions and frontend features.
- Run the Flask app python3 app.py
- It will give you the link
  - 
```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 849-618-898
```

## Deployment

- Deploy the application in a production environment, ensuring high availability and performance.

Click on the link above and it will take you to the webpage:

## home:



100% Organic and Fresh Produce

Locally Sourced from Trusted Farms

Eco-Friendly Packaging

Excellent Customer Service

Affordable Prices for Premium Quality

Happy Customers

Happy Customers

"FreshBasket always delivers high-quality produce. I'm a regular customer and highly recommend it!" - Jane D.

"I love the variety and freshness of the fruits and vegetables. The service is outstanding too!" - John S.

"Great place to shop for organic produce. The prices are fair and the quality is top-notch." - Emily R.

Admin Dashboard

# Register:

Register

Home

Name:

Siri

Mobile:

9898989898

Email:

siri@gmail.com

Password:

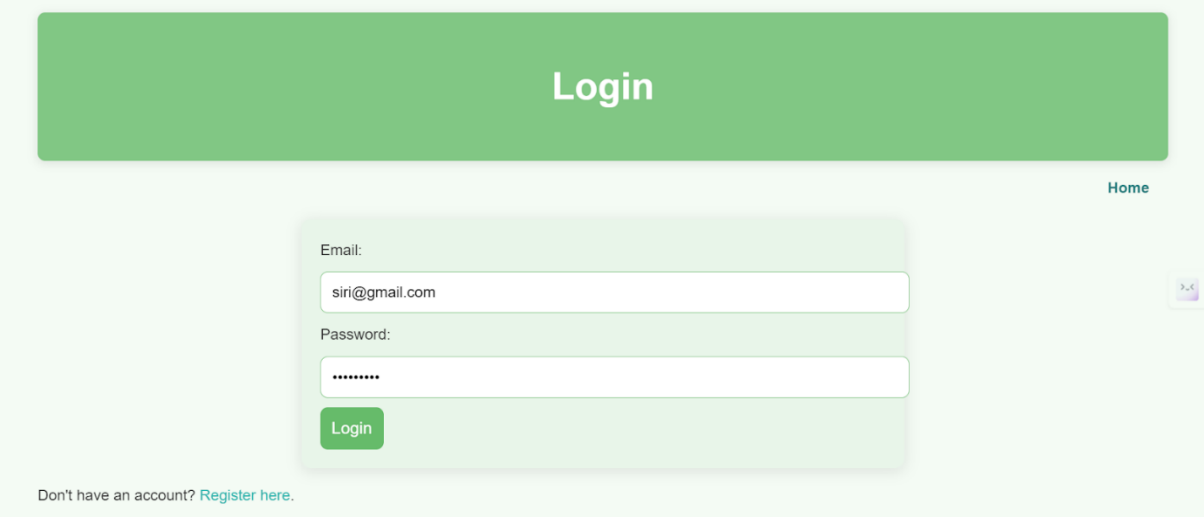
Default Address:

2a/a, Happy Homes, Gachibowli

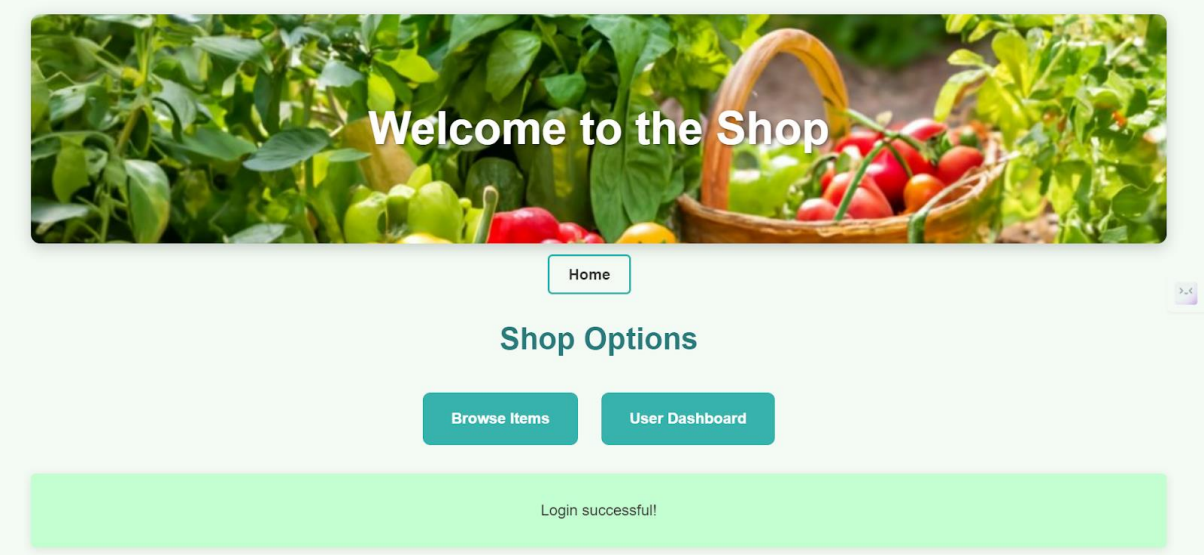
Register

Already registered? Login here.

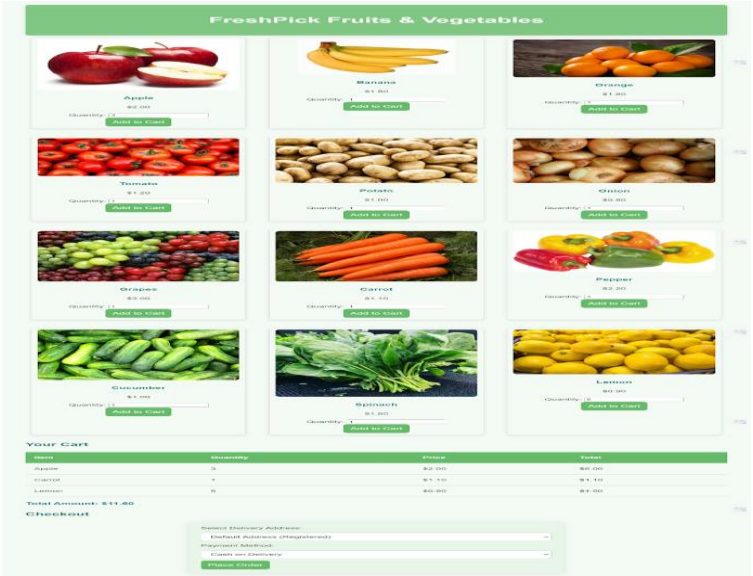
## Login:




## Shop:




## Items:



## Your Cart

| Item | Quantity | Price | Total |
|---|---|---|---|
| Apple | 3 | $2.00 | $6.00 |
| Carrot | 1 | $1.10 | $1.10 |
| Lemon | 5 | $0.90 | $4.50 |

**Total Amount: $11.60**

## Checkout

Select Delivery Address:

Default Address (Registered)

Payment Method:

Cash on Delivery

Place Order

# User Dashboard:

127.0.0.1:5000/user_dashboard

## Your Previous Orders

| Order ID | Items | Total Price | Date | Status |
|---|---|---|---|---|
| 3 | <built-in method items of dict object at 0x000002230E5FC880> | $0.00 | | Yet to Ship |
| 4 | <built-in method items of dict object at 0x000002230E5FF240> | $0.00 | | Yet to Ship |

# Admin Dashboard:

127.0.0.1:5000/admin_dashboard

Paused

## Manage Orders

| Order ID | User | Items | Total Price | Status | Update Status |
|---|---|---|---|---|---|
| 6 | Siri | <built-in method items of dict object at 0x0000020F031E4140> | $4.00 | Yet to Ship | Yet to Ship |
| 7 | Siri | <built-in method items of dict object at 0x0000020F031E4240> | $1.50 | Yet to Ship | Yet to Ship |
| 8 | Siri | <built-in method items of dict object at 0x0000020F031E4280> | $6.00 | Yet to Ship | Yet to Ship |
| 9 | Siri | <built-in method items of dict object at 0x0000020F031E4300> | $2.00 | Yet to Ship | Yet to Ship |

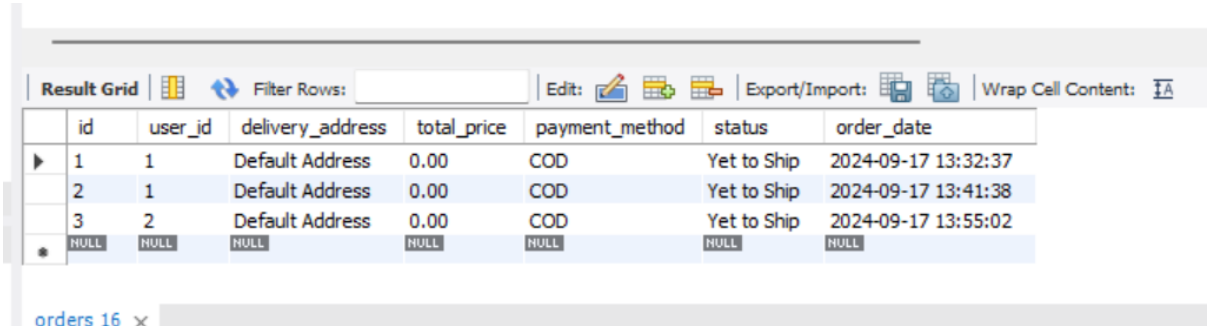
## Check the updations in the mysql database.

MySQL Database updating**:**

1. **Users table:**

| id | name | mobile | email | password | address |
|---|---|---|---|---|---|
| 1 | Siri Chakkala | 878787878 | siri@gmail.com | 123456789 | Shamshabad |
| 2 | Sri | 8989898989 | sri@gmail.com | 123456789 | Hyderabad |
| 3 | Raju | 898989898 | raju@gmail.com | 0000000000 | Kphb |
| 4 | Surya | 7878787878 | surya@gmail.com | 090909090 | Secunderabad |
| NULL | NULL | NULL | NULL | NULL | NULL |

2. **Orders table:**




| id | user_id | delivery_address | total_price | payment_method | status | order_date |
|---|---|---|---|---|---|---|
| 1 | 1 | Default Address | 0.00 | COD | Yet to Ship | 2024-09-17 13:32:37 |
| 2 | 1 | Default Address | 0.00 | COD | Yet to Ship | 2024-09-17 13:41:38 |
| 3 | 2 | Default Address | 0.00 | COD | Yet to Ship | 2024-09-17 13:55:02 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

3. **Items table:**

| item_id | item_name | price |
|---|---|---|
| 111 | Apple | 2.00 |
| 112 | Banana | 1.50 |
| 113 | Orange | 1.80 |
| 114 | Tomato | 1.20 |
| 115 | Potato | 1.00 |
| 116 | Onion | 0.80 |
| 117 | Grapes | 3.00 |
| 118 | Carrot | 1.10 |
| 119 | Pepper | 2.20 |
| 120 | Cucumber | 1.00 |
| 121 | Spinach | 1.50 |
| 122 | Lemon | 0.90 |
| NULL | NULL | NULL |

# CHAPTER 9

# Monitoring and Optimization

**Monitoring in E-Commerce**

1. Website Performance:
    - Track uptime and page load speeds using tools like Google Analytics, Pingdom, or New Relic.
    - Monitor error rates, user navigation, and bounce rates.
2. Customer Behavior:
    - Use tools like Hotjar or Crazy Egg for heatmaps and session recordings.
    - Monitor cart abandonment rates and on-site search tre

**Optimization in E-Commerce**

1. Improving User Experience:
    - Optimize website design for mobile responsiveness and ease of navigation.
    - Implement predictive search to improve product discovery.
2. Boosting Conversions:
    - Use personalized recommendations based on user behavior.
    - Simplify the checkout process and support multiple payment options.

## REFERENCE:

**GITHUB LINK**

https://github.com/Gopika1674 /Naan-Mudhalvan.git

# CHAPTER 10

# Challenges and Solutions

### 1. Website Performance and Scalability

**Challenges**:

- Slow loading times, especially during high traffic.
- Downtime during promotional events or peak seasons.

**Solutions**:

- Use Content Delivery Networks (CDNs) like Cloudflare to optimize load speeds.
- Implement scalable hosting solutions (e.g., AWS, Azure) with auto-scaling capabilities.

### 2. Cart Abandonment

**Challenges**:

- Customers leave without completing their purchases.
- Common reasons include high shipping costs, complicated checkout, or lack of payment options.

**Solutions**:

- Offer free or transparent shipping costs upfront.
- Simplify the checkout process to reduce friction (e.g., one-page checkout

### 3. Inventory Management

**Challenges**:

- Overstocking or running out of popular items.
- Delayed supplier deliveries affecting customer satisfaction.

**Solutions**:

- Use real-time inventory management software like TradeGecko or Odoo.
- Analyze historical data to forecast demand.

# CHAPTER 11

# FUTURE ENHANCEMENTS

Future enhancements for your E-commerce project can focus on innovation, customer experience, scalability, and leveraging new technologies. Here are some ideas:

**1. Advanced Personalization**

- **Dynamic Product Recommendations**: Use AI to suggest products based on browsing and purchase history.
- **Customizable User Dashboards**: Allow customers to tailor their experience, such as saved searches or preferred layouts.
- **Personalized Pricing**: Offer discounts or pricing based on customer loyalty or buying patterns.

**2. Enhanced Customer Engagement**

- **AI Chatbots**: Provide 24/7 customer support using chatbots with advanced natural language understanding.
- **Gamification**: Add reward systems for purchases, reviews, or social media shares (e.g., points, badges).
- **Community Building**: Integrate forums, Q&A sections, or social feeds to engage users around your products.

**3. Improved Checkout Process**

- **One-Click Checkout**: Similar to Amazon, streamline the purchase process.
- **Cryptocurrency Payments**: Accept Bitcoin or other cryptocurrencies to appeal to tech-savvy customers.
- **Buy Now, Pay Later (BNPL)**: Partner with providers like Afterpay or Klarna for flexible payment options.

# CHAPTER 12

# CERTIFICATES



aws academy
Gopika S
Certificate of Completion for
AWS Academy Graduate - AWS Academy Cloud Foundations
Course hours completed
20 hours
Issued on
11/07/2024
Digital badge
https://www.credly.com/go/3O0IgWCX


aws academy
Priya Darshini
Certificate of Completion for
AWS Academy Graduate - AWS Academy Cloud Foundations
Course hours completed
20 hours
Issued on
11/08/2024
Digital badge
https://www.credly.com/go/fNPcrONy