# GROCERY WEB APP USING MERN STACK

| S.NO | NAME | ROLE |
|---|---|---|
| 1 | BALA SUJITHA M | BACKEND |
| 2 | RESHMA BANU R | FRONTEND |
| 3 | SATHYA M | DATABASE |
| 4 | MONICA B | DATABASE |

## TABLE OF CONTENTS:

## ABSTRACT:

Smart Grocery is a web-based application designed to facilitate online grocery shopping, leveraging the MERN (MongoDB, Express.js, React.js, and Node.js) stack. This platform aims to provide a seamless and efficient shopping experience for users, while streamlining inventory management and order fulfillment for grocery stores. The grocery shopping experience has undergone significant transformations with the advent of e-commerce and digital technologies. Online grocery shopping has become increasingly popular, offering convenience, flexibility, and time-saving benefits to consumers. However, existing solutions often face challenges related to inventory management, order fulfillment, and user experience. To address these issues, this project proposes the development of Smart Grocery, a web-based application leveraging the MERN (MongoDB, Express.js, React.js, and Node.js) stack. Smart Grocery aims to provide an intuitive, efficient, and scalable platform for online grocery shopping, streamlining the shopping experience for users and inventory management for store owners.

## PROJECT OVERVIEW:

Smart Grocery is a web-based application designed to facilitate online grocery shopping. The platform allows users to browse and purchase groceries from local stores, while providing store owners with an efficient inventory management system. The goal of this project is to design and develop an intuitive and user-friendly grocery web application that allows users to browse, search, and purchase groceries online. The app will streamline the shopping experience with features like personalized recommendations, easy checkout, and secure payment options.

## PURPOSE:

Develop a user-friendly web application for online grocery shopping. Implement efficient inventory management and order fulfillment. Integrate secure payment gateways. Provide real-time order tracking and status updates. Secure payment processing. Real-time order tracking and status updates.

## FEATURES:

### Customer-Facing Features:

1. User Registration and Login
2. Product Search and Filtering
3. Product Details and Reviews
4. Shopping Cart and Checkout
5. Secure Payment Processing
6. Order Tracking and Status Updates
7. Personalized Recommendations
8. Shopping Lists and Favorites
9. Ratings and Reviews
10. Customer Support

### Admin Features:

1. User Management
2. Store Management
3. Order Management
4. Product Management
5. Reporting and Analytics

6. System Configuration

7. Security and Access Control

8. Integration with third-party services

9. Content Management

10. Customer Support

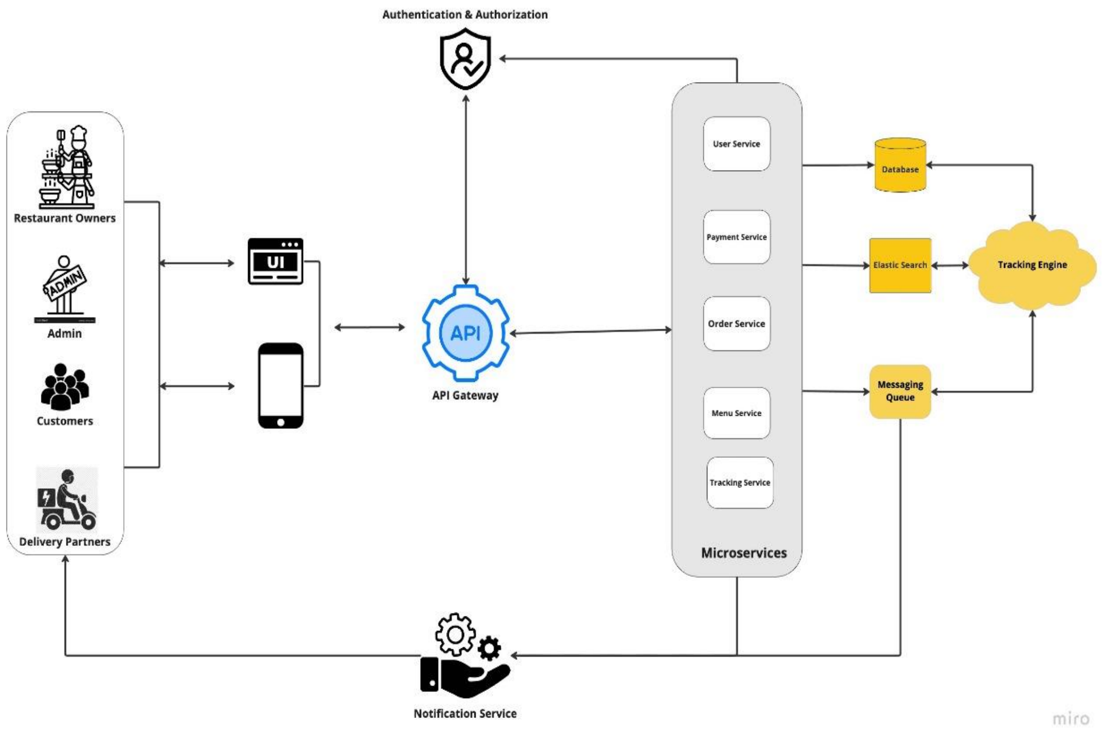
## Security Features:

1. Two-Factor Authentication

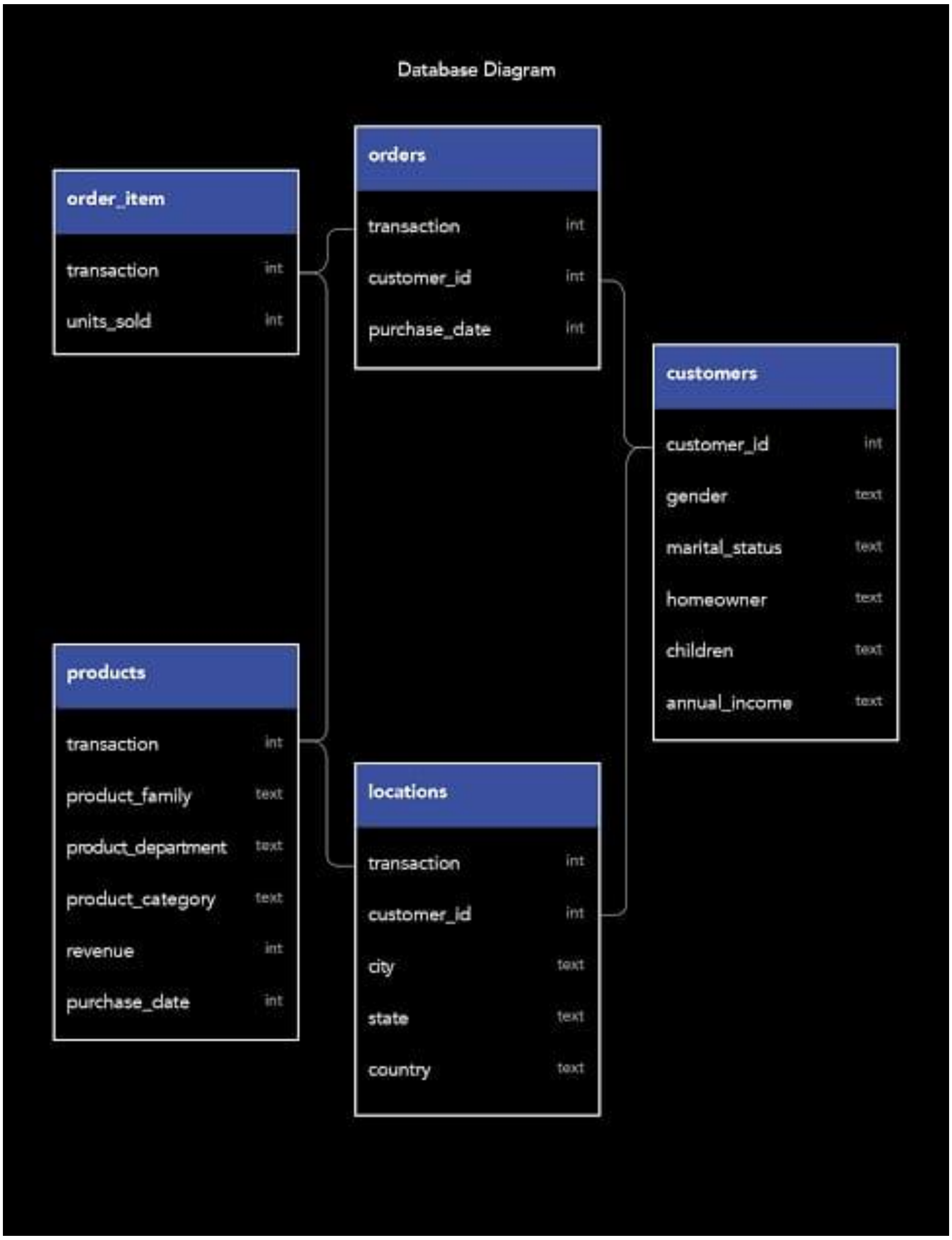
2. Access Control

3. Regular Security Audits

# ARCHITECTURE:



## Database Design:



## SETUP INSTRUCTION:

### Hardware Requirements:

1. Server: Dedicated or cloud-based (e.g., AWS, Google Cloud)

2. Database Server: MySQL, MongoDB, or other NoSQL databases

3. Load Balancer: For distributing traffic across multiple servers

4. Storage: Sufficient storage for product images, videos, and documents

5. Network: Reliable internet connectivity

### Software Requirements:

1. Operating System: Linux, Windows, or macOS

2. Web Server: Apache, Nginx, or IIS

3. Programming Languages:

    - Frontend: JavaScript (React, Angular, Vue), HTML, CSS

    - Backend: Node.js, Python, Ruby, PHP

4. Frameworks:

    - Frontend: React, Angular, Vue

    - Backend: Express.js, Django, Ruby on Rails

5. Database Management System: MySQL, MongoDB, PostgreSQL

6. Payment Gateway Integration: Stripe, PayPal, (link unavailable)

7. Security Software: SSL certificates, firewall, antivirus

## Technical Skills:

1. Frontend development: HTML, CSS, JavaScript, React, Angular, Vue

2. Backend development: Node.js, Python, Ruby, PHP, Express.js, Django

3. Database administration: MySQL, MongoDB, PostgreSQL

4. API integration: RESTful APIs, GraphQL

5. Security: SSL, firewall, antivirus

6. Testing: Unit testing, integration testing, UI testing

7. Deployment: Cloud deployment (AWS, Google Cloud), containerization (Docker)

## INSTALLATION:

**Step 1:** Install Node.js and npm

1. Download and install Node.js from the official website: (link unavailable)

2. Verify Node.js installation: node -v

3. Verify npm installation: npm -v

**Step 2**: Install MongoDB

1. Download and install MongoDB Community Server from the official website: (link unavailable)

2. Follow the installation instructions for your operating system.

3. Verify MongoDB installation: mongod --version

**Step 3:** Install Express.js

1. Create a new project directory: mkdir grocery-web-app

2. Navigate to the project directory: cd grocery-web-app

3. Initialize a new npm project: npm init

4. Install Express.js: npm install express

**Step 4:** Install React.js

1. Create a new React app: npx create-react-app client

2. Navigate to the client directory: cd client

3. Install required dependencies: npm install

**Step 5:** Install Required Dependencies

Backend (Node.js/Express.js)

1. Install mongoose (MongoDB ORM): npm install mongoose

2. Install bcrypt (password hashing): npm install bcrypt

3. Install json web token (JWT authentication): npm install json web token

4. Install cors (CORS middleware): npm install cors

Frontend (React.js)

1. Install axios (HTTP client): npm install axios

2. Install react-router-dom (client-side routing): npm install react-router-dom

**Step 6:** Set up MongoDB Database

1. Create a new MongoDB database: use grocery-db

2. Create collections for users, products, orders, etc.

**Step 7:** Configure Backend (Node.js/Express.js)

1. Create a new file server.js

2. Import required dependencies: express, mongoose, bcrypt, jsonwebtoken, cors

3. Set up Express.js app: const app = express ()

4. Configure MongoDB connection: mongoose.connect('mongodb://localhost/grocery-db')

5. Define API routes: /api/users, /api/products, /api/orders

**Step 8:** Configure Frontend (React.js)

1. Create a new file index.js

2. Import required dependencies: react, react-dom, axios, react-router-dom

3. Set up React app: ReactDOM.render(<App />, document.getElementById('root'))

4. Define components: App.js, ProductList.js, OrderForm.js

**Step 9:** Implement Authentication and Authorization

Backend (Node.js/Express.js)

1. Implement user registration: /api/users/register

2. Implement user login: /api/users/login

3. Implement JWT authentication: json web token

Frontend (React.js)

1. Implement login and registration forms

2. Implement JWT token storage: local Storage

**Step 10:** Implement Payment Gateway

1. Choose a payment gateway (e.g., Stripe, PayPal)

2. Implement payment gateway integration

**Step 11:** Test and Deploy

1. Test application functionality

2. Deploy application to production environment (e.g., Heroku, AWS)

## FOLDER STRUCTURE:

The root directory, grocery-web-app, contains two main folders: client and server. The client folder holds the frontend code, with subfolders including public for static assets, src for source code, components for reusable UI components, containers for container components, actions and reducers for Redux management, store for Redux store configuration, utils for utility functions, images for image assets, and styles for CSS styles. The src folder also contains the index.js entry point. The server folder holds the backend code, with subfolders including config for configuration files, controllers for API controllers, models for database models, routes for API routes, services for business logic services, and utils for utility functions. The server folder also contains the app.js entry point. The root directory also contains a package.json file for project dependencies and scripts

### Client (Frontend) Folder Structure:

client/

public/

index.html

favicon.ico

src/

components/

App.js

Header.js

Footer.js

ProductList.js

OrderForm.js

...

containers/

ProductDetails.js

OrderSummary.js

...

actions/

productActions.js

orderActions.js

...

reducers/

productReducers.js

orderReducers.js

...

store/

store.js

...

utils/

api.js

constants.js

...

images/

product-images/

...

styles/

global.css

...

index.js

package.json

## Server (Backend) Folder Structure:

server/

config/

database.js

auth.js

...

controllers/

productController.js

orderController.js

...

models/

productModel.js

orderModel.js

...

routes/

productRoutes.js

orderRoutes.js

...

services/

productService.js

orderService.js

...

utils/

auth.js

errors.js

...

app.js

package.json

## RUNNING THE APPLICATION:

### Frontend (Client) Server:

1. Navigate to the client directory: cd client
2. Install dependencies (if not already installed): npm install
3. Start the frontend server: npm start. Default port: 3000

   Access the frontend at: http://localhost:3000

### Backend (Server) Server:

1. Navigate to the server directory: cd server
2. Install dependencies (if not already installed): npm install
3. Start the backend server: npm start

Default port: 8080 (or configured port in server/app.js)

Access the backend API at: http://localhost:8080/api

- ➢ To start both frontend and backend servers concurrently

1. Install concurrently package: npm install concurrently
2. Add scripts to package.json in the root directory:

```
"scripts": {
  "start:client": "npm start --prefix client",
  "start:server": "npm start --prefix server",
  "start": "concurrently \"npm run start:client\" \"npm run
start:server\""
}
```

3. Start both servers: npm start

## API DOCUMENTATION:

An **API (Application Programming Interface)** is a set of rules and tools that allows different software systems to communicate and share data or functionalities. It acts as a bridge between applications, enabling them to interact without needing to know the internal details of each other.

Authentication Endpoints

1. Register User:

- Method: POST
- Endpoint: /api/auth/register
- Parameters:
  - name: String (required)
  - email: String (required, unique)
  - password: String (required)
- Request Body:

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- Response:

```
{
  "message": "User created successfully",
  "token": "jwt-token"
}
```

2. Login User:

- Method: POST
- Endpoint: /api/auth/login
- Parameters:
    - email: String (required)
    - password: String (required)
- Request Body:

```
{
  "email": "johndoe@example.com",
  "password": "password123"}
```

- Response:

```
{
  "message": "Login successful",
  "token": "jwt-token"
}
```

Product Endpoints

1. Get All Products

- Method: GET
- Endpoint: /api/products

- Parameters: None
- Response:

```
[
  {
    "_id": "product-id",
    "name": "Product Name",
    "price": 10.99,
    "description": "Product description"
  },
  ...]
```

2. Get Product by ID

- Method: GET
- Endpoint: /api/products/:id
- Parameters:
    - id: String (required)
- Response:

```
{
"_id": "product-id",
  "name": "Product Name",
  "price": 10.99,
  "description": "Product description"}
```

3. Create Product

- Method: POST
- Endpoint: /api/products
- Parameters:

   - name: String (required)
   - price: Number (required)
   - description: String (required)

- Request Body:

```
{
  "name": "New Product",
  "price": 9.99,
  "description": "New product description"
}
```

- Response:

```
{
  "message": "Product created successfully",
  "_id": "new-product-id"
}
```

4. Update Product

- Method: PUT
- Endpoint: /api/products/:id
- Parameters:
   - id: String (required)
   - name: String (optional)
   - price: Number (optional)
   - description: String (optional)
- Request Body:

```
{
  "name": "Updated Product Name"
```

}

- Response:

```
{
  "message": "Product updated successfully"
}
```

5. Delete Product

- Method: DELETE
- Endpoint: /api/products/:id
- Parameters:
   - id: String (required)
- Response:

```
{
  "message": "Product deleted successfully"
}
```

Order Endpoints

1. Create Order

- Method: POST
- Endpoint: /api/orders
- Parameters:
   - products: Array of product IDs (required)
   - total: Number (required)
- Request Body:

```
{
  "products": ["product-id-1", "product-id-2"],
  "total": 20.98
}
```

- Response:

```
{
  "message": "Order created successfully",
  "_id": "order-id"
}
```

2. Get Order by ID

- Method: GET
- Endpoint: /api/orders/:id
- Parameters:
    - id: String (required)
- Response:

```
{
  "_id": "order-id",
  "products": ["product-id-1", "product-id-2"],
 "total": 20.98,
  "status": "pending"
}
```

3. Update Order Status

- Method: PUT
- Endpoint: /api/orders/:id
- Parameters:
    - id: String (required)
    - status: String (required, e.g., "shipped", "delivered")
- Request Body:

```
{
  "status": "shipped"
}
```

- Response:

```
{
  "message": "Order status updated successfully"
}
```

## AUTHENTICATION:

Authentication is the process of verifying the identity of users attempting to access a grocery web app, ensuring only authorized individuals can:

1. Place orders
2. Access account information
3. View order history
4. Make payments

It includes:

1. User Registration: When a user registers, their password is hashed using bcrypt and stored in the MongoDB database.

2. User Login: When a user logs in, the provided password is hashed and compared to the stored hash.

3. JWT (JSON Web Tokens) Generation: Upon successful login, a JWT is generated using the user's ID, email, and other relevant information.

4. Token Signing: The JWT is signed with a secret key using the HS256 algorithm.

## Authorization:

1. Token Verification: On subsequent requests, the client sends the JWT in the Authorization header.

2. Token Verification: The server verifies the token using the secret key and checks its expiration.

3. User Retrieval: If valid, the server retrieves the user's information from the database using the ID in the token.

4. Permission Checking: The server checks the user's permissions and roles to determine access to protected routes.

## Token-Based Authentication:

1. Token Structure: The JWT contains the following information:

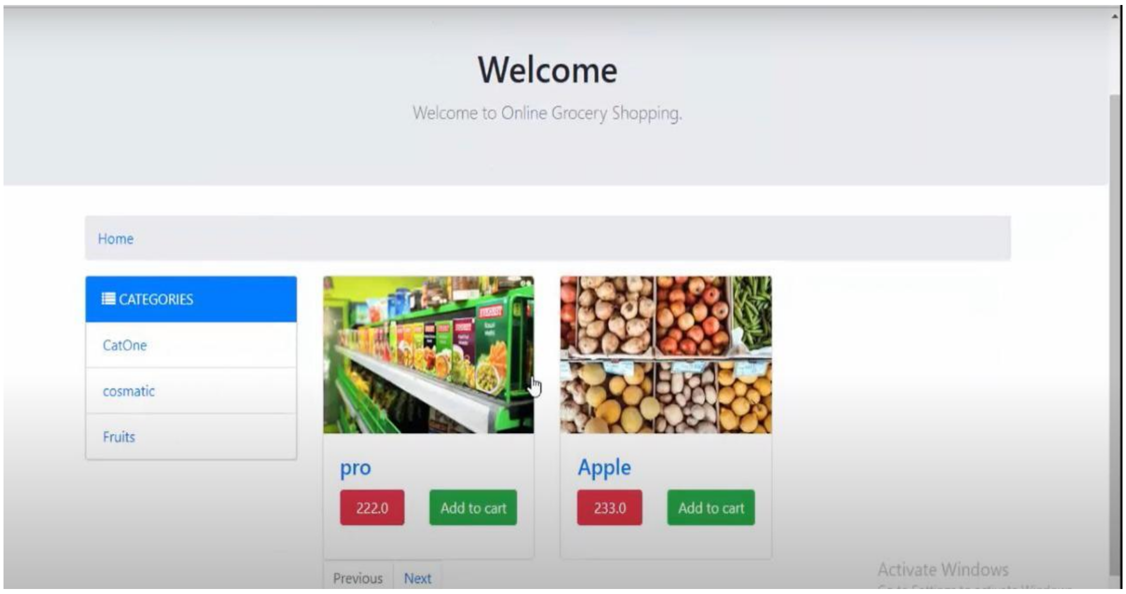
   - User ID (sub)
   - Email (email)
   - Role (role)
   - Expiration Time (exp)

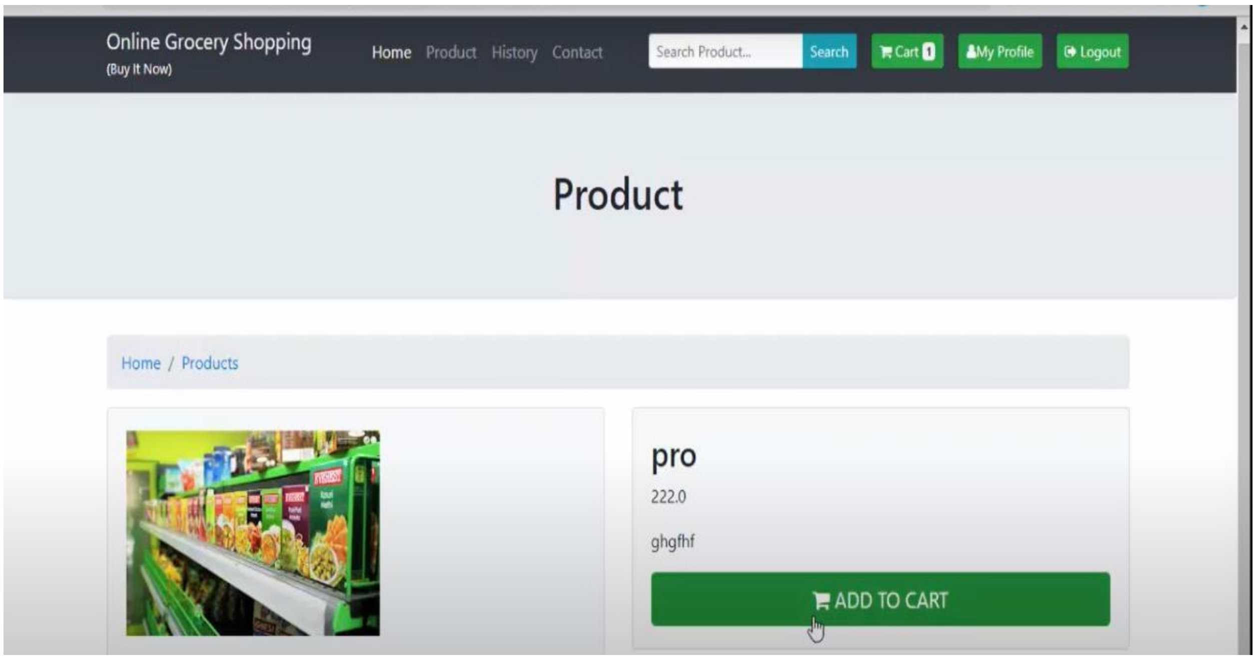
2. Token Storage: The client stores the JWT in local storage or cookies.

3. Token Renewal: The token is renewed upon expiration or when the user logs out.

# USER INTERFACE DESIGN:

User Interface (UI) Design is the process of designing the visual elements and interactive features of a digital product, such as a website, mobile app, or software. The goal of UI design is to create an interface that is intuitive, visually appealing, and user-friendly.


Welcome
Welcome to Online Grocery Shopping.
Home
CATEGORIES
CatOne
cosmatic
Fruits
pro
222.0
Add to cart
Apple
233.0
Add to cart
Previous
Next
Activate Windows


Online Grocery Shopping
(Buy It Now)
Home
Product
History
Contact
Search Product...
Search
Cart 1
My Profile
Logout
Product
Home / Products
pro
222.0
ghgfhf
ADD TO CART

# TESTING:

**Test Cases**

- Sample test case table:

| **Test cases ID** | **Description** | **Input** | **Expected output** | **Status** |
|---|---|---|---|---|
| TC 001 | Login Functionality | Valid login | Redirect to home | Pass |

**Testing Tools Used:**

- List tools like Selenium, Postman, etc.

# CHALLENGES:

## Technical Challenges:

1. Integration with multiple payment gateways
2. Secure payment processing
3. Real-time inventory management
4. Product information management (PIM)

## Business Challenges:

1. Competition from established players
2. Managing logistics and supply chain
3. Maintaining profit margins
4. Ensuring customer satisfaction and retention

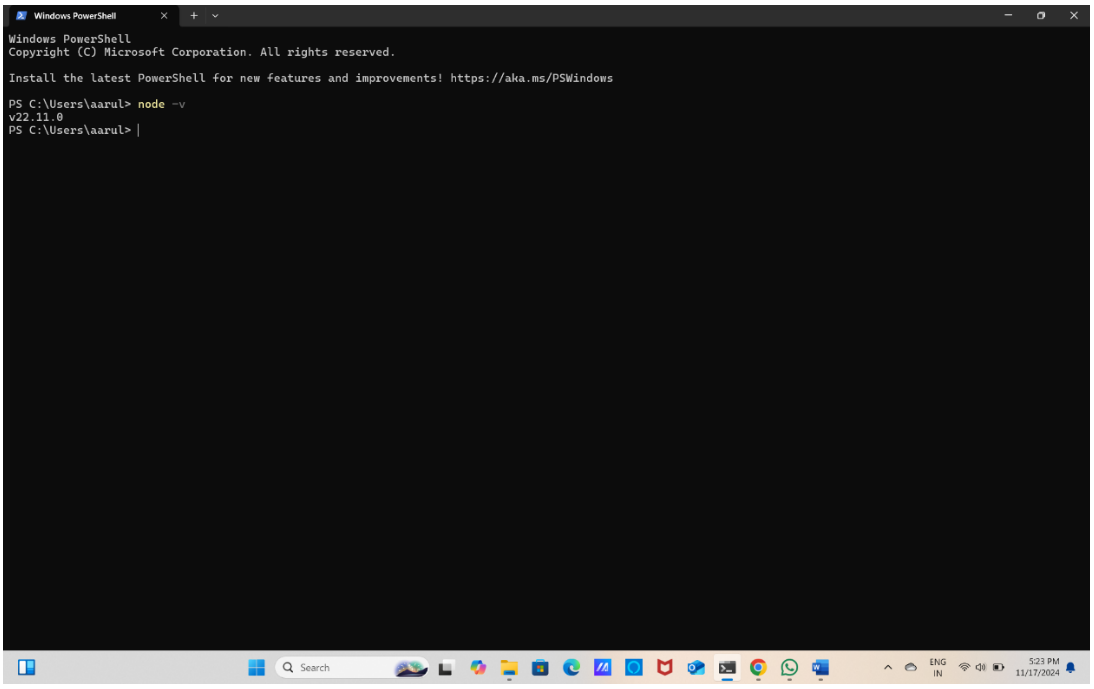
## User Experience Challenges:

1. Simple and intuitive user interface
2. Easy product discovery and navigation
3. Personalized recommendations
4. Streamlined checkout process

## Security Challenges:

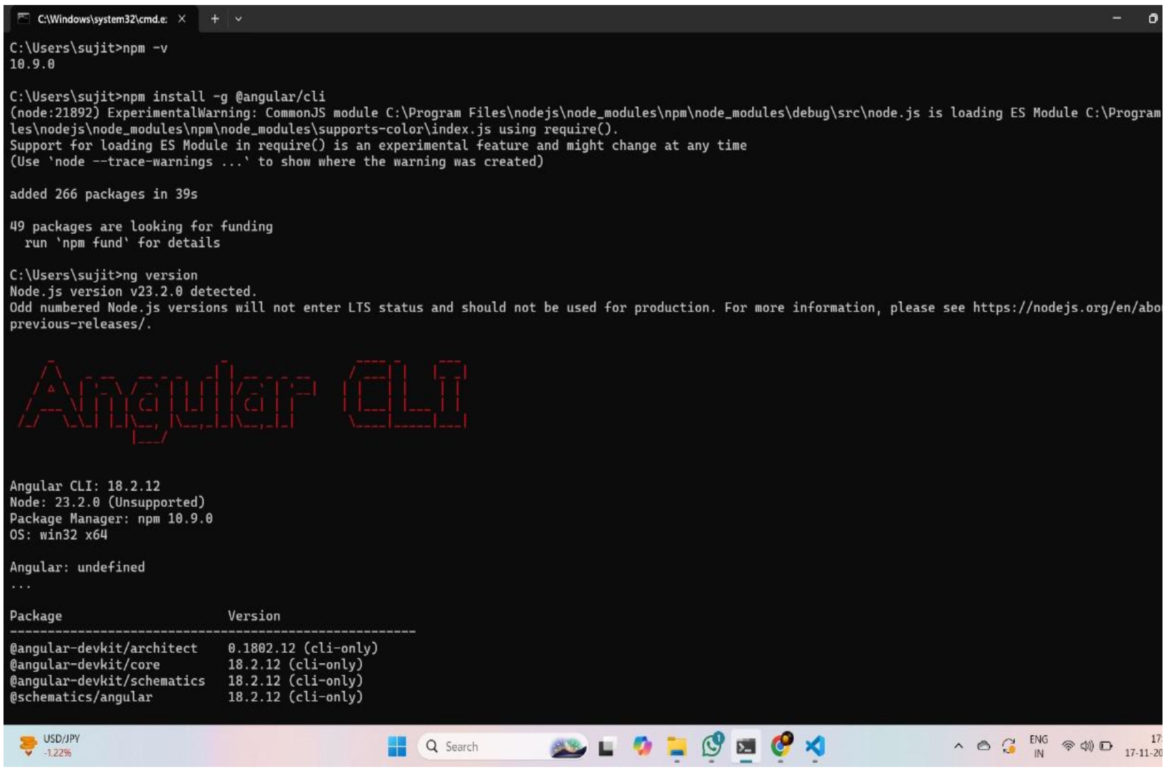
1. Protecting user data and privacy
2. Preventing cyberattacks and hacking
3. Ensuring secure payment processing
4. Complying with data protection regulations

## SCREENSHOT OR DEMO:



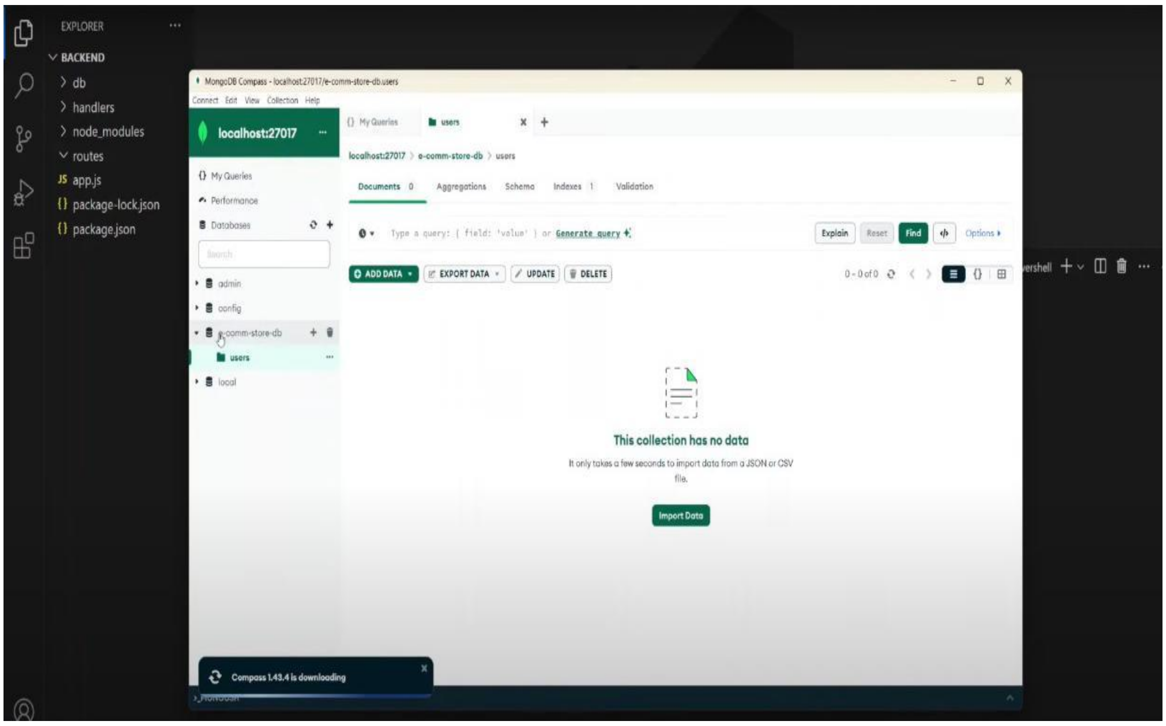
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\aarul> node -v
v22.11.0
PS C:\Users\aarul> |
```

✓ **Installation and Verification the version of .node js package**



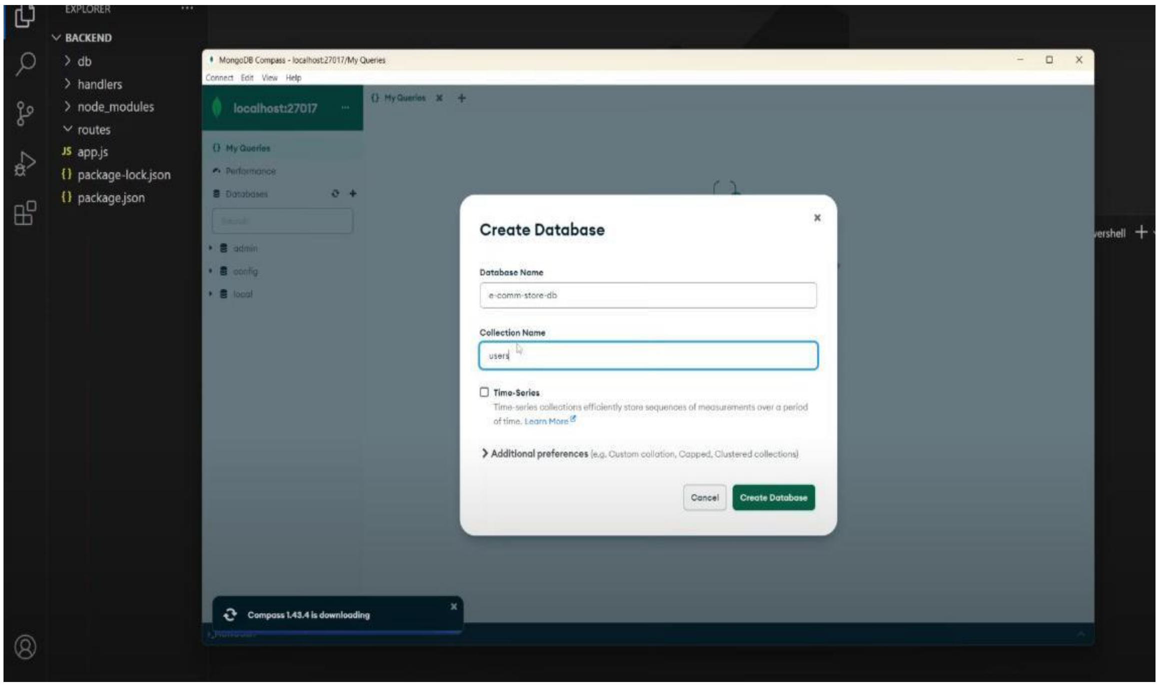
```
C:\Users\sujit>npm -v
10.9.0

C:\Users\sujit>npm install -g @angular/cli
(node:21892) ExperimentalWarning: CommonJS module C:\Program Files\nodejs\node_modules\npm\node_modules\debug\src\node.js is loading ES Module C:\Program
les\nodejs\node_modules\npm\node_modules\supports-color\index.js using require().
Support for loading ES Module in require() is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)

added 266 packages in 39s

49 packages are looking for funding
  run `npm fund` for details

C:\Users\sujit>ng version
Node.js version v23.2.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production. For more information, please see https://nodejs.org/en/abo
previous-releases/.

Angular CLI: 18.2.12
Node: 23.2.0 (Unsupported)
Package Manager: npm 10.9.0
OS: win32 x64

Angular: undefined
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1802.12 (cli-only)
@angular-devkit/core         18.2.12 (cli-only)
@angular-devkit/schematics   18.2.12 (cli-only)
@schematics/angular          18.2.12 (cli-only)
```

✓ **Install and verify the npm and angular CLI for further testing.**



This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.



Create Database

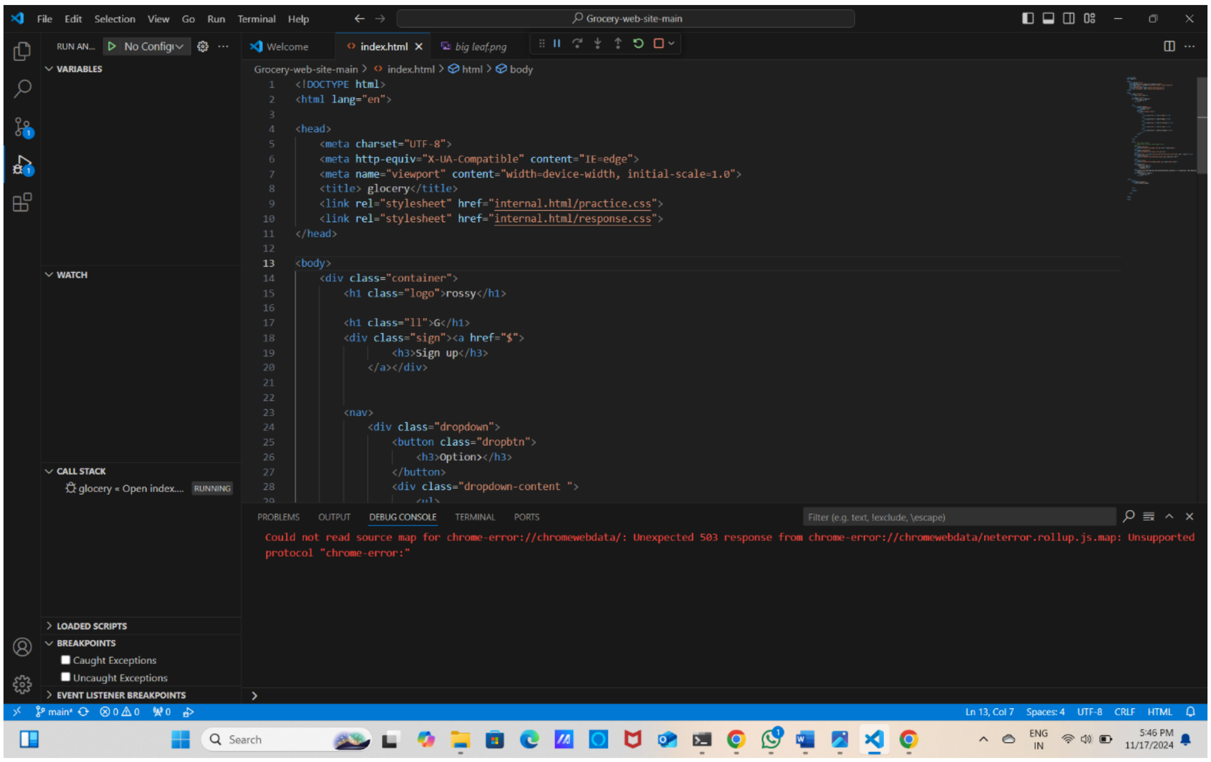
Database Name: e-comm-store-db

Collection Name: users

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

✓ **Setup a mongodb (database) path for managing operations and collection.**



```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> glocery</title>
    <link rel="stylesheet" href="internal.html/practice.css">
    <link rel="stylesheet" href="internal.html/response.css">
</head>

<body>
    <div class="container">
        <h1 class="logo">rossy</h1>

        <h1 class="ll">G</h1>
        <div class="sign"><a href="$">
                <h3>Sign up</h3>
            </a></div>


        <nav>
            <div class="dropdown">
                <button class="dropbtn">
                    <h3>Option></h3>
                </button>
                <div class="dropdown-content ">
```

```
Could not read source map for chrome-error://chromewebdata/: Unexpected 503 response from chrome-error://chromewebdata/neterror.rollup.js.map: Unsupported protocol "chrome-error:"
```

# FUTURE ENHANCEMENT:

**Future enhancements for a grocery web app** to improve user experience, efficiency, and business performance:

1. **AI-Powered Shopping List**
    - Enable users to create smart shopping lists based on past purchases, seasonal trends, or recipes they are viewing.
    - Automatically categorize items into sections like produce, dairy, and beverages.
2. **Augmented Reality (AR)**
    - Provide AR features to visualize product packaging or check item placement in their kitchen/pantry.
3. **Live Order Tracking**
    - Show real-time updates on order preparation and delivery, similar to food delivery apps.
4. **Voice Search Integration**
    - Allow users to search and add items to their cart using voice commands, integrating with virtual assistants like Alexa or Google Assistant.
5. **Subscription Services**
    - Introduce subscription options for frequently purchased items (e.g., weekly milk delivery or monthly rice purchase)

## CONCLUSION:

In conclusion, a grocery web app serves as a vital tool in today’s digital shopping landscape, offering convenience and efficiency to consumers seeking their everyday essentials. By providing an intuitive user experience, real-time inventory management, and secure payment options, such an app can not only enhance customer satisfaction but also foster brand loyalty. As the demand for online grocery shopping continues to grow, addressing challenges such as logistics, personalization, and data security will be essential for success. Ultimately, a well-designed grocery web app can bridge the gap between traditional shopping and modern convenience, transforming the way customers engage with their local grocery stores and ensuring a seamless shopping experience that meets their evolving needs.