FreshBasket : Scalable E-Commerce Platform Deployment with Flask on AWS EC2 and RDS

Project Documentation

RRASECOLLEGEOFENGINEERING

Affiliated to Anna University

Department of Information Technology

Team Members:

1.Santhosh Kumar	411822243019
2.Blesson B	411822243006
3.Parthiban R	411822243015
4.Haritha S	411822243007

Submitted to Naan Mudhalavan Team

ANNA UNIVERTSITY.

UNDER THE GUIDANCE OF

Mrs G.Gayathri

RRASE COLLEGE OF ENGINEERING

PADAPPAI, CHENNAI.

DEPARTMENT OF INFORMATION TECHNOLOGY

NM1028- AWS CLOUD PRACTIONER

2021 REGULATION

Name : _____

Reg.No. : _____

Branch : _____

Year : _____

Semester : _____

RRASE COLLEGE OF ENGINEERING

PADAPPAI, CHENNAI.

LABORATORY RECORD

UNIVERSITY REGISTER NO. | |

Certified that this is the bonafide record of work done by

Mr. /Ms. _____ **of** _____

Department in the _____ **Laboratory**

and submitted for University Practical Examination conducted on _____

at RRASE COLLEGE OF ENGINEERING -601301.

Lab In-charge **Principal** **Head of the Department**

External Examiner **Internal Examiner**

Table of content

FreshBasket : Scalable E-Commerce Platform Deployment with Flask on AWS EC2 and RDS

1.scenario

Scenario 1: Scalable Web Applications for Online Retail

In online retail scenarios, AWS EC2 provides a scalable infrastructure that can adapt to varying levels of customer traffic. For example, an online electronics store could use EC2 to handle peak shopping periods like Black Friday, ensuring that the website remains responsive and available. By leveraging Flask for backend development, the store can manage user sessions, product catalogs, and order processing efficiently. This setup allows the retail platform to grow and handle increased user demand without performance degradation.

Scenario 2: Efficient Database Management for SaaS Platforms

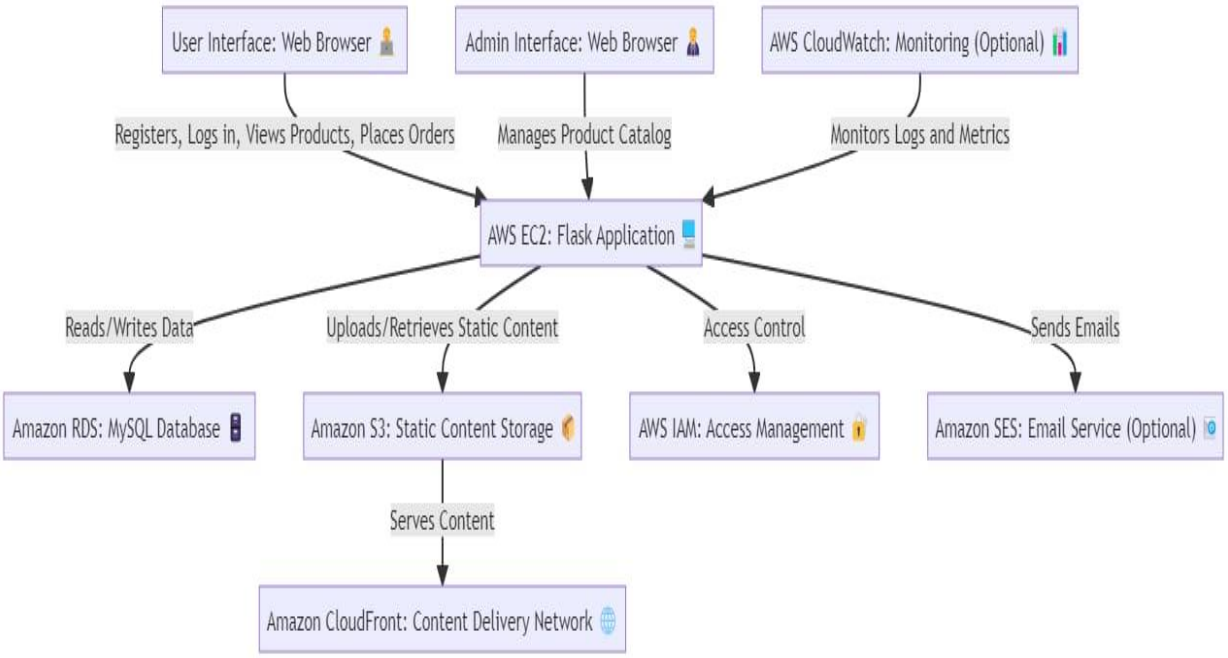
For Software-as-a-Service (SaaS) platforms, Amazon RDS offers a managed database solution that simplifies operations and enhances performance. Consider a SaaS application for project management where RDS handles user data, project details, and task histories. By utilizing RDS with MySQL, the platform benefits from automated backups, easy scaling, and high availability. This ensures that the application can support a growing user base while maintaining data integrity and reliability.

Scenario 3: Secure and Scalable Hosting for Health Tech Application

In health tech applications, AWS EC2 can provide a secure and scalable environment for hosting applications that manage patient records, appointment scheduling, and telemedicine services. By integrating IAM (Identity and Access Management) for secure access control, the platform ensures that only authorized personnel can

access sensitive health data. EC2's scalability allows the application to handle varying levels of user activity and data processing needs, supporting a high level of performance and security.

2.Architecture



Pre-reqisites:

1. AWS Account Setup:
   https://youtu.be/CjKhQoYeR4Q?si=nSh6eyilYHoqnAoI
2. Understanding of IAM:
   https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR
3. Knowledge of Amazon EC2 :
   https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz

4. Mobaxtream : https://youtu.be/dvoU2SKG6oA?si=Hs8Pu4Crry5-BRrD

5. RDS :

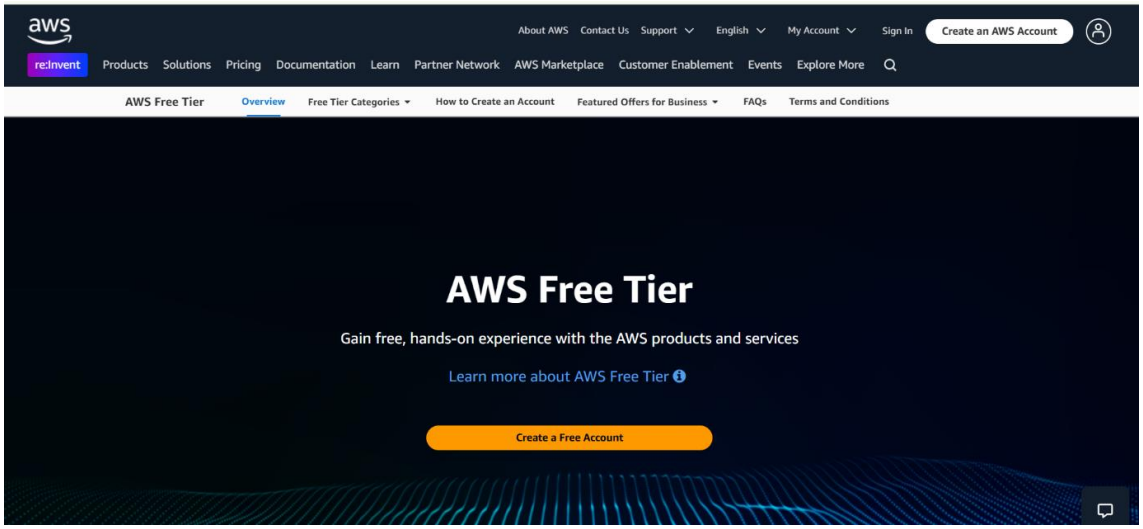
https://www.youtube.com/live/MPau9c7PT74?si=A8OK- zFGbSKkAFWN

6. MySQL WorkBench:

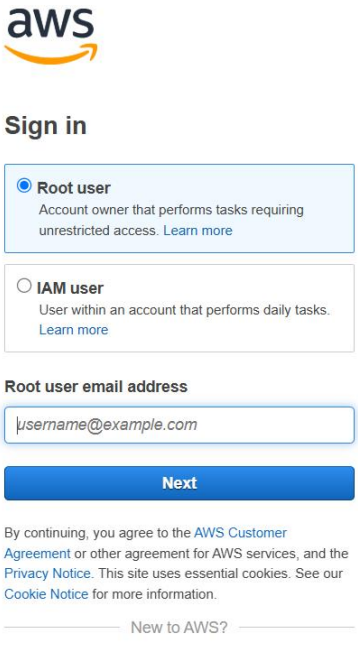
https://youtu.be/wALCw0F8e9M?si=ovMF9qMx5rLxaznB

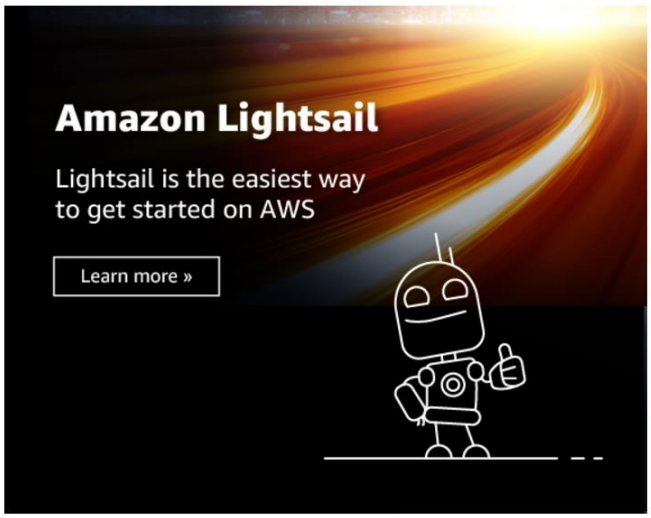
3.Project workflow

Create a AWS account:



Login into AWS management console:





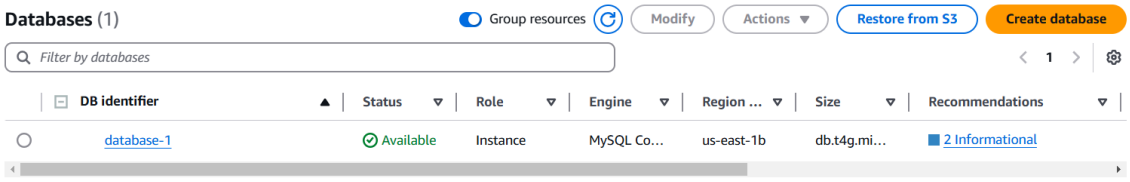
4.RDS database creation and setup

Create an RDS instance:

- Sign in to AWS Console and go to RDS.
- Click Create database.
- Select a Database Engine (e.g., MySQL, PostgreSQL).

- Choose a use case (e.g., Production or Dev/Test).
- Set the DB Instance Identifier, Master Username, and Password.
- Choose an Instance Class (size) and Storage.
- Configure VPC and Public Access settings.
- (Optional) Configure Backups, Monitoring, and Maintenance.
- Review settings and click Create database.

Like I created the RDS instance:



Configure database access:

Modify Security Groups:

- Go to EC2 > Security Groups, add an Inbound Rule to allow traffic on the database port (e.g., 3306 for MySQL).
- Set the Source to your IP or 0.0.0.0/0 for open access (not recommended for production).

Enable Public Access (if needed):

- In the RDS Dashboard, select your instance and make sure Publicly Accessible is set to Yes if you need external access.

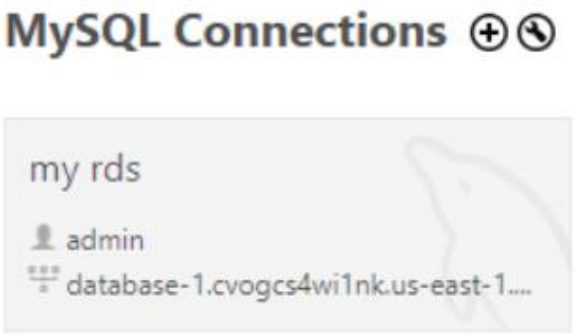
Create a Database User:

- Connect to your RDS instance and create new database users (e.g., using CREATE USER in SQL).

Connect to the Database:

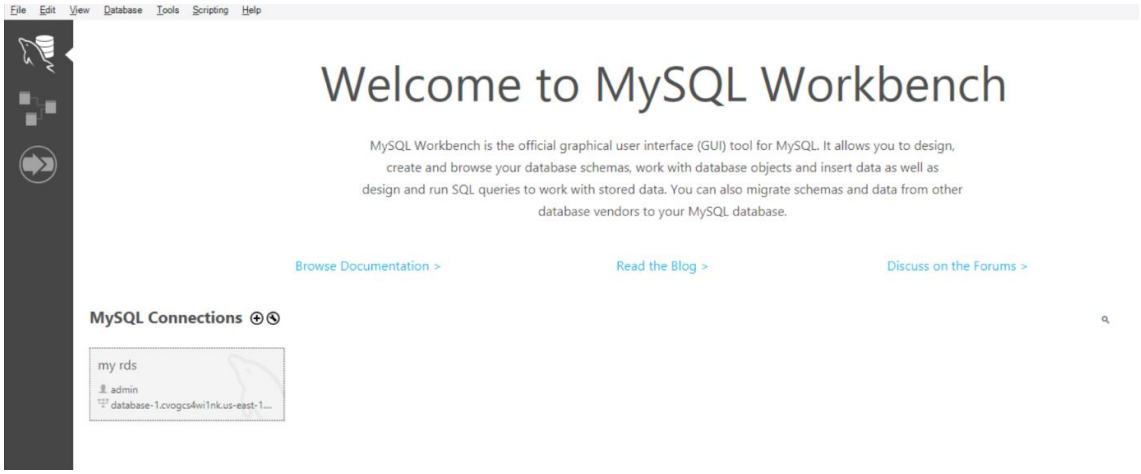
- Use the RDS Endpoint with a database client (e.g., MySQL Workbench) and log in with your database username and password.

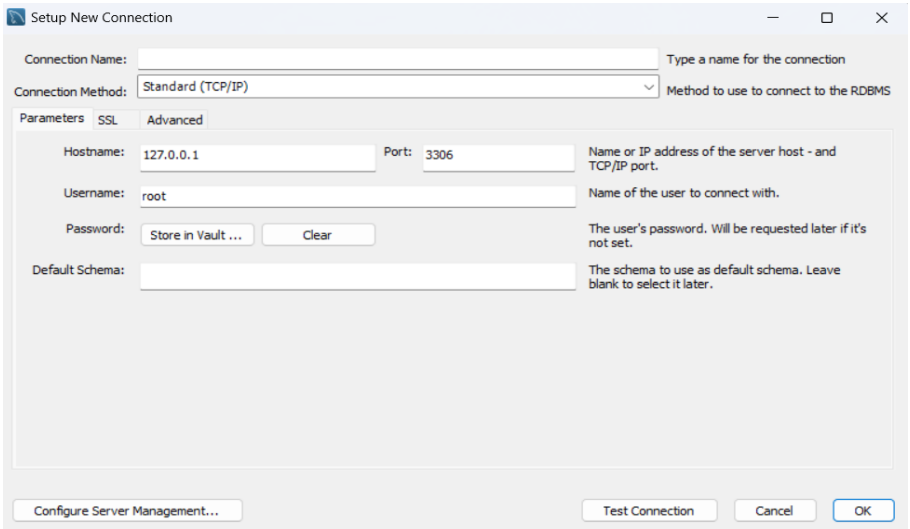
Like I configured database access:



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 like the above one.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

```
1 •  create database fresh;
2 •  use fresh;
```

Tables Created :

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,password,address

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

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;
```

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.

Colomns : 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
);
```

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.

Colomns : id(Primary key),order_id(Foreign key),item_name ,item_price,item_quantity.

Data operations : 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.

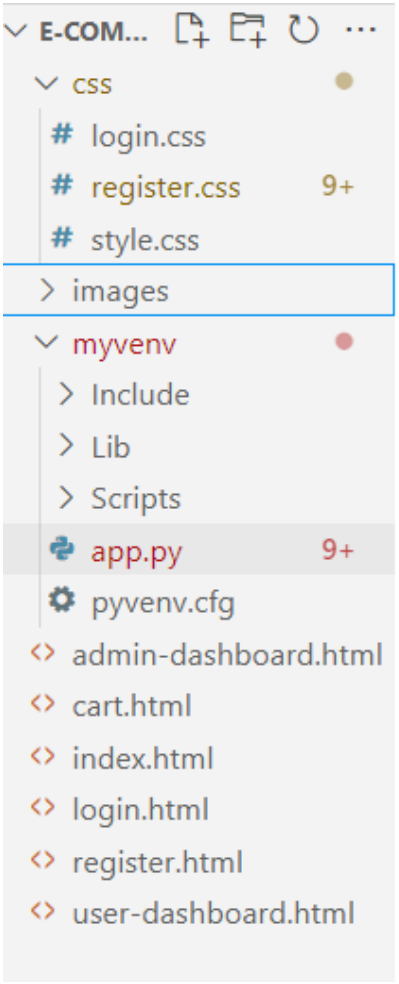
Data Retrieval: We retrieved data from each table to view the stored information, including user details, available items, and orders.

```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
);

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),
('Capsicum',2.20),
('Cucumber',1.00),
('Spinach',1.50),
('Lemon',0.90);

select * from items;
select * from users;
select * from orders;
select * from order_items
```

5.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:

a) 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 = "Madhu"#needed for flash message
```

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

```
# MySQL Database Configuration
ddb_config = {
'host': 'database-1.cvogcs4wi1nk.us-east-1.rds.amazonaws.com', # Your RDS endpoint
'user': 'admin', # Your DB username
'password': 'Madhu#03', # Your DB password
'database': 'fresh' #Your DB name
}
```

c) 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
```

d) Home Route: Renders the home page template when the root URL is accessed.

e) Register Route (GET/POST): Handles user registration, inserts user data into the database.

```
@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')
```

f) 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')
```

g) Shop Route: Renders the shop page, showing available products to logged-in users.

h) Add to Cart (POST): Adds selected items to the user's session-based shopping cart.

```python
@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)
```

i) 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)
```

j) 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))
```

k) 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)
```

l) Admin Dashboard (GET/POST): Allows admin to update order statuses and view detailed orders.

```python
@app.route('/admin_dashboard', methods=['GET', 'POST']) 209

def admin_dashboard():
    if request.method == 'POST':
        order_id = request.form['order_id']
        status = request.form['status']

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('UPDATE orders SET status = %s WHERE id = %s', (status, order_id))
        conn.commit()
        cursor.close()
        conn.close()
        flash('Order status updated successfully!', 'success')

    # Fetch orders with user details and item
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # SQL query to fetch order details with items concatenated as a string

    cursor.execute('''
            SELECT o.id, o.total_price, o.status, o.order_date, u.name AS user_name,
                GROUP_CONCAT(CONCAT(oi.item_name, (x', oi.item_quantity, ')') SEPARATOR ') AS items
            FROM orders o
            JOIN users u ON o.user_id = u.id
            JOIN order_items oi ON o.id oi.order_id
            GROUP BY 0.id
           ''')

    orders = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('admin_dashboard.html', orders=orders)
```

```
if_name_ == '_main_':
    app.run(debug=True)
```

m)Database Queries: Uses SQL queries to interact with MySQL RDS for user and order data.

n) Session Management: Uses Flask session to store user and cart data for navigation.

o) Error Handling: Implements exception handling during order placement to ensure transaction safety.

p) Order Status Update: Admin can change order status, like "shipped" or "delivered."

q) Flash Messages: Provides user feedback through flash messages for login, registration, and updates.

r) Item Data Fetching: Retrieves items from the database to display on the shopping interface.

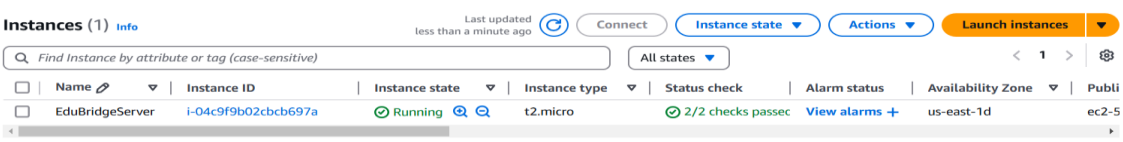
s) Group Concatenation: Combines item names in orders for easy display on the dashboards.

6.EC2 instance setup

Launch an EC2 instance:

- Click Launch Instance.
- Choose an AMI (Amazon Machine Image), like Amazon Linux or Ubuntu.
- Select an Instance Type (e.g., t2.micro for free tier).
- Configure Instance Details (defaults are fine for most).
- Add Storage (default 8GB is fine).
- Add Tags (optional for identification).
- Configure Security Group (allow SSH for Linux or RDP for Windows).
- Review and Launch, then select a Key Pair for access

Lile I launched an EC2 instace:

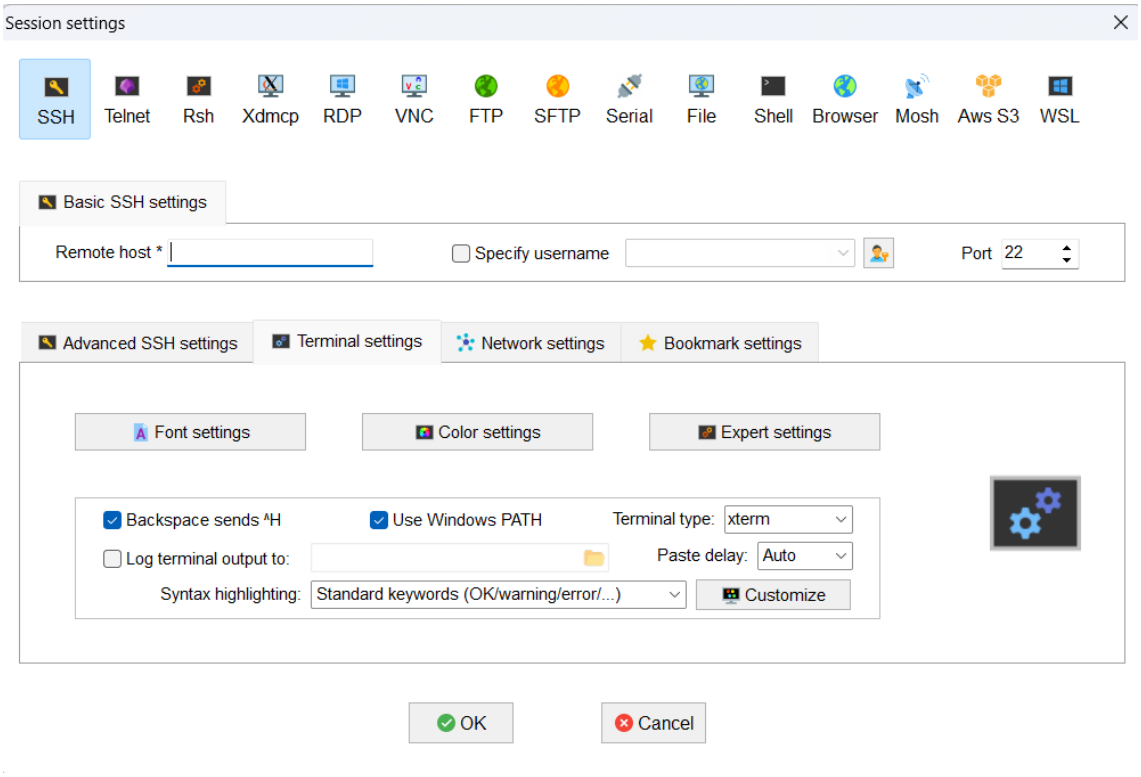


Configure network settings:

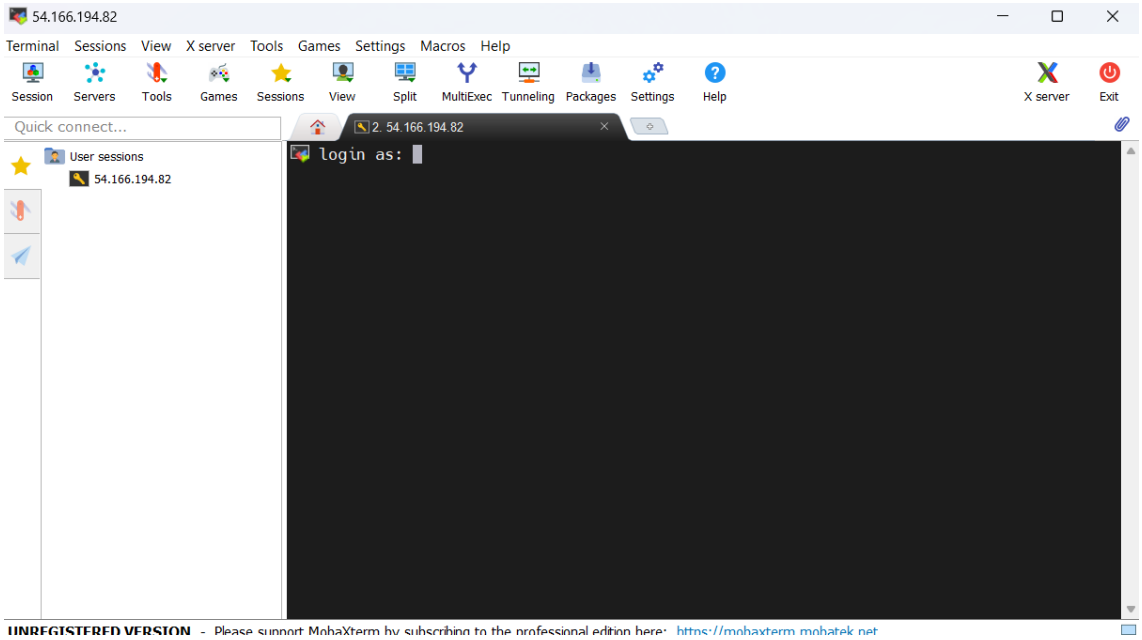
- Set up the security group to allow HTTP, HTTPS, and SSH traffic.
- Create and download the key pair for SSH access.
- Setting up Inbound and Outbound rules
- Add Type : HTTP > Source : Anywhere
- Add Type : HTTPS > Source : Anywhere

7.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).
- 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.

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

Deployment (output):

Welcome to Fresh Market

Home Login Register Cart User

100% Organic and Fresh Produce

Locally Sourced from Trusted Farms

Eco-Friendly Packaging

Excellent Customer Service

Affordable Prices for Premium Quality

Our Products

Apple

$2.00

Quantity (kg): 1 Add to Cart

Banana

$1.50

Quantity (kg): 1 Add to Cart

Orange

$1.80

Quantity (kg): 1 Add to Cart

Tomato

$1.20

Quantity (kg): 1 Add to Cart

Potato

$1.00

Quantity (kg): 1 Add to Cart

Onion

$0.80

Quantity (kg): 1 Add to Cart

Grape

$3.00

Quantity (kg): 1 Add to Cart

Carrot

$1.10

Quantity (kg): 1 Add to Cart

Capsicum

$2.20

Quantity (kg): 1 Add to Cart

Cucumber

$1.00

Quantity (kg): 1 Add to Cart

Spinach

$1.50

Quantity (kg): 1 Add to Cart

Lemon

$0.90

Quantity (kg): 1 Add to Cart

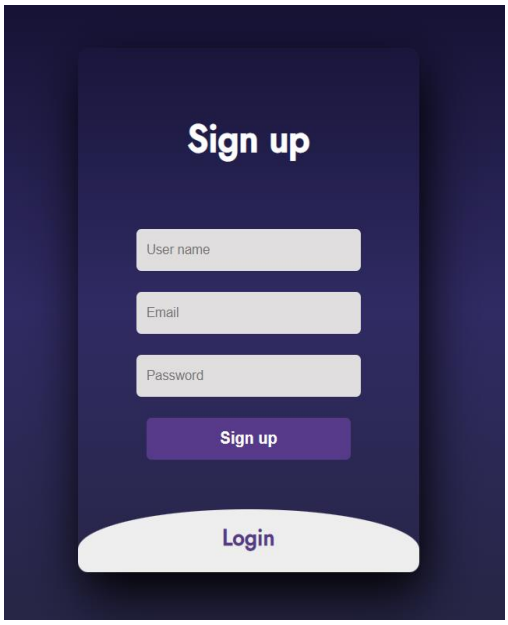
Happy Customers

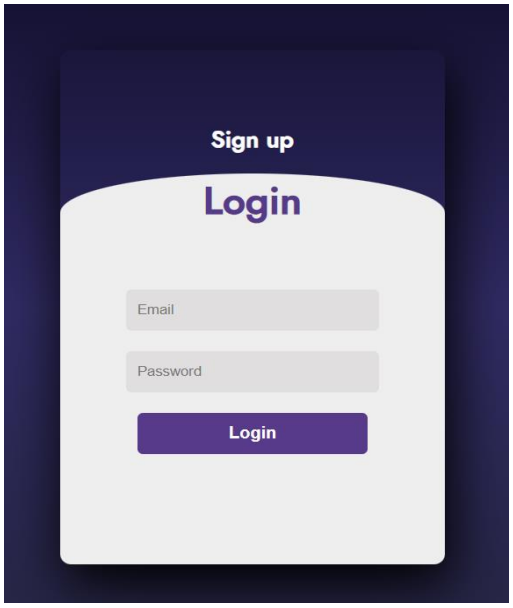
"FreshBasket always delivers high-quality produce I'm a regular customer and highly recommend it!" *-Jane D.*

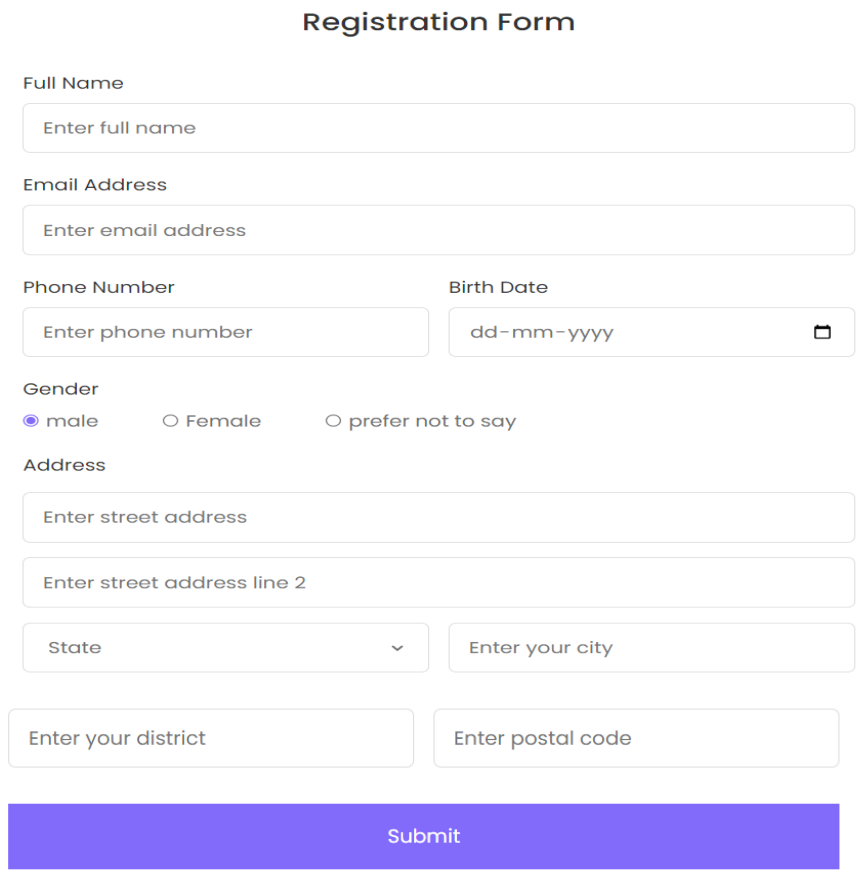
"I love the variety and freshness of the fruits and vegetables. The service is outstanding too!" *-John S .*

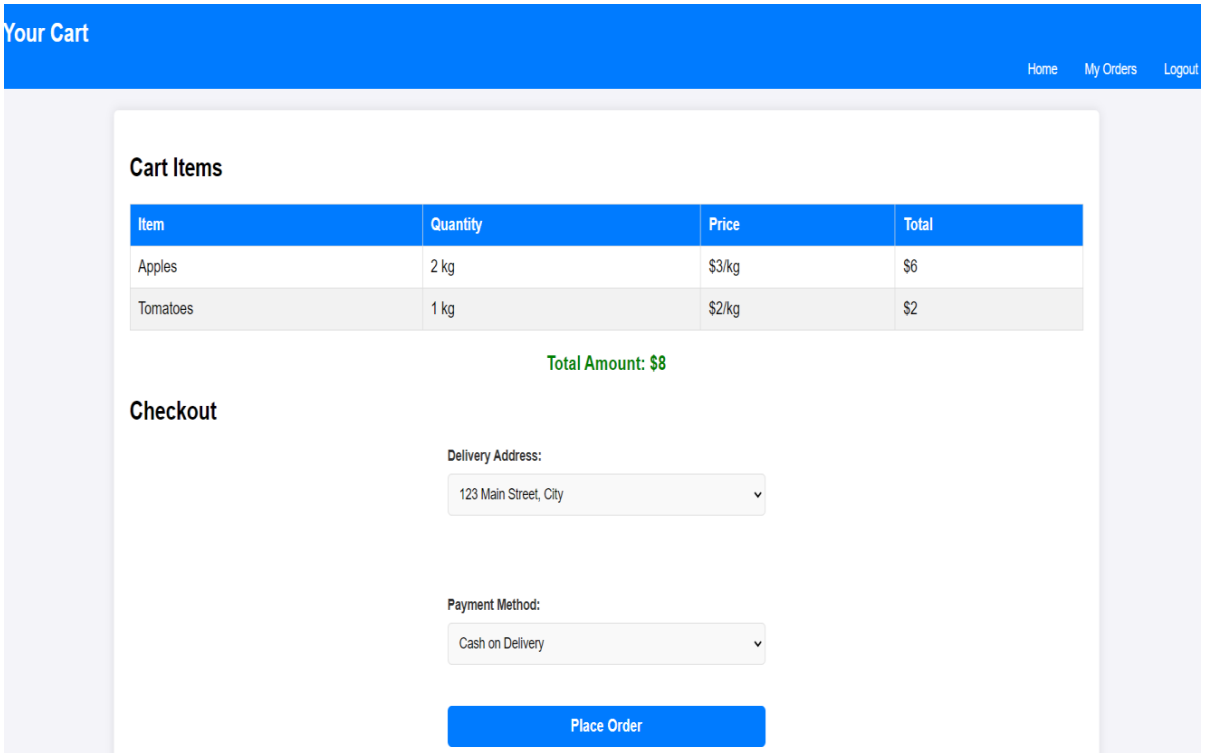
"Great place to shop for organic produce The prices are fair and the quality is top-notch." *Emily R.*

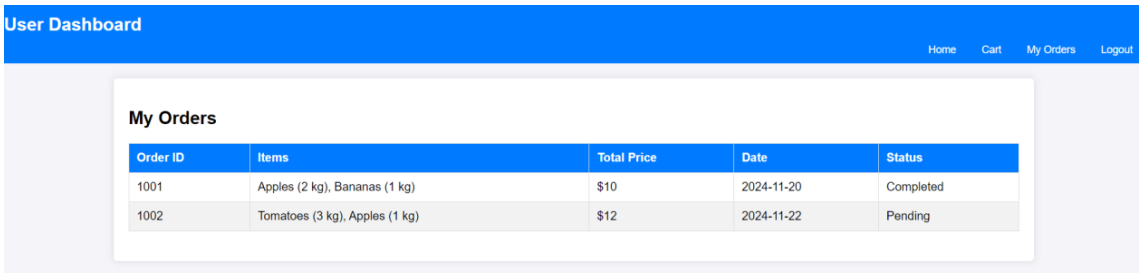
Admin dashboard

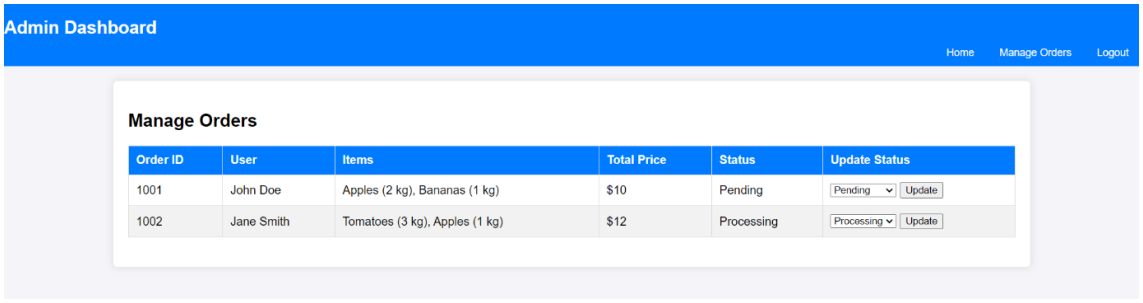

Sign up
User name
Email
Password
Sign up
Login


Sign up
Login
Email
Password
Login


Registration Form
Full Name
Enter full name
Email Address
Enter email address
Phone Number
Enter phone number
Birth Date
dd-mm-yyyy
Gender
male
Female
prefer not to say
Address
Enter street address
Enter street address line 2
State
Enter your city
Enter your district
Enter postal code
Submit

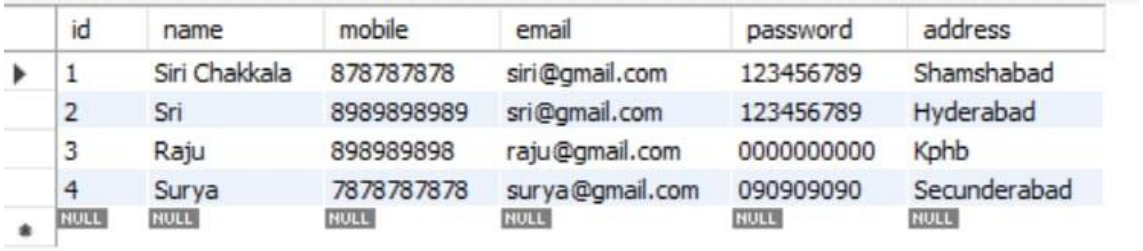

Your Cart
Home
My Orders
Logout
Cart Items
Item
Quantity
Price
Total
Apples
2 kg
$3/kg
$6
Tomatoes
1 kg
$2/kg
$2
Total Amount: $8
Checkout
Delivery Address:
123 Main Street, City
Payment Method:
Cash on Delivery
Place Order





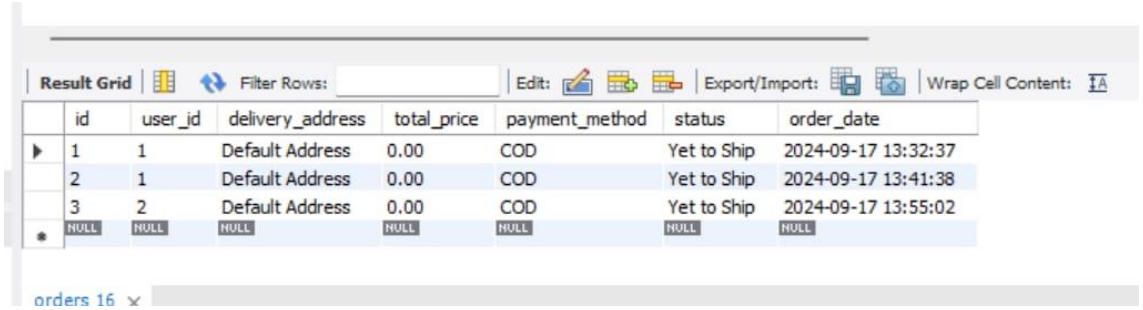
Check the updations in the MYSQL databases:

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:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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

orders 16

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	Capsicum	2.20
120	Cucumber	1.00
121	Spinach	1.50
122	Lemon	0.90

9.Monitoring and optimization:

Performanc monitoring:

- Set up AWS CloudWatch for monitoring EC2 and RDS performance metrics.
- Implement alerts and notifications for critical performance thresholds.

Optimization:

Optimize the server and database configurations based on monitoring results, including adjusting instance types and query optimization.

10.Conclusion:

The FreshBasket project showcases the deployment of a scalable and efficient e-commerce platform using AWS. By integrating Flask with MySQL RDS, it ensures smooth data management and secure backend operations. The EC2 instance, coupled with MobaXterm, enables seamless remote management and deployment of the application. From database setup to frontend development and deployment, each milestone was carefully executed to ensure reliability and performance. AWS CloudWatch is used to monitor performance and ensure smooth functioning. Overall, the project demonstrates the ability of AWS services to support dynamic, scalable web applications in real-world scenarios.