**AI-Powered Virtual Shopping Assistant**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** |  |
|  | **INTRODUCTION** |  |
|  | * 1. General Introduction |  |
|  | * 1. Objectives |  |
|  | **SYSTEM** **PROPOSAL** |  |
|  | * 1. Existing System |  |
|  | * + 1. Disadvantages |  |
|  | * 1. Proposed System |  |
|  | 2.2.1 Advantages |  |
|  | **SYSTEM DESIGN** |  |
|  | * 1. Architecture Diagram |  |
|  | * 1. Flow Diagram |  |
|  | * 1. UML Diagrams |  |
|  | * 1. Activity Diagram |  |
|  | * 1. E-R Diagram |  |
|  | * 1. Sequence Diagram |  |
|  | **IMPLEMENTATION** |  |
|  | * 1. Modules |  |
|  | * 1. Modules Description |  |
|  | **SYSTEM** **REQUIREMENTS** |  |
|  | * 1. Hardware Requirements |  |
|  | * 1. Software Requirements |  |
|  | * 1. Software Description |  |
|  | * 1. Testing of Products |  |
| **6.** | **CONCLUSION** |  |
| **7.** | **FUTURE ENHANCEMENT** |  |
| **8.** | **SAMPLE** **CODING** **AND** **SAMPLE** **SCREENSHOT** |  |
| **9.** | **REFERENCES** |  |

**Abstract:**

Nowadays online ordering systems play a vital role in every one's life. An online ordering system can be defined as software that allows customers to view and order multiple items over the internet. The main objective of an Online store is to ensure customer and vendor satisfaction. Instead of a customer being confined to Restaurants, Super Markets, Pharmacy around their home or one that only attracts nearby residents, customers can now discover new Online stores. Connecting new customers to online retailers is one of the industry's greatest advantages, and it also encourages customers to be a bit more curious and find a new location. But the online ordering app has only one or two categories with food. We have implemented an app for online ordering which allows customers to order multiple categories like food, grocery, medicinal items, fruits and vegetables on the same platform.

**CHAPTER 1**

**INTRODUCTION**

* 1. **GENERAL INTRODUCTION:**
* An E-commerce website requires appropriate strategy of successful design and implementation. Everything is required to plan from scratch to end of website.
* The e-commerce sector is seen the exponential growth thus a new option will easily part of this regatta of commercial website.
* The e-commerce website will feature the online shopping facility of various fashion products under a single web space
* It will allow multiple shopping vendors to sale their products online. The product management in the system will be done in the form of categories.

**1.2. OBJECTIVES:**

* The main objective of the system is to reach maximum customers at the right time to increase sales and profitability of the business.
* To improve the functions of e-commerce include buying and selling goods, transmitting funds or data over the internet.in suggestion as product as health condition.
* That enables users to shop virtually through internet and to purchase the products of their choice from the retailer.
* Web based shopping is the course of clients purchases merchandise from a seller progressively over the Internet without the need of a mediator supplier. It's a kind of electronic exchange.

**CHAPTER 2**

**SYSTEM PROPOSAL**

**2.1 EXISTING SYSTEM:**

* The longer future also includes a scenario that will see the growth of specialty organizations, as well as mergers and acquisitions to enable companies to grow inorganically.
* To attract and retain a larger audience, businesses should spend in the organization of services and the use of web advertising.
* Potential open doors are emerging inside the local business sectors through flexible entry, including mobile application framework, content improvement in regional dialects, and walkability

**2.1.1. DISADVANTAGES:**

* Less secure
* .It takes server's processing and time, which poses some issues for low-end websites which can run on even very little bandwidth.

**2.2. PROPOSED SYSTEM:**

* The proposed model is introduced to overcome all the disadvantages that arises in the existing system.
* The proposed, even though online merchants have tried their best to beef up the security, threats and attacks still prevail. For this reason, consumer should act fast to protect their privacy when shopping online explains many ways that consumers could do to enhance the privacy and security aspect apart from what online merchants have done for the same reason.
* Taking all these contents as a whole, I would say that in any situation, people can still shop online safely provided they understand the reality and take some precautions above all.

**2.2.1. ADVANTAGE:**

* Generates working software quickly and early during the software life cycle.
* It is easier to test and debug during a smaller iteration.
* In this model customer can respond to each built.
* Lowers initial delivery cost.
* Easier to manage risk because risky pieces are identified and handled during it’d iteration.
* This model is more flexible and less costly to change scope and requirement

**2.2.2. PROBLEM STATEMENT**

* Traditionally, customers are used to buying the products at the real, in other words, factual shops or supermarkets.
* It needs the customers to show up in the shops in person, and walk around different shopping shelves, and it also needs the owners of shops to stock, exhibit, and transfer the products required by customers.
* It takes labor, time and space to process these operations.

**CHAPTER 3**

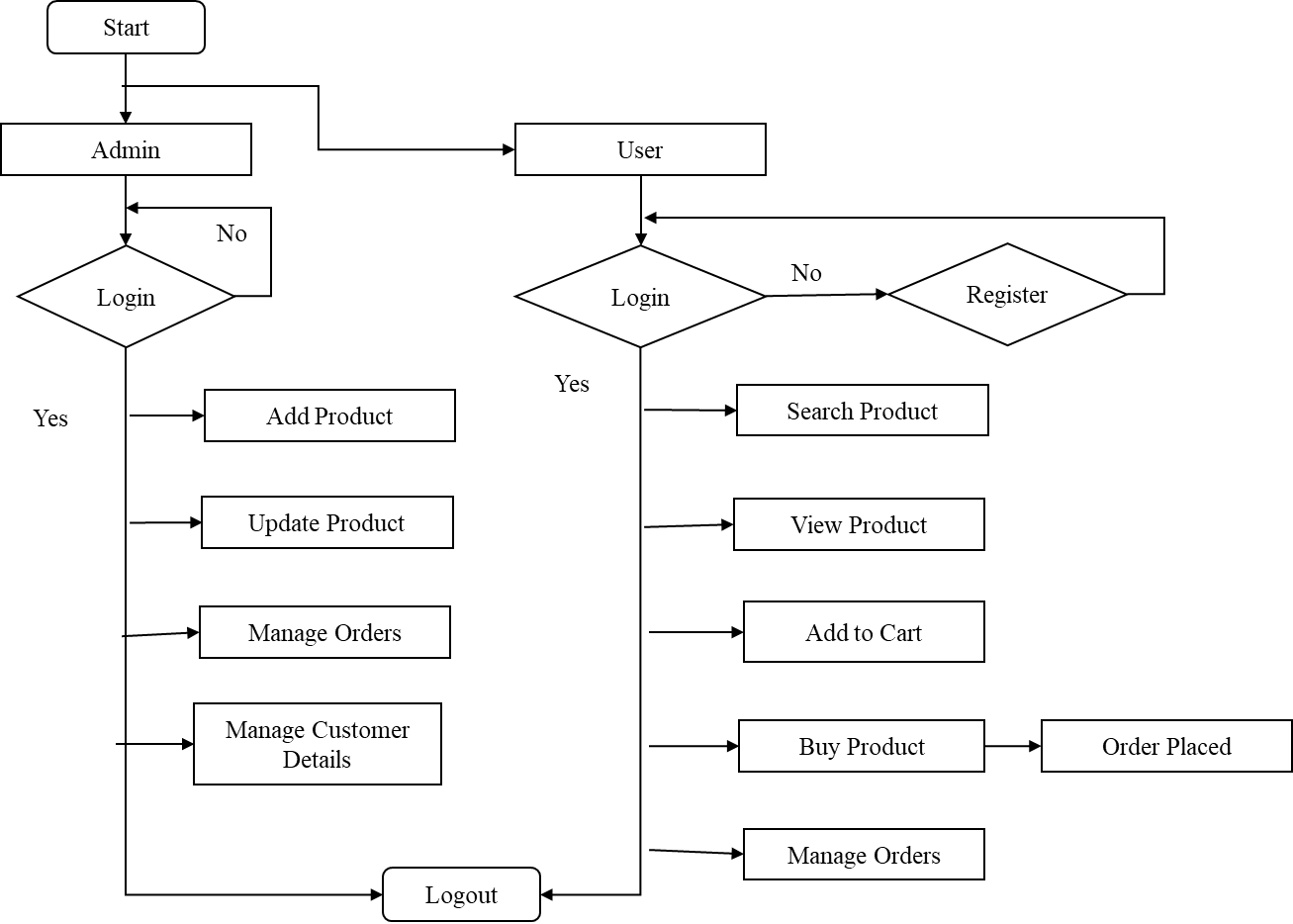
**SYSTEM DIAGRAMS**

**3.1 SYSTEM ARCHITECTURE:**

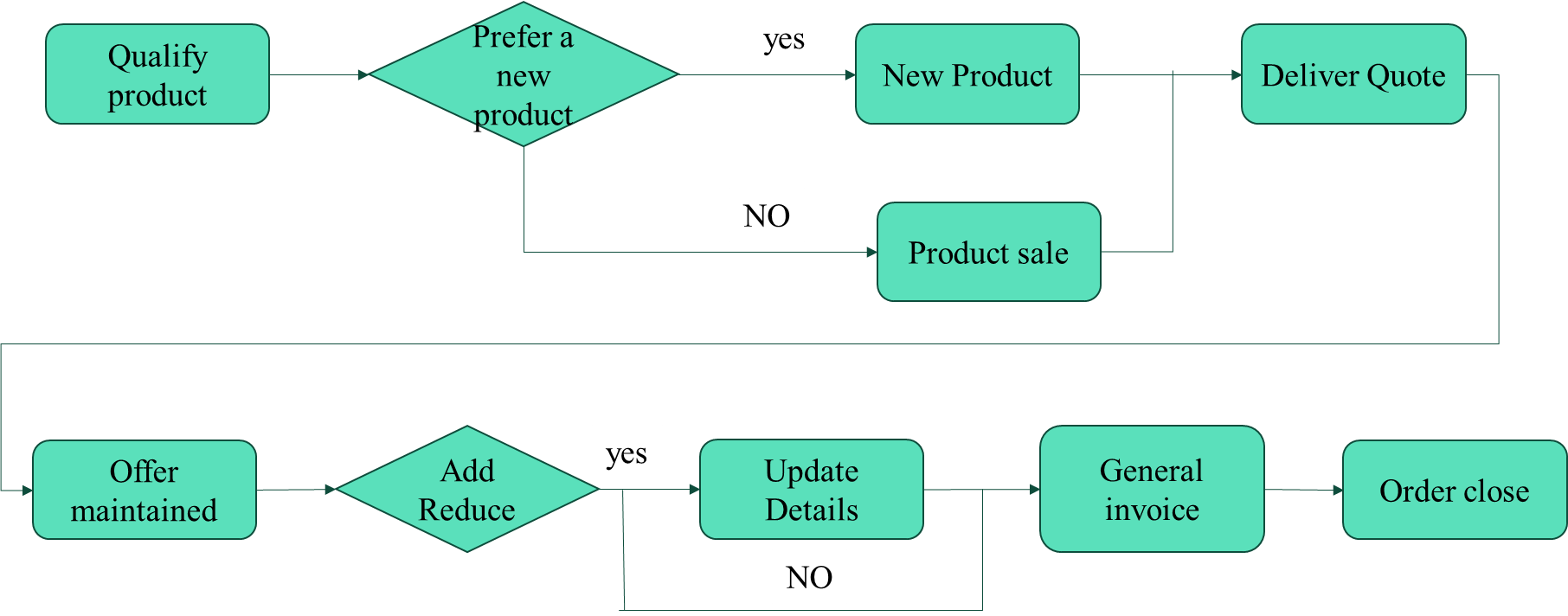


**3.2. FLOW DIAGRAM:**

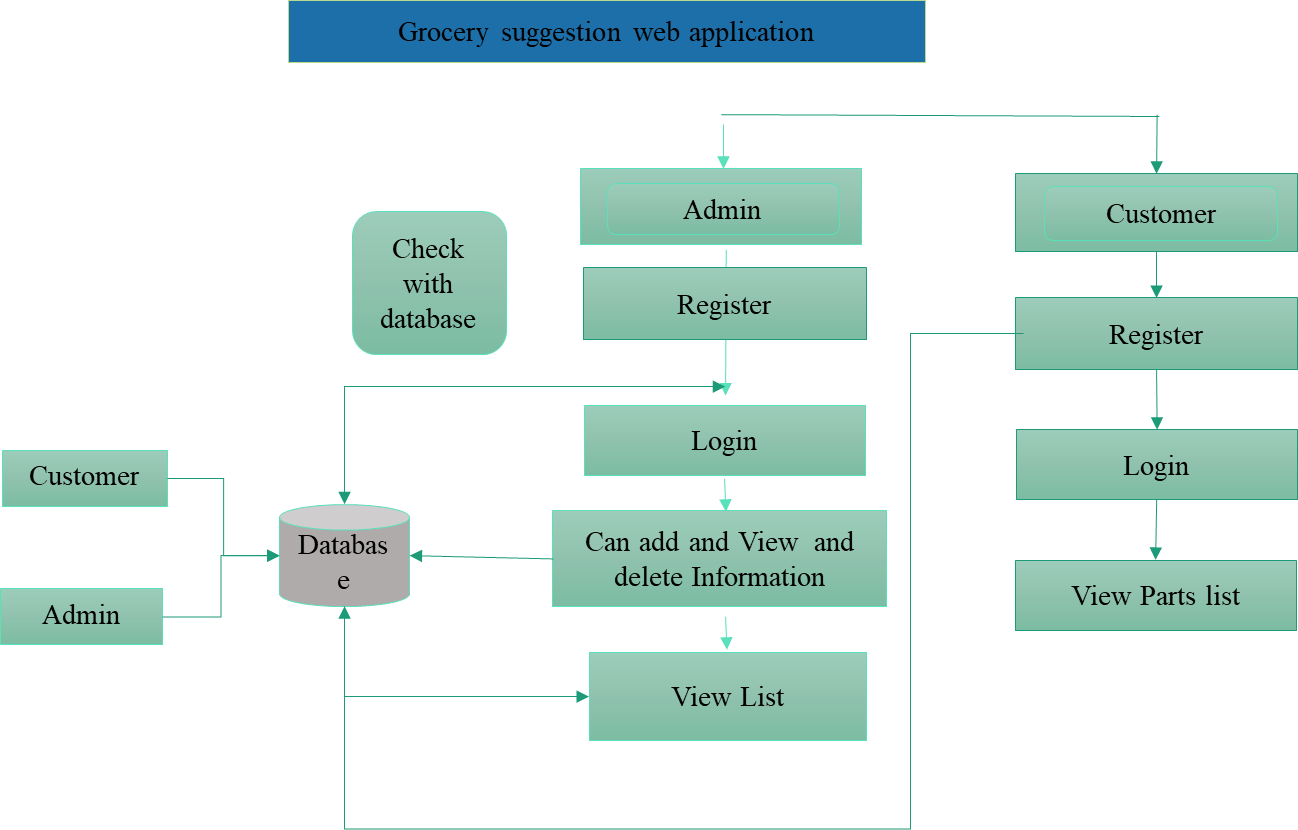
**3.2.1. Flow Diagram (level-0)**



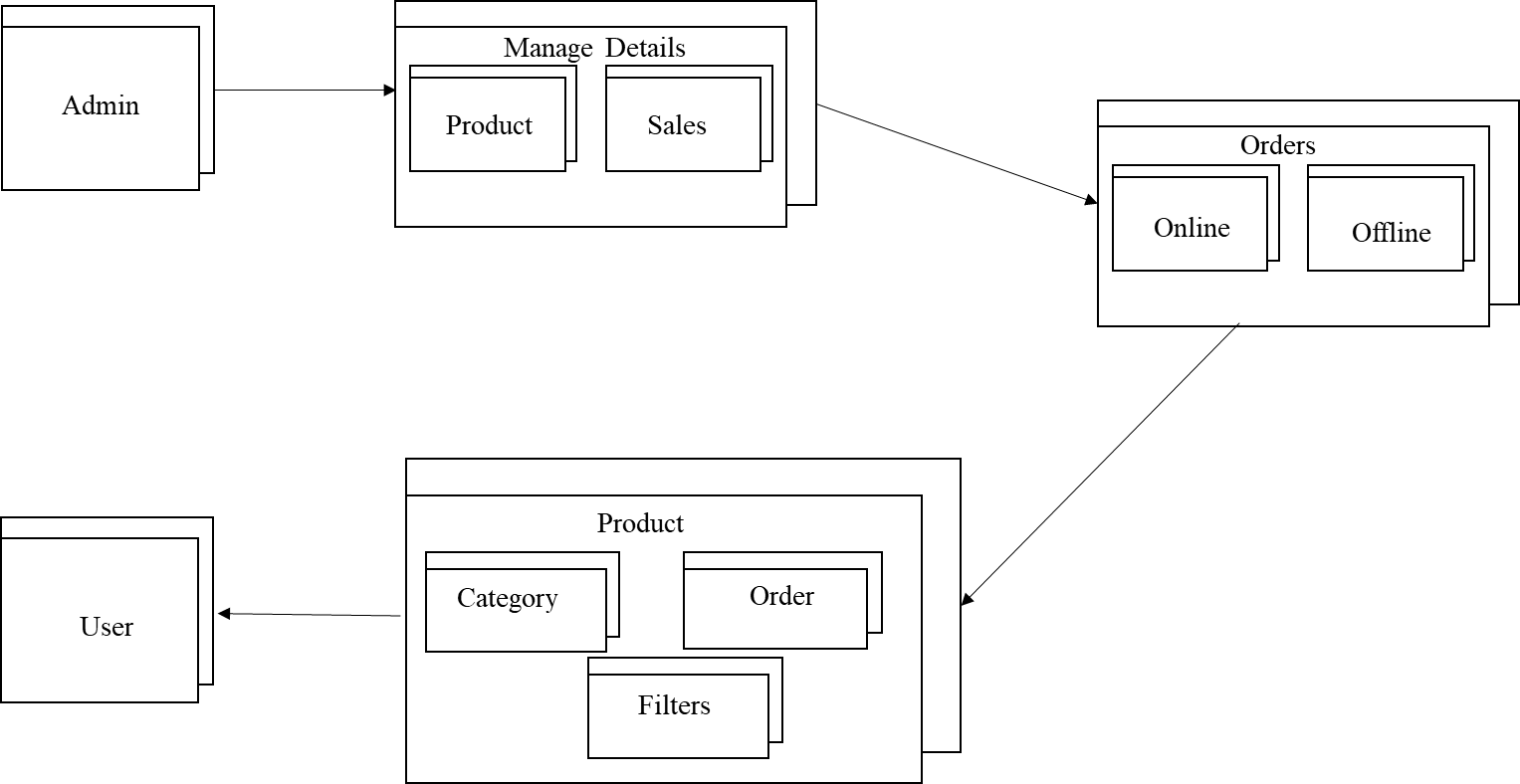
**3.2.2. Flow Diagram (level-1)**



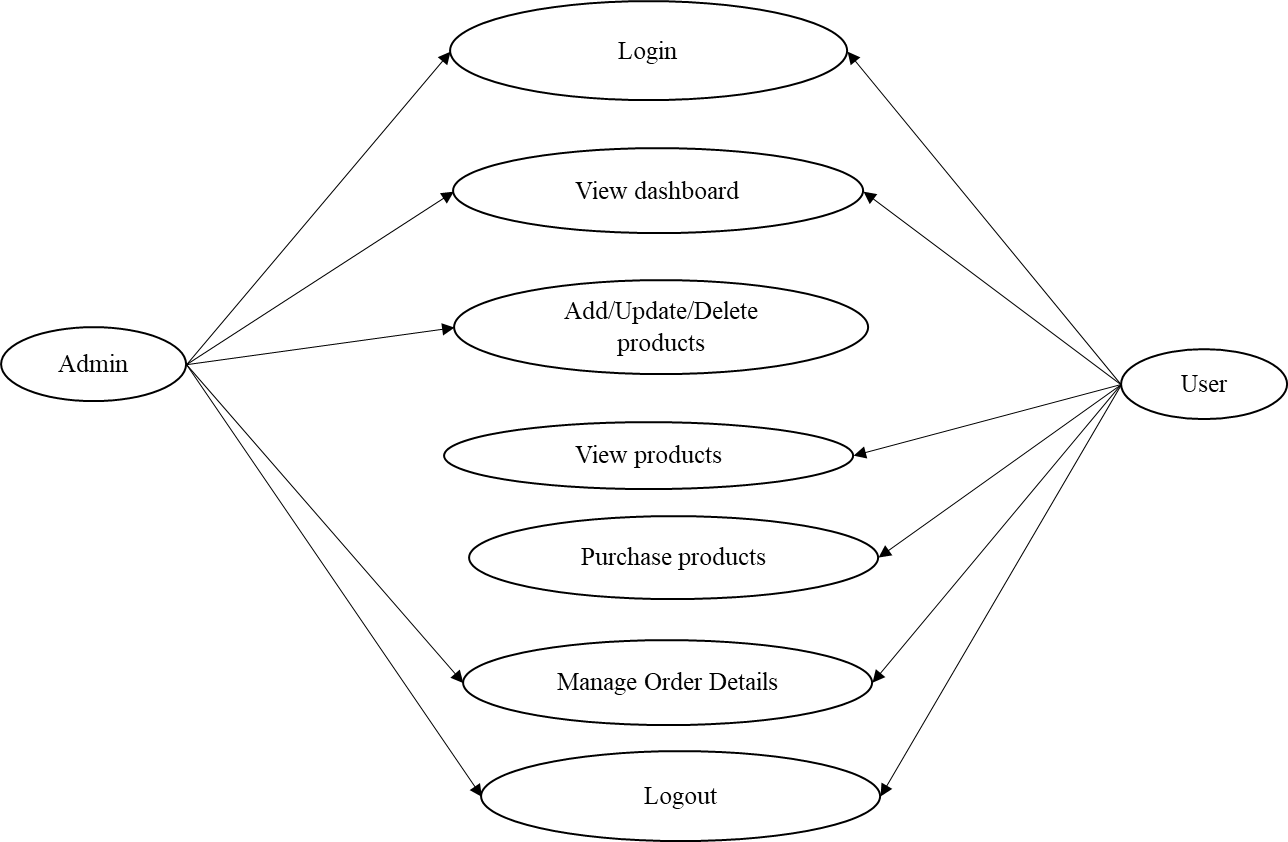
**3.2.2. Flow Diagram (level-2)**



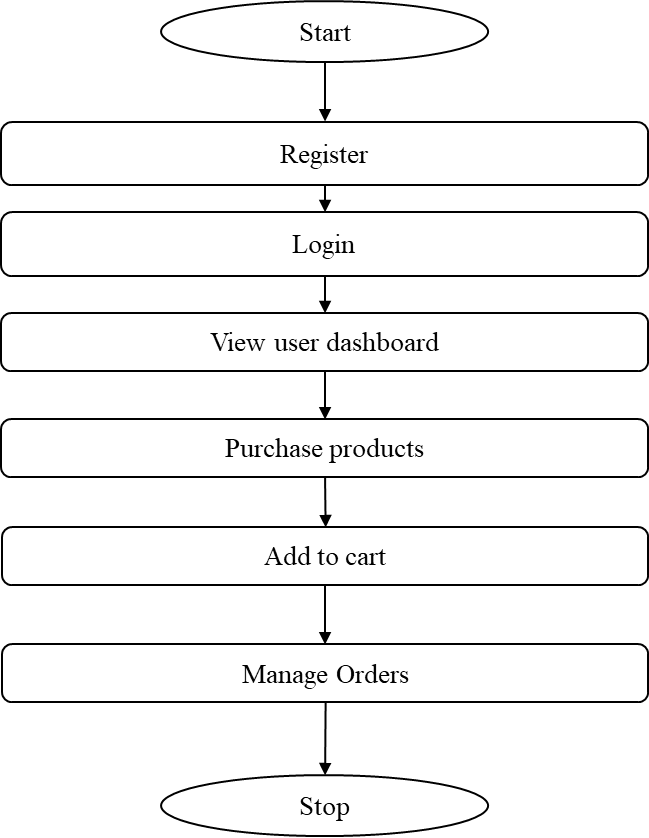
**3.3. UML Diagram:**



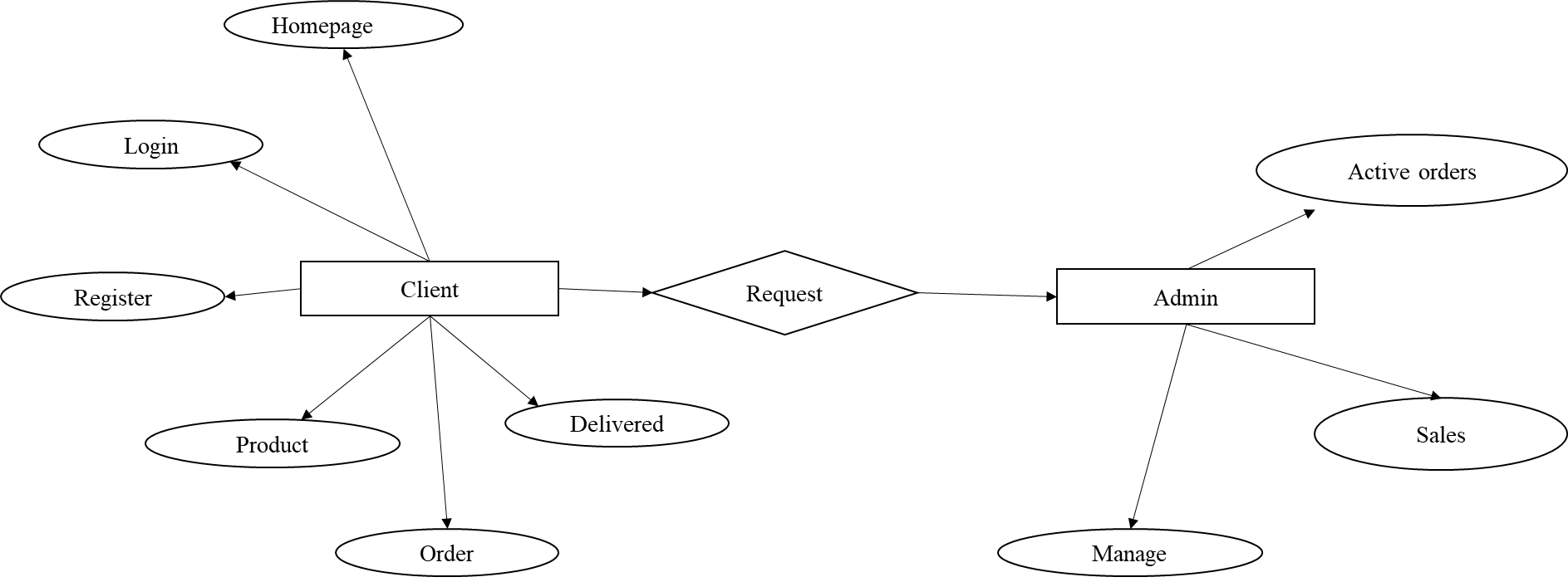
**3.3.1. Use case diagram:**



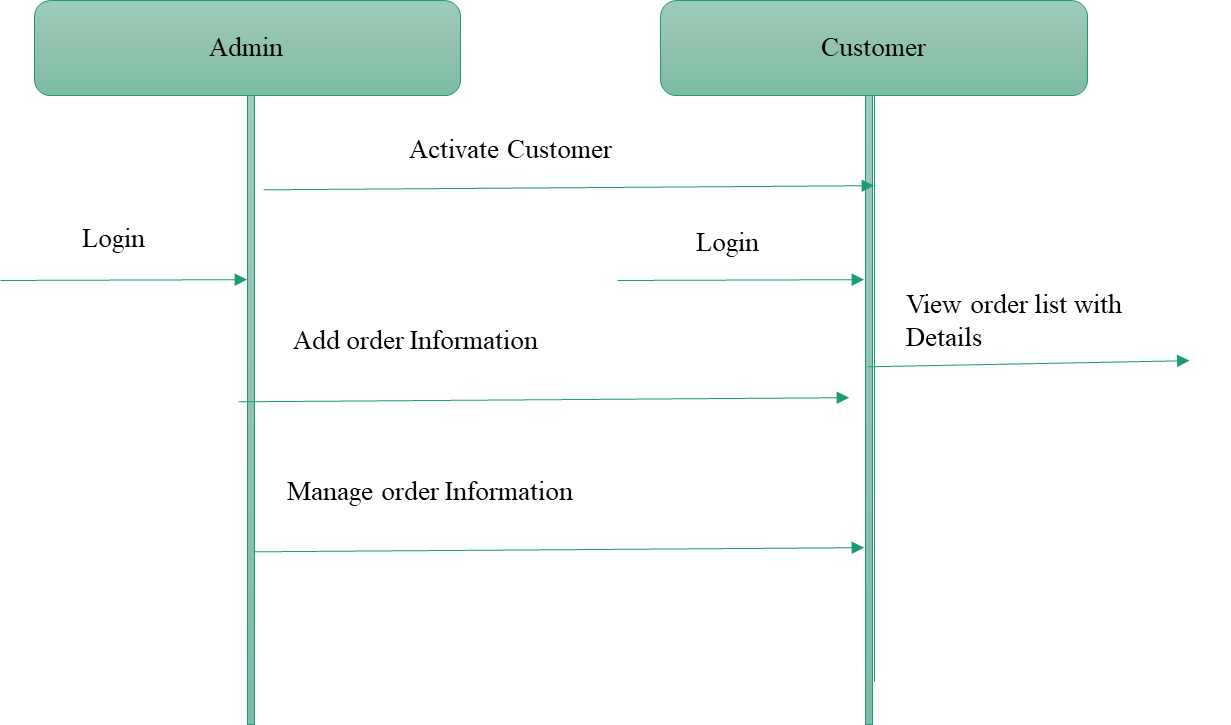
**3.4. Activity diagram:**



**3.5. E-R Diagram:**



**3.6 SEQUENCE DIAGRAM:**



**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Modules**

* Admin module
* User module

**4.2 MODULE DESCRIPTION**

**ADMIN MODULE**

* **Dashboard** – For the admin dashboard, you will be able to all the basic access in the whole system. Such as products, orders, users and categories.
* **Manage Products** – The admin has access to the product management information system. He can add, update and delete the product.
* **Manage Orders** – As the main functions of the admin, the admin can see if the order is already paid or not from the customers on a case to case basis.
* **Manage Categories** – For the categories, the admin has the features of managing the category. The example category used in this system is Snacks, Fruits, and Vegetables.  
  **Manage Users** – The admin can manage the user’s account. Admin can add, update and delete users in the system.
* **Login and Logout** – By default one of the security features of this system is the secure login and logout system.

**USER MODULE**

* **Register page-** If a user does not have an account when they log in, they will be routed to the registration page. Once an account is created, the user will then be directed to the home page.
* **Login Page-** The process is fairly simple; users input their credentials on the website's login form. That information is then sent to the authentication server where the information is compared with all the user credentials on file. When a match is found, the system will authenticate users and grant them access to their accounts
* **Home Page** – On the home page, you can see directly the list of products for sale and you can filter the result based according to category.
* **Viewing Products** – by default on the frontend, the customer can automatically view all the products.
* **Checkout Order**– The customer can checkout order in the frontend that can confirmed by the admin in the backend.
* **Suggestion part:** Health condition will enrolled as suggest the grocery as used the pick customized the product will suggestion the particular item, then purchased product.
* **Customers Profile** – In the frontend, the customer can register their profile. In short, this system has a customer management system feature.
* **Add to Cart**– One of the features of this system is that, wherein the customer can temporarily add their order in the add to cart
* **Update Accounts-** In addition, the user may access their bank accounts, alter their profiles, change their passwords, and delete their personal account.
* **List of Orders-**The customer can view the list of order and edit the cart can control the remove and add the products.

**CHAPTER 5**

**SYSTEM REQUIREMENTS**

**5.1 HARDWARE REQUIREMENTS:**

Processor Name : Intel core 4

Processor Speed : 3.2 GHz

RAM : 8 GB

Hard Disk Capacity : 128 GB

Display Device : 14’ to 19’ Inch Monitor

Keyboard Type : PS2 or USB

Mouse Type : PS2 or USB

**5.2 SOFTWARE REQUIREMENTS:**

Language Used : Node JS

Server : Node JS

Database : Mongo dB

User Interface Design : React JS

Web Browser : Chrome

5.3 **SOFTWARE DESCRIPTION**

## **INTRODUCTION TO JAVASCRIPT:**

An explanation of exactly what JavaScript is has to begin with Java. Java is a new kind of Web programming language developed by Sun Microsystems. A Java program, or applet, can be loaded by an HTML page and executed by the Java Interpreter, which is embedded into the browser.

Java is a complex language, similar to C++. Java is object-oriented and has a wide variety of capabilities; it's also a bit confusing and requires an extensive development cycle. That's where JavaScript comes in.

JavaScript is one of a new breed of Web languages called scripting languages. These are simple languages that can be used to add extra features to an otherwise dull and dreary Web page. While Java is intended for programmers, scripting languages make it easy for nonprogrammers to improve a Web page.

JavaScript was originally developed by Netscape Corporation for use in its browser, Netscape Navigator. It includes a convenient syntax, flexible variable types, and easy access to the browser's features. It can run on the browser without being compiled; the source code can be placed directly into a Web page.

You can program in JavaScript easily; no development tools or compilers are required. You can use the same editor you use to create HTML documents to create JavaScript, and it executes directly on the browser (currently, Netscape or Microsoft Internet Explorer).

JavaScript was originally called Live Script, and was a proprietary feature of the Netscape browser. JavaScript has now been approved by Sun, the developer of Java, as a scripting language to complement Java. Support has also been announced by several other companies.

Although useful in working with Java, you'll find that JavaScript can be quite useful in its own right. It can work directly with HTML elements in a Web page, something Java can't handle. It is also simple to use, and you can do quite a bit with just a few JavaScript statements.

**The Advantages of JavaScript**

**An Interpreted Language**: JavaScript is an interpreted language, which requires no compilation steps. This provides an easy development process. The syntax is completely interpreted by the browser just as it interpreted HTML tags.

**Embedded Within HTML**: JavaScript does not requires any special or separate editor for programs to be written edited or compiled. It can be written in any text editor like Notepad, along with appropriate HTML tags, and saved as filename.html.HTML files with embedded JavaScript commands can then be read and interpreted by any browser that is JavaScript enabled.

**Minimal Syntax-Easy to Learn:** By learning just a few commands and simple rules of syntax, complete applications can be built using JavaScript.

**Quick Development:** Because JavaScript does not require time-consuming compilations, scripts can be developed in a short period of time. This is enhanced by the fact many GUI interface features, such as alerts, prompts, confirm boxes, and other GUI elements, are handle by client side JavaScript, the browser and HTML code.

**Design for Simple, Small Programs:** It is well suited to implement simple, small programs (for example, a unit conversion calculator between miles and kilometres or pounds and kilograms).Such programs can be easily written and executed at an acceptable speed using JavaScript. In addition, they can be easily interpreted into a web page.

**Performance:** JavaScript can be written such that the HTML files are fairly compact and quite small. This minimizes storage requirements on the web server and download time for the client.

Additionally, because JavaScript are usually include in the same file as the HTML code for a web page, they require fewer separate network accesses.

**Procedural Capabilities:** Every programming language needs to support facilities such as Condition checking, Looping and Branching .JavaScript provides syntax, which can be used to add such procedural capabilities to web page (filename.html) coding.

**Designed for Programming User Events:** JavaScript supports Object/Events based programming JavaScript recognizes when a form **Button** is pressed. This event can have suitable JavaScript code attached, which will executed when the **Button Pressed** event occurs.

JavaScript can be used to implement context sensitive help. Whenever an HTML form’s **Mouse** cursor **Mouse Over** a button or a link on the page a helpful and informative massage can be displayed in the status bar at the button of the browser window.

**Easy Debugging and Testing :** Being an interprets language ,scripts in JavaScript are tested line by line, and the errors are also listed as they are encountered ,i.e. an appropriate error message along with the line number is listed for every error that is encountered. It is thus easy to locate errors, make changes, and test it again without the overhead and delay of compiling.

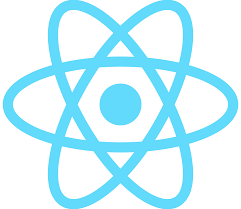
**Platform Independence / Architecture Neutral:** JavaScript is a programming language that is completely independent of the hardware on which it works. It is a language that is understood by any JavaScript enabled browser .Thus ,JavaScript application work on any machine that has an appropriate JavaScript enabled browser can be anywhere on the network.

Since each browser is for a specific platform, JavaScript interpretation will be with respect to the specific platform. The browser will add whatever platform specific

Information is required to the JavaScript while it interprets the code. Thus, JavaScript is truly platform independent. A JavaScript programmer developed on a UNIX machine will work perfectly well on a Windows machine.

The fact that a platform specific browser , maintained at the client end, does the interpretation of JavaScript , relieves the developer of the responsibility of maintaining multiple source code files for multiple platform

**React JS**



**React JS History:**

* Current version of React.JS is V18.0.0 (April 2022).
* Initial Release to the Public (V0.3.0) was in July 2013.
* React.JS was first used in 2011 for Facebook's Newsfeed feature.
* Facebook Software Engineer, Jordan Walke, created it.
* Current version of create-react-app is v5.0.1 (April 2022).
* create-react-app includes built tools such as webpack, Babel, and ESLint.

React JS is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and manage the application state efficiently.

**Some key features of React JS include:**

Virtual DOM: React uses a virtual DOM to improve performance by minimizing direct manipulations of the actual DOM.

Component-based architecture: React follows a component-based architecture, where UI elements are divided into independent, reusable components.

JSX: React uses JSX (JavaScript XML) syntax, which allows developers to write HTML-like code within JavaScript. JSX is then compiled to JavaScript code.

One-way data flow: React follows a one-way data flow, where data flows from parent components to child components. This helps in managing the state of the application.

React router: React-router is a library that helps in managing the navigation and routing in a React application.

State management: React provides a way to manage the application state using the useState and useEffect hooks. These hooks allow developers to manage and update the state of a component.

React JS is widely used in front-end development and is known for its simplicity, reusability, and performance optimization techniques. It has a large and active community, which provides support and contributes to the development of various libraries and tools for React.

## **Setting up a React Environment:**

If you have npx and Node.js installed, you can create a React application by using create-react-app.

If you've previously installed create-react-app globally, it is recommended that you uninstall the package to ensure npx always uses the latest version of create-react-app.

To uninstall, run this command: npm uninstall -g create-react-app.

Run this command to create a React application named my-react-app:

* npx create-react-app my-react-app

The create-react-app will set up everything you need to run a React application.

## **Run the React Application:**

Now you are ready to run your first real React application!

Run this command to move to the my-react-app directory:

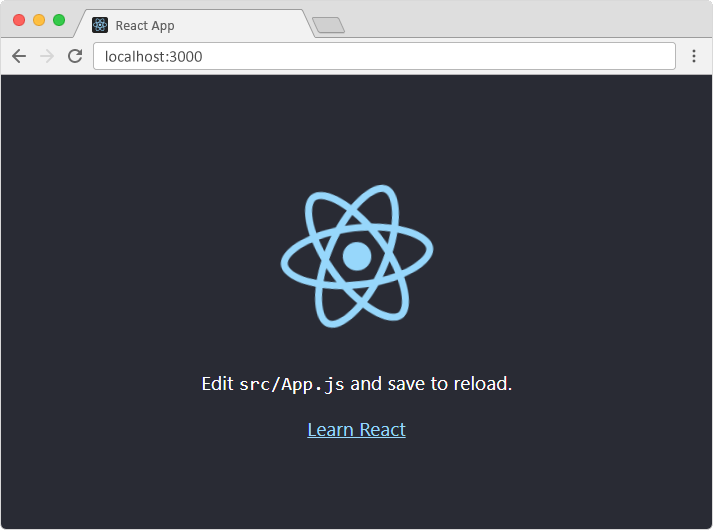
* cd my-react-app

Run this command to run the React application my-react-app:

* npm start

A new browser window will pop up with your newly created React App! If not, open your browser and type localhost:3000 in the address bar.

The result:



**Hooks:**

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Although Hooks generally replace class components, there are no plans to remove classes from React.

## **What is a Hook?**

Hooks allow us to "hook" into React features such as state and lifecycle methods.

### **Example:**

import React, { useState } from "react";

import ReactDOM from "react-dom/client";

function FavoriteColor() {

const [color, setColor] = useState("red");

return (

<>

<h1>My favorite color is {color}!</h1>

<button

type="button"

onClick={() => setColor("blue")}

>Blue</button>

<button

type="button"

onClick={() => setColor("red")}

>Red</button>

<button

type="button"

onClick={() => setColor("pink")}

>Pink</button>

<button

type="button"

onClick={() => setColor("green")}

>Green</button>

</>

);

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<FavoriteColor />);

You must import Hooks from react.

Here we are using the useState Hook to keep track of the application state.

State generally refers to application data or properties that need to be tracked.

## **Hook Rules**

There are 3 rules for hooks:

* Hooks can only be called inside React function components.
* Hooks can only be called at the top level of a component.
* Hooks cannot be conditional

**Note:** Hooks will not work in React class components.

## **Custom Hooks**

If you have stateful logic that needs to be reused in several components, you can build your own custom Hooks.

We'll go into more detail in the [Custom Hooks section](https://www.w3schools.com/REACT/react_customhooks.asp).

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

useState:

To use the useState Hook, we first need to import it into our component.

Example:

At the top of your component, import the useState Hook.

* import { useState } from "react";

Notice that we are destructuring useState from react as it is a named export.

To learn more about destructuring, check out the [ES6 section](https://www.w3schools.com/REACT/react_es6.asp).

Initialize useState

We initialize our state by calling useState in our function component.

useState accepts an initial state and returns two values:

The current state.

A function that updates the state.

Example:

Initialize state at the top of the function component.

import { useState } from "react";

function FavoriteColor() {

const [color, setColor] = useState("");

}

-------------------------------------------\*\*\*\*\*\*\*------------------------------------------

Express JS:



Express.js is a web application framework for Node.js that allows developers to build web applications and APIs quickly and easily. It provides a simple and flexible way to handle HTTP requests and responses, as well as middleware functions for adding additional functionality to your applications.

**Some of the key features of Express.js include:**

Routing: Express.js allows you to define routes for different HTTP methods and URLs, making it easy to handle different requests. You can also handle parameters and query strings in your routes.

Middleware: Express.js uses middleware functions to process requests before they reach the route handlers. Middleware can be used for tasks like authentication, request validation, logging, and more.

Template engines: Express.js supports a wide variety of template engines, allowing you to render dynamic HTML pages on the server and send them to the client.

Error handling: Express.js provides a default error handling mechanism, allowing you to handle errors and send appropriate responses to the client. You can also define your own error handlers for specific routes or middleware.

Static file serving: Express.js makes it easy to serve static files, such as HTML, CSS, and images, directly from the file system.

Overall, Express.js is a lightweight and flexible framework that provides all the tools you need to build robust web applications with Node.js. It has a large and active community, with numerous plugins and middleware available to extend its functionality.

**Why Express ?**

Develops Node.js web applications quickly and easily.

It’s simple to set up and personalise.

Allows you to define application routes using HTTP methods and URLs.

Includes a number of middleware modules that can be used to execute additional requests and responses activities.

Simple to interface with a variety of template engines, including Jade, Vash, and EJS.

Allows you to specify a middleware for handling errors.

Installing Express:

We can install it with npm. Make sure that you have [Node.js](https://www.geeksforgeeks.org/installation-of-node-js-on-windows/)and[npm](https://www.geeksforgeeks.org/node-js-npm-node-package-manager/)installed.

Step 1: Creating a directory for our project and make that our working directory.

* $ mkdir gfg
* $ cd gfg

Step 2: Using npm init command to create a package.json file for our project.

* $ npm init

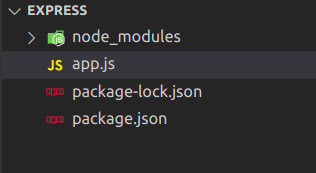
This command describes all the dependencies of our project. The file will be updated when adding further Installing Express

Step 3: Now in your *gfg(name of your folder)* folder type the following command line:

* $ npm install express --save

Now let’s understand the working of express.js through an example.

Project Structure: It will look like the following.



Example: Write the following code in app.js.

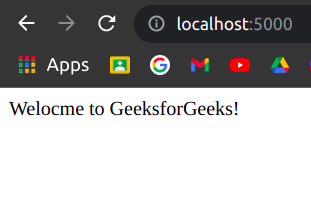
app.js

|  |
| --- |
| var express = require('express');  var app = express();  app.get('/', function (req, res) {  res.send("Welocme to GeeksforGeeks!");  });  app.listen(5000); |

Step to run the application: Start the app by typing following command.

* node app.js

Output:



Request Object Methods

Following is a list of some generally used request object methods:

**req.accepts (types)**

This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.

**Examples:**

req.accepts('html');

//=>?html?

req.accepts('text/html');

// => ?text/html?

req.get(field)

This method returns the specified HTTP request header field.

**Examples:**

req.get('Content-Type');

// => "text/plain"

req.get('content-type');

// => "text/plain"

req.get('Something');

// => undefined

**req.is(type)**

This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.

**Examples:**

// With Content-Type: text/html; charset=utf-8

req.is('html');

req.is('text/html');

req.is('text/\*');

// => true

req.param(name [, defaultValue])

This method is used to fetch the value of param name when present.

**Examples:**

// ?name=sasha

req.param('name')

// =**>** "sasha"

// POST name=sasha

req.param('name')

// =**>** "sasha"

// /user/sasha for /user/:name

req.param('name')

// =**>** "sasha"

**Node js:**



**Node.js is a runtime environment that allows developers to run JavaScript outside of a web browser. It uses the V8 JavaScript engine by Google, which makes it a powerful and efficient tool for building server-side applications. Node.js is particularly well-suited for building scalable network applications and is widely used for web servers, real-time applications, and command-line scripts. It provides a vast ecosystem of packages and modules through the npm package manager, which allows developers to easily add functionality to their applications.**

## **What is Node.js?**

* Node.js is an open source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

## **Why Node.js?**

**Node.js uses asynchronous programming!**

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

## **What Can Node.js Do?**

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

## **What is a Node.js File?**

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

## **Download Node.js**

The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)

## **Getting Started**

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.

Create a Node.js file named "myfirst.js", and add the following code:

myfirst.js

var http = require('http');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);

Save the file on your computer: C:\Users\Your Name\myfirst.js

The code tells the computer to write "Hello World!" if anyone (e.g. a web browser) tries to access your computer on port 8080.

For now, you do not have to understand the code. It will be explained later.

## **Command Line Interface**

Node.js files must be initiated in the "Command Line Interface" program of your computer.

How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

Navigate to the folder that contains the file "myfirst.js", the command line interface window should look something like this:

C:\Users\Your Name>\_

## **Initiate the Node.js File**

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write node myfirst.js and hit enter:

Initiate "myfirst.js":

C:\Users\Your Name>node myfirst.js

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: [http://localhost:8080](http://localhost:8080/)

Use the module "myfirstmodule" in a Node.js file:

var http = require('http');  
**var dt = require('./myfirstmodule');**  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write("The date and time are currently: " + **dt.myDateTime()**);  
  res.end();  
}).listen(8080);

# **Node.js Timer**

Node.js Timer functions are global functions. You don't need to use require() function in order to use timer functions. Let's see the list of timer functions.

**Set timer functions:**

* **setImmediate():** It is used to execute setImmediate.
* **setInterval():** It is used to define a time interval.
* **setTimeout():** ()- It is used to execute a one-time callback after delay milliseconds.

Node.js Streams

Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

**Readable:**This stream is used for read operations.

**Writable:**This stream is used for write operations.

**Duplex:**This stream can be used for both read and write operations.

**Transform:**It is type of duplex stream where the output is computed according to input.

Each type of stream is an Event emitter instance and throws several events at different times. Following are some commonly used events:

**Data:**This event is fired when there is data available to read.

**End:**This event is fired when there is no more data available to read.

**Error:**This event is fired when there is any error receiving or writing data.

**Finish:**This event is fired when all data has been flushed to underlying system.

**Mongodb:**



MongoDB is a NoSQL document database that is based on the concept of collections and documents. It stores data in a flexible schema format called BSON (Binary JSON), allowing for dynamic and nested data structures. MongoDB provides high scalability and performance, making it suitable for large-scale applications that require fast and efficient data access.

Some key features of MongoDB include:

Flexible Data Model: MongoDB's flexible document model allows for easy representation of complex data structures, with support for nested arrays and documents.

Replication and High Availability: MongoDB supports replica sets, which are groups of database nodes that maintain multiple copies of the data for high availability and data redundancy.

Sharding: MongoDB can distribute data across multiple machines using sharding, which allows for horizontal scaling and improved performance.

Indexing and Querying: MongoDB supports indexing on any field and provides powerful query capabilities, including support for complex joins and aggregations.

Transactions: MongoDB now supports multi-document transactions, allowing for consistent and atomic operations across multiple documents.

Horizontal Scalability: MongoDB can be scaled horizontally by adding more nodes to a cluster, enabling the database to handle increased traffic and workload.

MongoDB uses a JSON-like query language called MongoDB Query Language (MQL) for querying and manipulating data. It also provides drivers for various programming languages, making it easy to integrate MongoDB into different applications and frameworks.

Overall, MongoDB is a popular choice for developers and organizations due to its ease of use, flexibility, and scalability, making it well-suited for a wide range of use cases, including web and mobile applications, content management systems, real-time analytics, and IoT (Internet of Things) applications.

Create a Database

From the [Databases](https://www.mongodb.com/docs/compass/current/databases/#std-label-database-tab) tab, click the **Create Database** button to bring up the **Create Database** dialog.

Enter database and first collection information.

In the dialog, enter the name of the database to create and its first collection. Both the database name and the collection name are required.

If you want to create a [capped collection](https://www.mongodb.com/docs/manual/core/capped-collections/), select the **Capped Collection** checkbox and enter the maximum bytes.

If you want to use [custom collation](https://www.mongodb.com/docs/manual/reference/collation/#collation-document) on the collection, select the **Use Custom Collation** checkbox and select the desired collation settings.

If your deployment is connected using **In-Use Encryption**, you can use [Queryable Encryption](https://www.mongodb.com/docs/v6.0/core/queryable-encryption/) on the newly created collection. Check the **Queryable Encryption** option and indicate the following encryption properties:

[Encrypted Fields.](https://www.mongodb.com/docs/v6.0/core/queryable-encryption/fundamentals/encrypt-and-query/)

(Optional) [KMS Provider.](https://www.mongodb.com/docs/v6.0/core/queryable-encryption/fundamentals/kms-providers/)

(Optional) [Key Encryption Key.](https://www.mongodb.com/docs/v6.0/core/queryable-encryption/fundamentals/keys-key-vaults/)

Click *Create Database* to create the database and its first collection.

Drop a Database

Click the trash can icon for the database.

From the [Databases](https://www.mongodb.com/docs/compass/current/databases/#std-label-database-tab) tab, to delete a database, click on the trash can icon for that database. A confirmation dialog appears.

Confirm the database to delete.

In the dialog, enter the name of the database to delete.

Click *Drop Database* to delete the database.

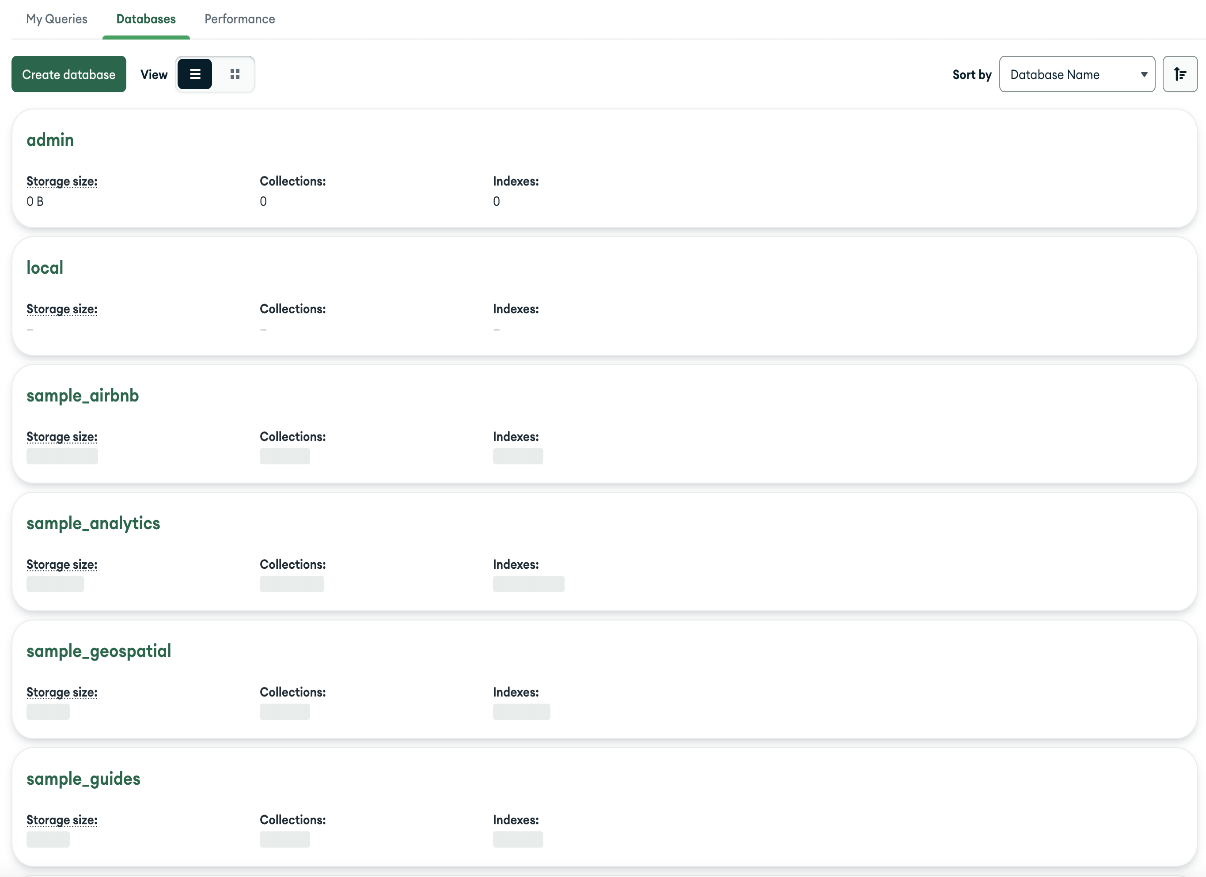
Limitations

Creating and dropping databases is not permitted in **MongoDB Compass Readonly Edition**.

Creating databases is not permitted if you are connected to a [Data Lake.](https://www.mongodb.com/docs/atlas/data-lake/)

Databases Tab

The **Databases** tab lists the existing databases for your MongoDB deployment. To access the **Databases** tab, click the cluster name in the upper left corner and select the **Databases** tab.



**5.4 TESTING OF PRODUCT**

**Unit testing in Node.js**

Unit testing is a software testing method in which individual pieces of code (usually the smallest piece of code that can be logically isolated in a system) are tested in isolation. Unit tests should be isolated so that there are no external dependencies.

Let’s look at some advantages associated with unit testing.

First, unit testing makes it easier to identify bugs in code. Appropriate test cases should be written for every piece of code to ensure that they meet specifications and provide the desired output. Any changes that result in failing tests will indicate that an error or bug has been introduced. Additionally, unit testing makes it easier to narrow down the offending piece of code.

Second, unit tests act as self-documentation. A new team member can gain a better understanding of the code base by going through unit tests.

Third, the debugging process is made a lot easier. This is because when the test fails, the focus will be on the latest changes made.

Fourth, refactoring code is made easier, because changes can be verified using tests to ensure that the unit being tested still behaves in the desired manner.

Finally, costs that would be incurred fixing bugs or system outages are reduced.

Testing frameworks provide a set of reusable components or modules, such as test runners and utilities, for running automated tests. The testing framework is responsible for:

1. Describing the format used to convey test expectations
2. Creating a way of accessing the application or code to be tested
3. Executing the tests
4. Reporting test results

They are particularly useful when tests are an integral part of your continuous integration process. Frameworks are built for a specific type of testing: unit, integration, functional, or combinations of these.

**What makes a good Node.js testing framework?**

There are a thousand and one testing frameworks out there. To pick something that works for your use case, you need to evaluate each framework based on your project needs and how effective you consider it to be for your team.

Below are six key characteristics of a strong Node.JS testing framework:

1. Ease of setup: getting up and running with your tests should take a minimal amount of effort
2. Well-supported: there is plenty of excellent documentation and communities to get help
3. Wide array of feature sets: the framework has things such as matchers, spies, and mocking built in
4. Speed: for tests that are CPU-bound, choosing the right framework can save you a lot of time during test runs
5. Ease of reporting: coverage reports should be easy to generate using built-in reporting and external reporting libraries should be easy to integrate
6. Ease of integration: a good testing library should be easy to integrate into your continuous integration process

We’ll compare four of these Node.js unit testing frameworks:

* Mocha
* Jest
* Jasmine
* AVA

## Mocha

[Mocha](https://mochajs.org/) has been around for quite a while; it was initially released in November 2011. However, unlike other frameworks like Jest and Jasmine, it relies on third-party assertions, mocking, and spying tools like [Sinon](https://sinonjs.org/) and [Chai](https://www.chaijs.com/). It is very extensible and has a lot of plugins, extensions, and libraries designed to run on top of it.

### Pros

* Highly extensible with support for various assertion and mocking libraries
* Easy asynchronous testing
* Adding support for generators to test suites is relatively easy. Using the [co-mocha](https://www.npmjs.com/package/co-mocha) package, all you have to do is require it in your tests and you’re ready to use generators
* Supported by some CI servers and plugins for others

### Cons

* The use of extra libraries can introduce configuration complexity and increases maintenance work
* No auto-mocking

## Jest

[Jest](https://jestjs.io/) is a JavaScript testing framework developed and regularly maintained by Facebook. Its popularity has grown steadily since 2016, when only six percent of respondents to that year’s “[State of JS](http://2016.stateofjs.com/2016/testing/)” survey said they had used Jest before and would use it again. This figure climbed to [a quarter of respondents in 2017](https://2017.stateofjs.com/2017/testing/results/) before reaching [40 percent in 2018](https://2018.stateofjs.com/testing/overview/). As of the most recent edition, a whopping 73 percent of JavaScript developers had tried Jest and plan to use it again.

### Pros

* Comprehensive [documentation](https://jestjs.io/docs/en/getting-started) includes detailed instructions to help you set up testing, write various types of tests, and use its many features, as well as great examples
* Easy setup with flexible and easy configuration and less boilerplate code than other frameworks
* Parallel test running enabled
* Optimal performance: tests are parallelized by running them in their own processes to maximize performance
* Useful features such as snapshots, coverage, and test watching

### Cons

* Displays multiple messages for the same error
* It can require more dependencies during initial setup (e.g., [Babel](https://blog.logrocket.com/why-you-dont-need-babel/))

## Jasmine

Developed by Pivotal Labs and released in 2010, Jasmine has been around for a lot longer than Jest. It aims to run on any JavaScript-enabled platform and is highly flexible and compatible with a variety of other testing frameworks and libraries, including Sinon and Chai. Due to its longevity, it has developed a significant community and enjoys ample support with loads of libraries, blog articles, and tutorials.

### Pros

* Simple to set up — Jasmine has a CLI tool that creates a spec folder and a JSON configuration file, so with one command you’re ready to start testing your code
* Thoroughly tested, documented, and supported by numerous tutorials on how to use it
* Behaviour-driven development focused with descriptive syntax
* Supported by many CI servers with plugins available for those that don’t have out-of-the box support

### Cons

* Unfriendly error logs
* Test files must have a specific suffix (e.g., spec.js)
* Assertion library is not as rich as Chai

## AVA

Minimalism is the focus of AVA. It has a simple API while still supporting advanced features. It achieves its blazing speed by running tests in parallel as separate Node processes. Unlike other testing frameworks such as Jest and Jasmine, it does not create test global.

### Pros

* Easy to use. To install and setup AVA, all you have to do is run npm init ava
* Parallel test running
* Native ES6/ES7 support
* Built-in support for async functions
* If a promise is returned, you don’t need to end the test yourself; it will end when the promise resolves

### Cons

* AVA is relatively new. The community is still growing and there isn’t a lot of documentation or tutorials like other testing frameworks
* AVA has a lot of open issues on GitHub

#### Jenkins

[Jenkins](https://www.softwaretestinghelp.com/integration-of-jenkins-with-selenium-webdriver/) is a continuous integration system that can hook into your version control (e.g. git) and automatically execute mocha any time a commit occurs. This means your product is being tested every time a change occurs.

**FEASIBILITY TESTING**

Feasibility study is carried out based on many purposes to analyse whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

**Types of Feasibility Study:**   
The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility –**   
   In Technical Feasibility current resources both hardware software along with required technology are analysed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyses technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.
2. **Operational Feasibility –**   
   In Operational Feasibility degree of providing service to requirements is analysed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.
3. **Economic Feasibility –**   
   In Economic Feasibility study cost and benefit of the project is analysed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analysed whether project will be beneficial in terms of finance for organization or not.
4. **Legal Feasibility –**   
   In Legal Feasibility study project is analysed in legality point of view. This includes analysing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

### **5 .Schedule Feasibility**

This assessment is the most important for [project success](https://www.simplilearn.com/how-to-make-a-project-successful-article); after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

When these areas have all been examined, the feasibility analysis helps identify any constraints the proposed project may face, including:

* Internal Project Constraints: Technical, Technology, Budget, Resource, etc.
* Internal Corporate Constraints: Financial, Marketing, Export, etc.
* External Constraints: Logistics, Environment, Laws, and Regulations, etc.

**Aim of feasibility study:**

* The overall objective of the organization are covered and contributed by the system or not.
* The implementation of the system be done using current technology or not.
* can the system be integrated with the other system which are already exist.
* Improves project teams’ focus
* Identifies new opportunities
* Provides valuable information for a “go/no-go” decision
* Narrows the business alternatives

**CHAPTER-6**

**CONCLUSION**

As we can see, online shopping may or may not be greener than traditional shopping. There are simply way too many factors that we have to consider in such a model.

As we focus on the costs of online shopping, it seems that online shopping is really detrimental for the environment. Online shopping brings us great convenience, but it also encourages irresponsible consumption habits like exploiting the advantages of free returns and expedited shipping. These add on to the existing pool of environmental problems that we are dealing with – global warming, wastes and pollution. Therefore, we should change our attitude towards e-commerce – to be more responsible, less exploitative and more thoughtful for the environment.

**CHAPTER-7**

**FUTURE ENHANCEMENT**

Predictive maintenance: AI and ML can be used to analyses data and predict when maintenance is needed, which can help to reduce downtime and costs for customers.

Personalized recommendations: AI and ML can be used to personalize recommendations for car sales and services based on customer needs and preferences.

Chabot’s: AI-powered Chabot’s can be used to provide 24/7 customer support and answer questions.

**CHAPTER-8**

**SOURCE CODE**

**App.js**

import React, { useEffect } from "react";

import "./App.css";

import { BrowserRouter, Route, Routes } from "react-router-dom";

import Home from "./Pages/Home";

import Product from "./Pages/Product/Products";

import Login from "./Pages/Login/Login";

import SignUp from "./Pages/SignUp/SignUp";

import ForgetPassword from "./Pages/ForgetPassword";

import UpdatePassword from "./Pages/UpdatePassword";

import Checkout from "./Pages/CheckOut/Checkout";

import MyOrders from "./Pages/Orders/MyOrders";

import { useDispatch, useSelector } from "react-redux";

import { loadUserAction } from "./Redux/Actions/userAction";

import RestPassword from "./Pages/RestPassword";

import AdminHome from "./Pages/Admin/AdminHome";

import AddProduct from "./Pages/Admin/Product/AddProducts/AddProduct";

import AddCategory from "./Pages/Admin/Product/AddCategory/AddCategory";

import ViewProducts from "./Pages/Admin/Product/ViewsProducts/ViewProducts";

import OrderList from "./Pages/Admin/Orders/OrderList";

import UsersList from "./Pages/Admin/Users/UsersList";

import Family from "./Pages/Family";

import OrderDetails from "./Pages/Orders/OrderDetails";

import AllOrders from "./Pages/Orders/AllOrders";

import AddReviews from "./Pages/Reviews/AddReviews";

import AllReviews from "./Pages/Reviews/AllReviews";

import About from "./Pages/About/About";

import PageNotFound from "./Components/PageNotFound/PageNotFound";

import AdminProducts from "./Pages/Admin/Product/AdminProducts";

import UpdateProducts from "./Pages/Admin/Product/UpdateProducts/UpdateProducts";

import ViewCategory from "./Pages/Admin/Product/ViewCategory/ViewCategory";

import UpdateCategory from "./Pages/Admin/Product/UpdateCategory/UpdateCategory";

import AdminOrderDetails from "./Pages/Admin/Orders/OrderDetails";

import UpdateUser from "./Pages/Admin/Users/UpdateUser";

import ReviewsList from "./Pages/Admin/Reviews/ReviewsList";

const App = () => {

const { isAuthUser, user } = useSelector((state) => state.user);

const dispatch = useDispatch();

useEffect(() => {

dispatch(loadUserAction());

}, [dispatch]);

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/Signup" element={!isAuthUser ? <SignUp /> : <Home />} />

<Route path="/Login" element={!isAuthUser ? <Login /> : <Home />} />

<Route

path="/RestPassword"

element={isAuthUser ? <UpdatePassword /> : <Login />}

/>

<Route

path="/Forget/Password"

element={!isAuthUser ? <ForgetPassword /> : <Home />}

/>

<Route path="/products" element={<Product />} />

<Route path="/products/:keyword" element={<Product />} />

<Route path="/products/category/:categoryId" element={<Product />} />

<Route path="/Order/Checkout" element={<Checkout />} />

<Route

path="/Order/MyOrder"

element={isAuthUser ? <AllOrders /> : <Login />}

/>

<Route

path="/Order/OrderDetails/:orderId"

element={isAuthUser ? <OrderDetails /> : <Login />}

/>

<Route

path="/reset-password/:id/:token"

element={!isAuthUser ? <RestPassword /> : <Login />}

/>

<Route

path="/review/add"

element={isAuthUser ? <AddReviews /> : <Login />}

/>

{/\* family Route \*/}

<Route

path="/user/:userId"

element={isAuthUser ? <Family /> : <Login />} />

<Route path="/reviews/all" element={<AllReviews />} />

<Route path="/about" element={<About />} />

{user && user.role === "Admin" ? (

<>

<Route path="/admin/home" element={<AdminHome />} />

{/\* Product Route \*/}

<Route path="/admin/products" element={<AdminProducts />} />

<Route path="/admin/add/products" element={<AddProduct />} />

<Route path="/admin/view/products" element={<ViewProducts />} />

<Route

path="/admin/update/products/:productId"

element={<UpdateProducts />}

/>

<Route path="/admin/add/category" element={<AddCategory />} />

<Route path="/admin/view/category" element={<ViewCategory />} />

<Route

path="/admin/update/category/:categoryId"

element={<UpdateCategory />}

/>

{/\* Orders Route \*/}

<Route path="/admin/view/orders" element={<OrderList />} />

<Route

path="/admin/update/order/:orderId"

element={<AdminOrderDetails />}

/>

{/\* Users Route \*/}

<Route path="/admin/view/users" element={<UsersList />} />

<Route path="/admin/update/:userId" element={<UpdateUser />} />

{/\* Reviews Route \*/}

<Route path="/admin/view/reviews" element={<ReviewsList />} />

</>

) : (

""

)}

<Route path="\*" element={<PageNotFound />} />

</Routes>

</BrowserRouter>

);

};

export default App;

**Store.js**

**import { createStore, applyMiddleware, combineReducers } from "redux";**

**import thunk from "redux-thunk";**

**import { composeWithDevTools } from "@redux-devtools/extension";**

**import {**

**loginReducer,**

**registerReducer,**

**logOutUser,**

**getUserReducer,**

**userPasswordUpdateReducer,**

**passwordResetEmailSendReducer,**

**userPasswordResetReducer,**

**getAllUserAdminReducer,**

**deleteUserAdminReducer,**

**updateUserRoleReducer,**

**} from "./Reducers/userReducers";**

**import {**

**getAllProductsReducer,**

**addProductsReducer,**

**deleteProductReducer,**

**getSingleProductReducer,**

**updateProductReducer,**

**} from "./Reducers/productReducer";**

**import { myCartProductReducer } from "./Reducers/cartReducer";**

**import {**

**addCategoryReducer,**

**deleteCategoryReducer,**

**getCategoryReducer,**

**updateCategoryReducer,**

**} from "./Reducers/categoryReducer";**

**import {**

**getAllOrdersAdminReducer,**

**getUpdateOrderAdminReducer,**

**getUserOrderDetailsReducer,**

**getUserOrderReducer,**

**} from "./Reducers/ordersReducer";**

**import {**

**deleteReviewsReducer,**

**getAllAdminReviewsReducer,**

**getAllReviewsReducer,**

**} from "./Reducers/reviewsReducer";**

**const rootReducer = combineReducers({**

**//User Reducer**

**register: registerReducer,**

**user: loginReducer,**

**userdetails: getUserReducer,**

**logOut: logOutUser,**

**passwordUpdate: userPasswordUpdateReducer,**

**sendRestPassMail: passwordResetEmailSendReducer,**

**passwordReset: userPasswordResetReducer,**

**getAllOrders: getUserOrderReducer,**

**getOrderDetails: getUserOrderDetailsReducer,**

**getAllReviews: getAllReviewsReducer,**

**userCart: myCartProductReducer,**

**// Product Reducers:**

**addProduct: addProductsReducer,**

**getAllProducts: getAllProductsReducer,**

**getSingleProduct: getSingleProductReducer,**

**deleteProduct: deleteProductReducer,**

**updateProduct: updateProductReducer,**

**//Category Reducer**

**addCategory: addCategoryReducer,**

**getAllCategory: getCategoryReducer,**

**deleteCategory: deleteCategoryReducer,**

**//Admin Reducer**

**adminAllOrders: getAllOrdersAdminReducer,**

**adminUpdateOrder: getUpdateOrderAdminReducer,**

**adminAllUsers: getAllUserAdminReducer,**

**adminDeleteUser: deleteUserAdminReducer,**

**adminUpdateUser: updateUserRoleReducer,**

**adminUpdateCategory: updateCategoryReducer,**

**adminGetReviews: getAllAdminReviewsReducer,**

**deleteReview: deleteReviewsReducer,**

**});**

**const initialState = {**

**userCart: {**

**cartItems: localStorage.getItem("userCart")**

**? JSON.parse(localStorage.getItem("userCart"))**

**: [],**

**},**

**};**

**const middleware = [thunk];**

**const store = createStore(**

**rootReducer,**

**initialState,**

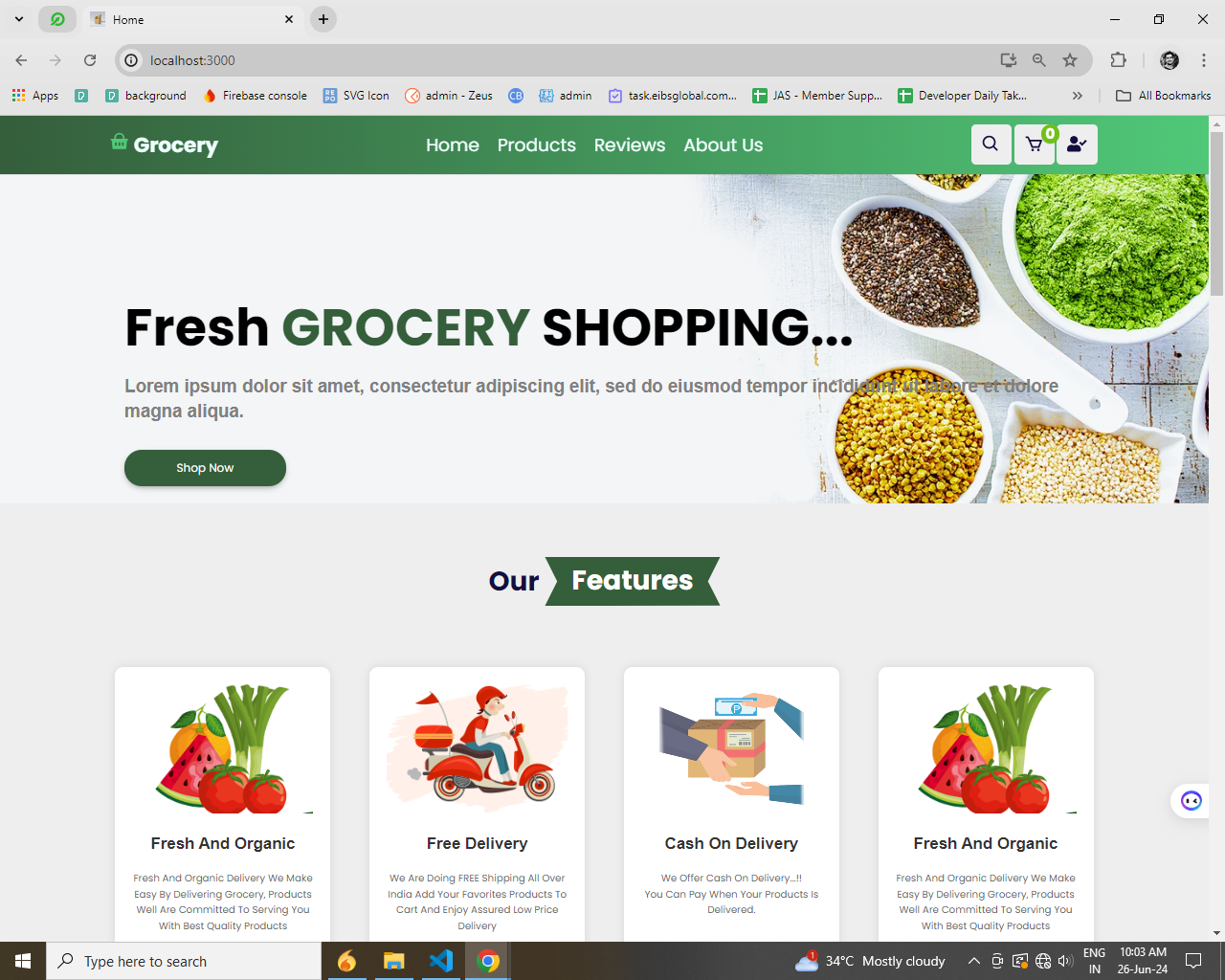
**composeWithDevTools(applyMiddleware(...middleware))**

**);**

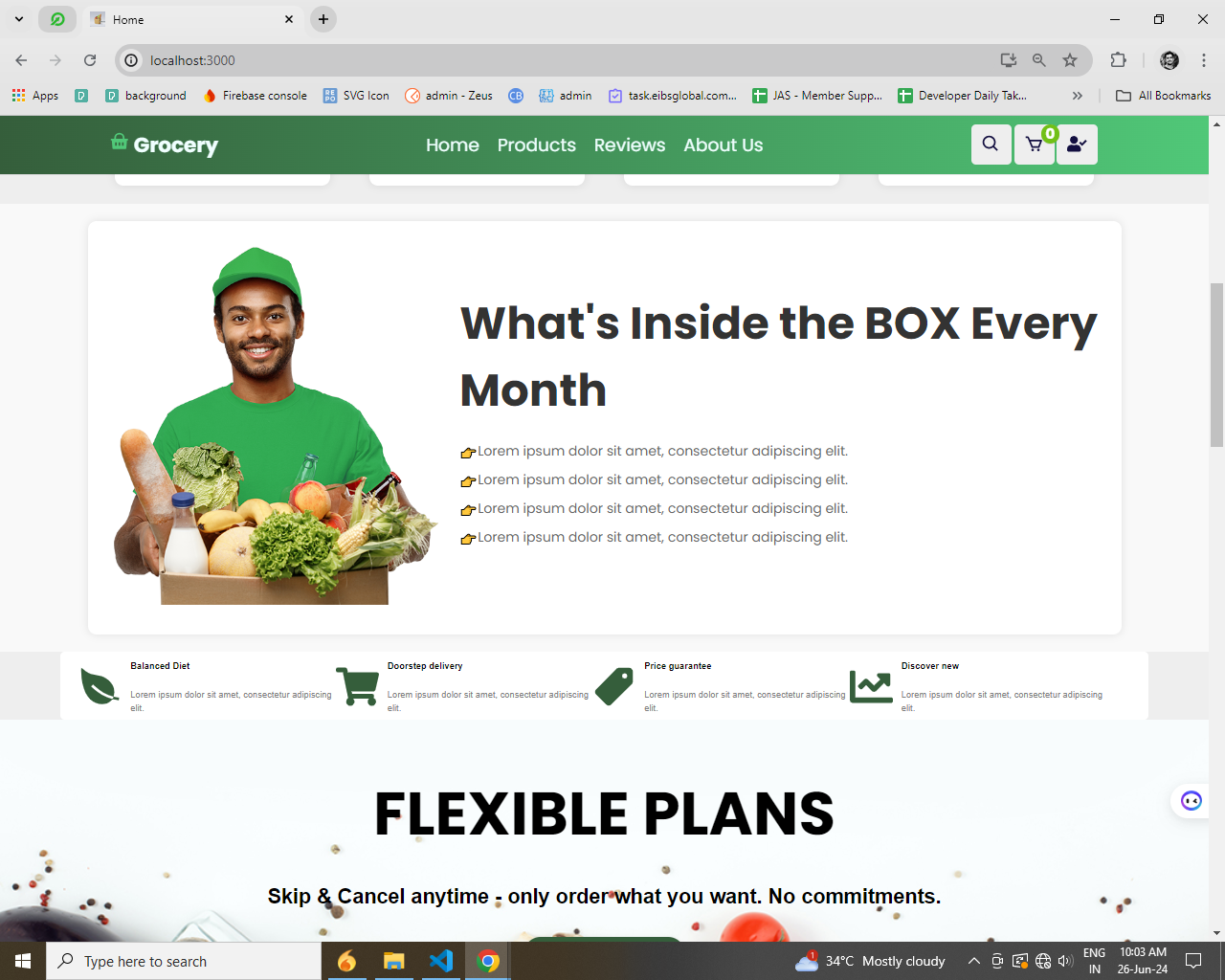
**export default store;**

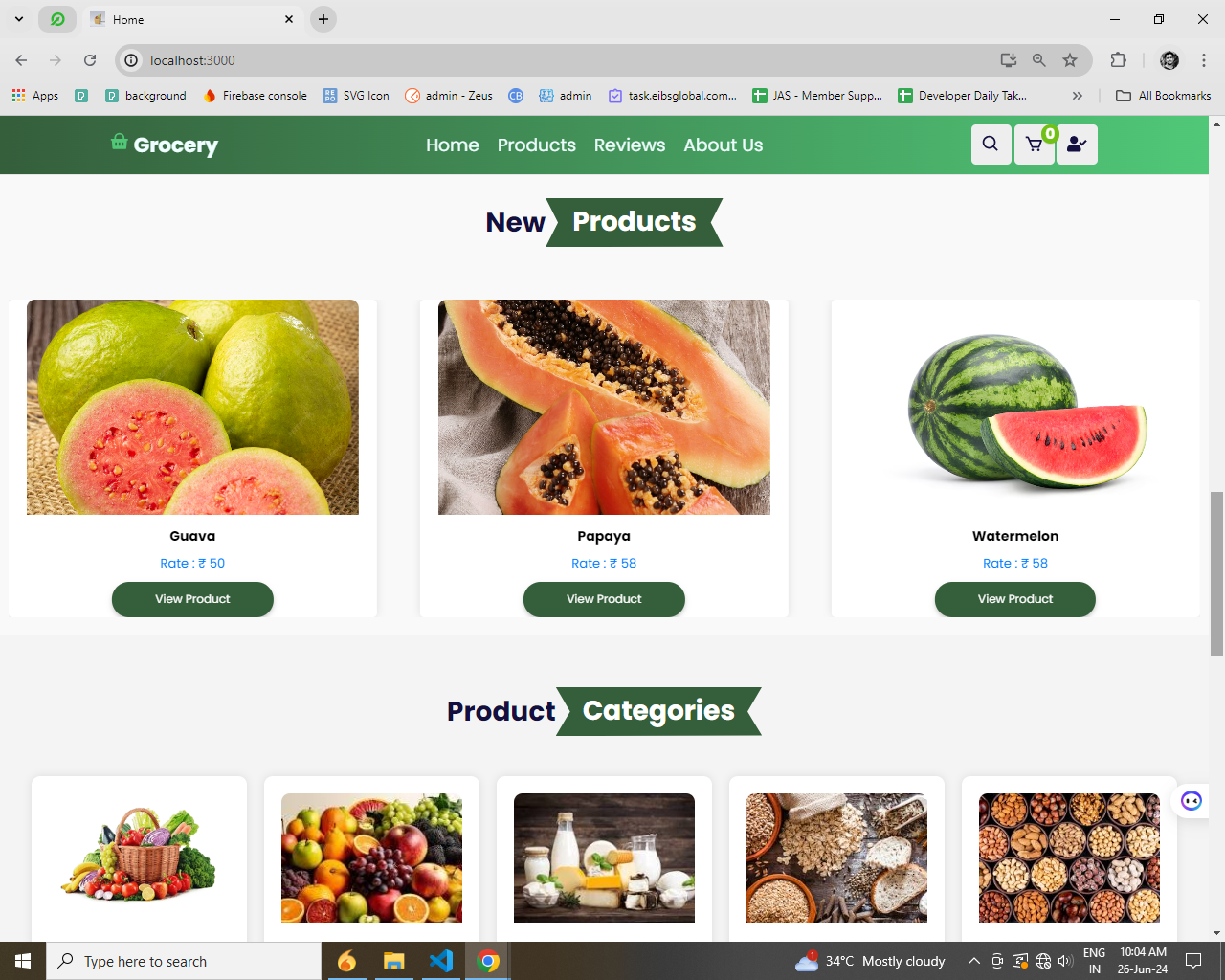
**SCREENSHOT**

**HomePage:**

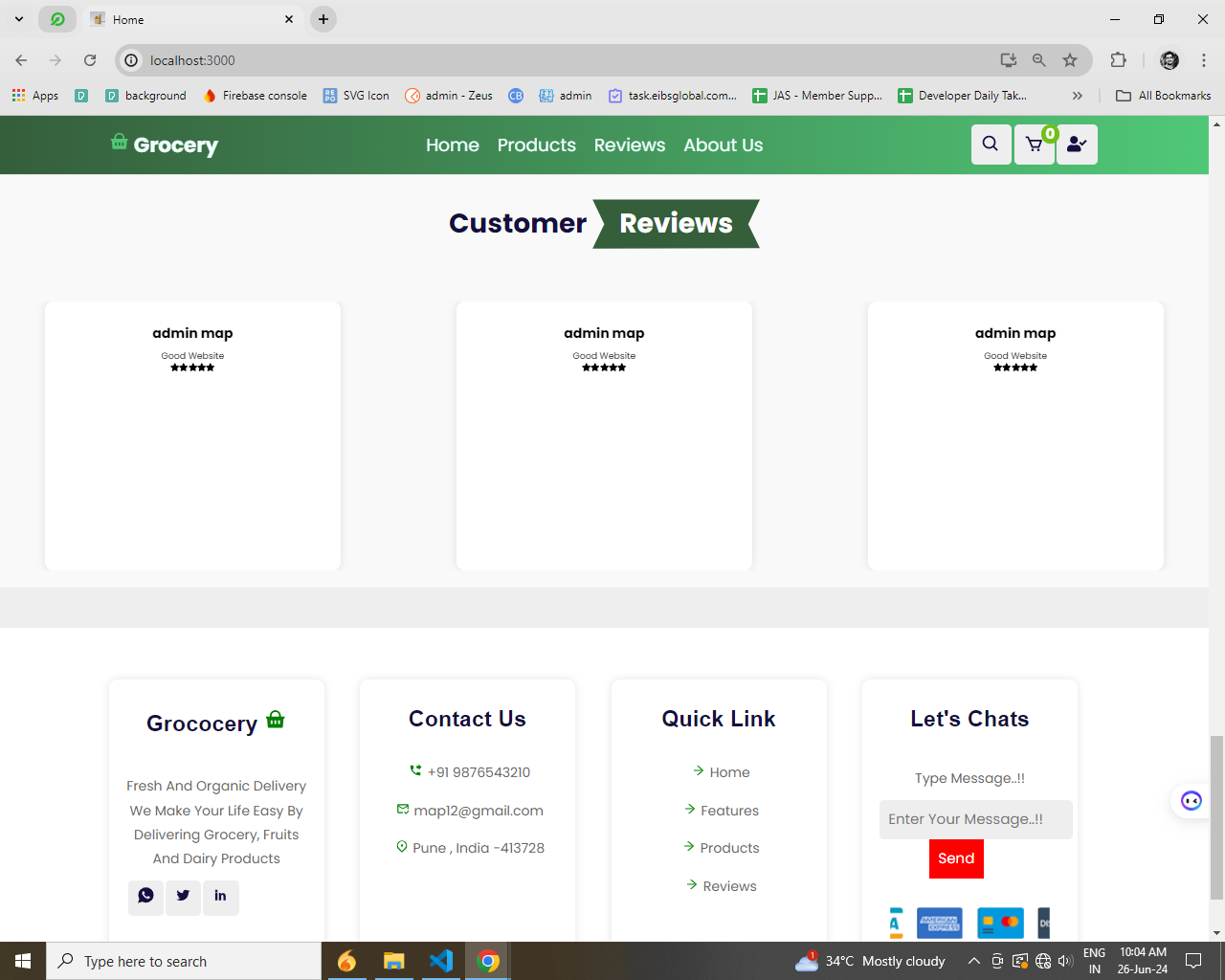
****

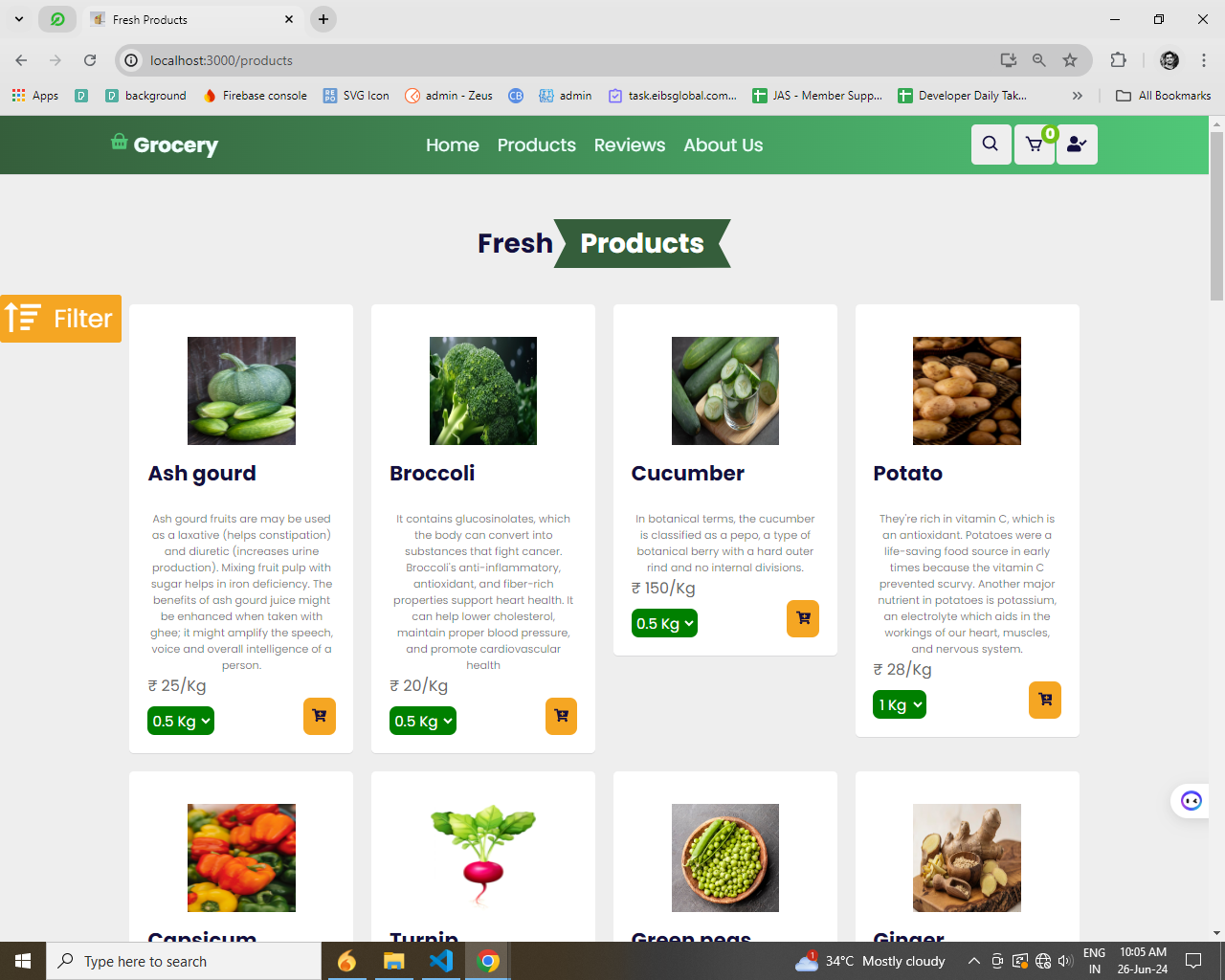
**Middle page**

****

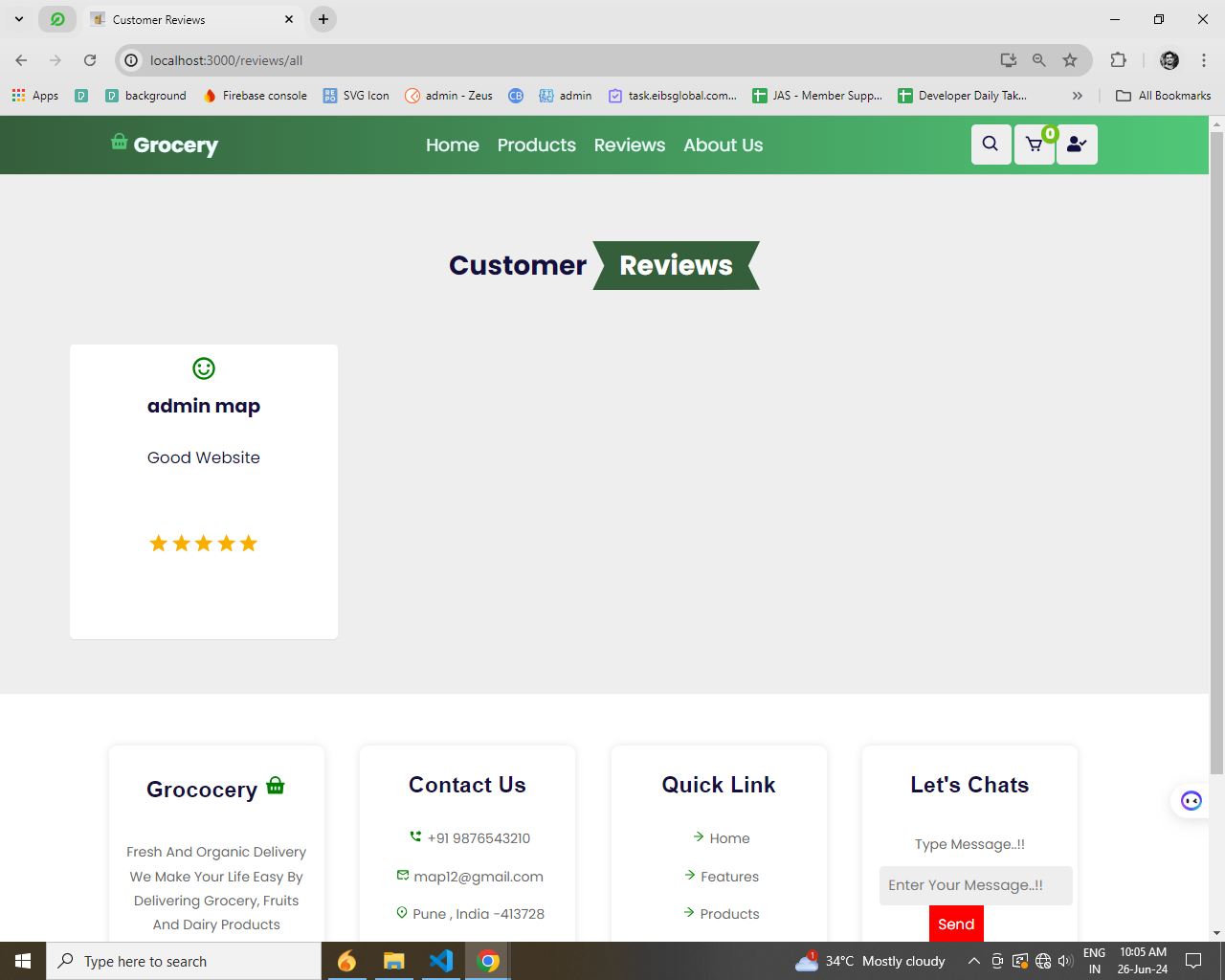
****

**Footer**

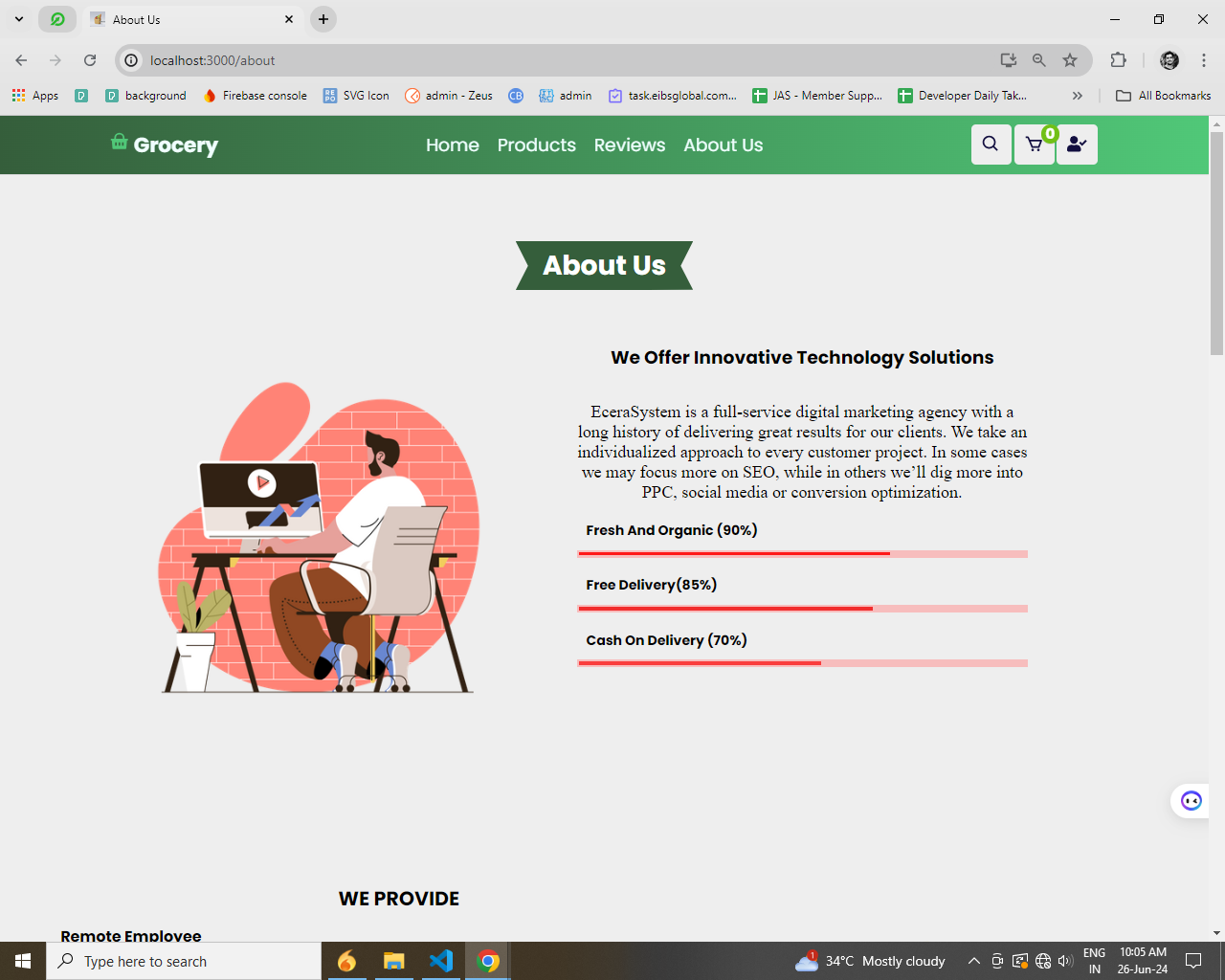
****

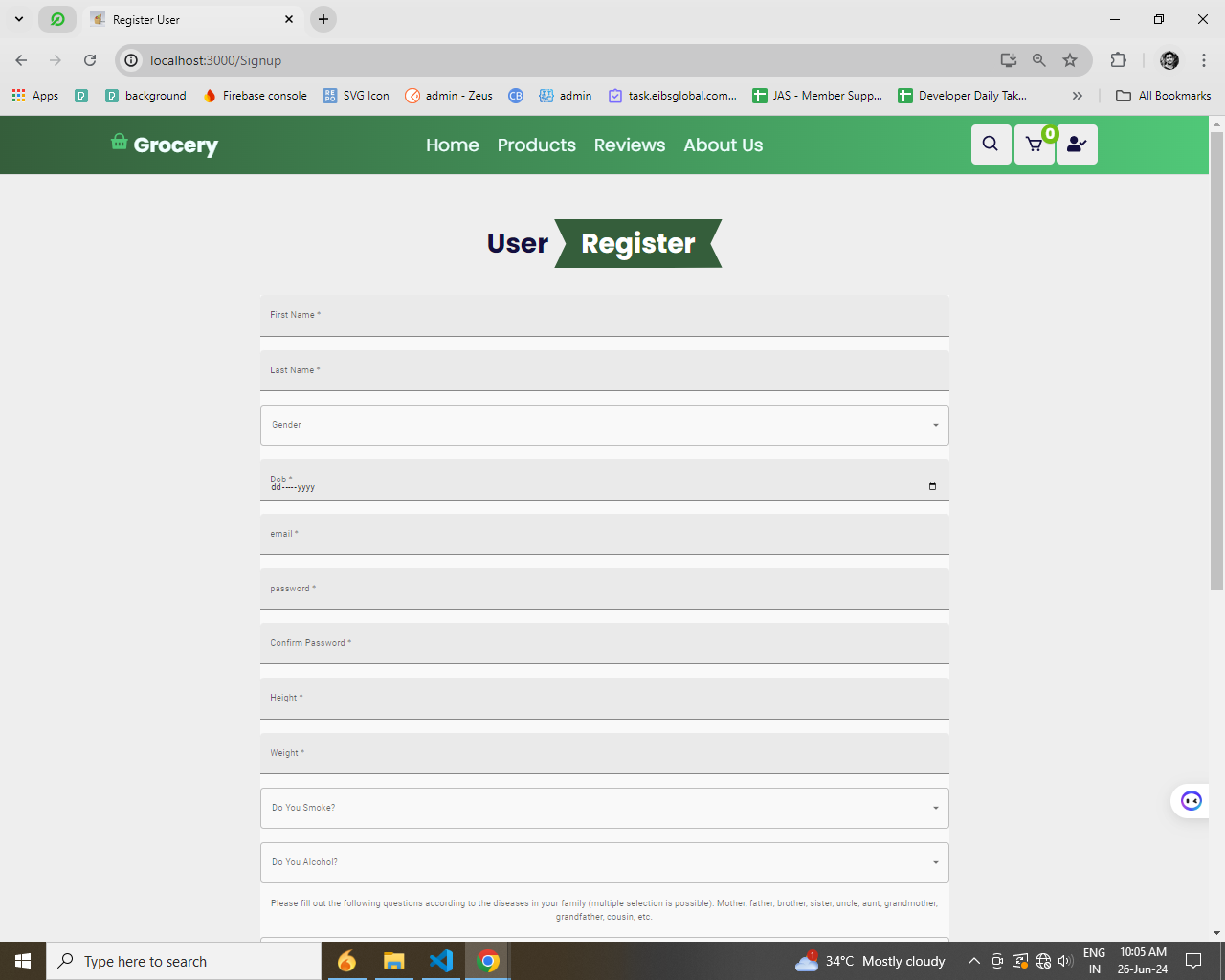
**Product **

**Review Page:**

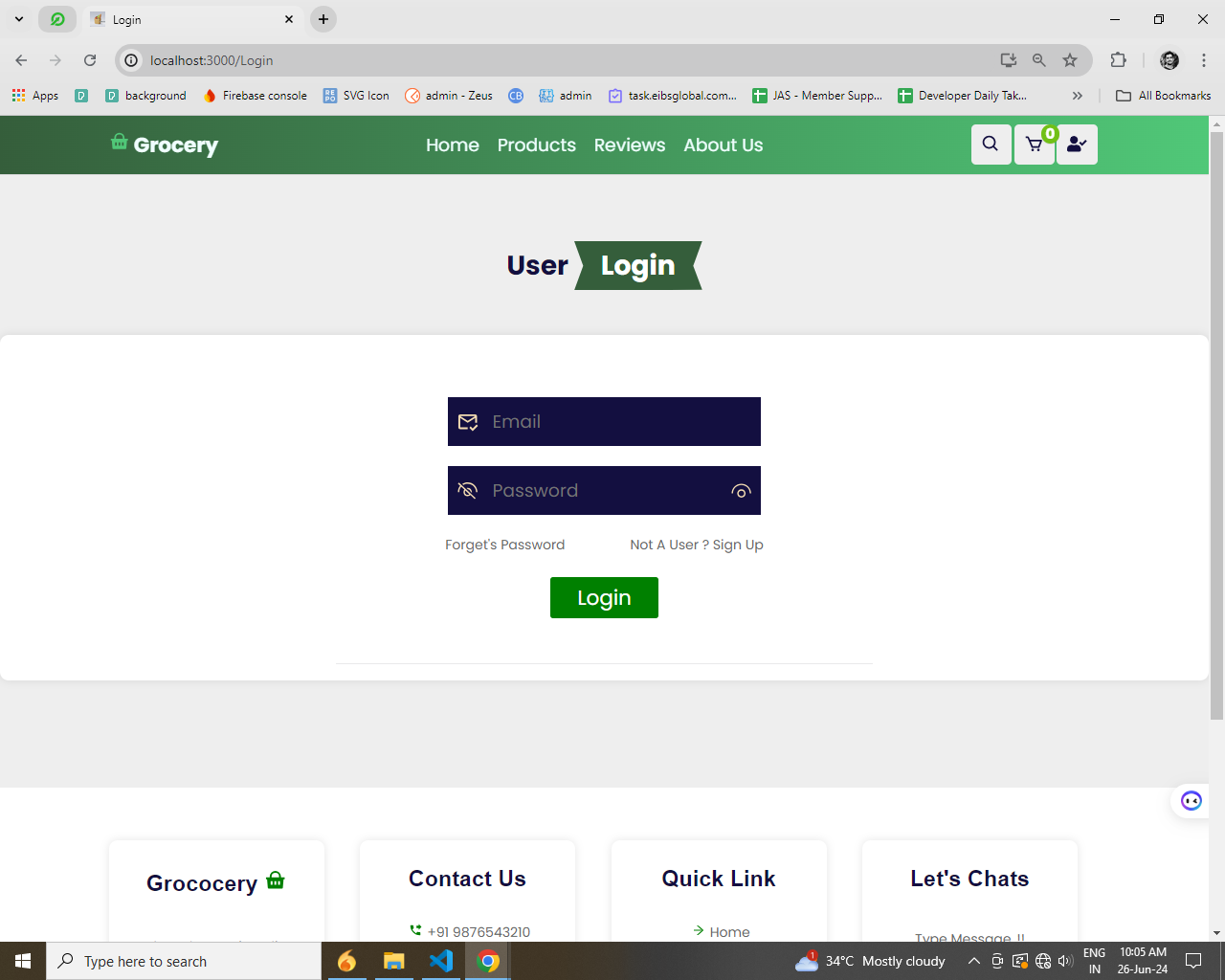
****

**About us**

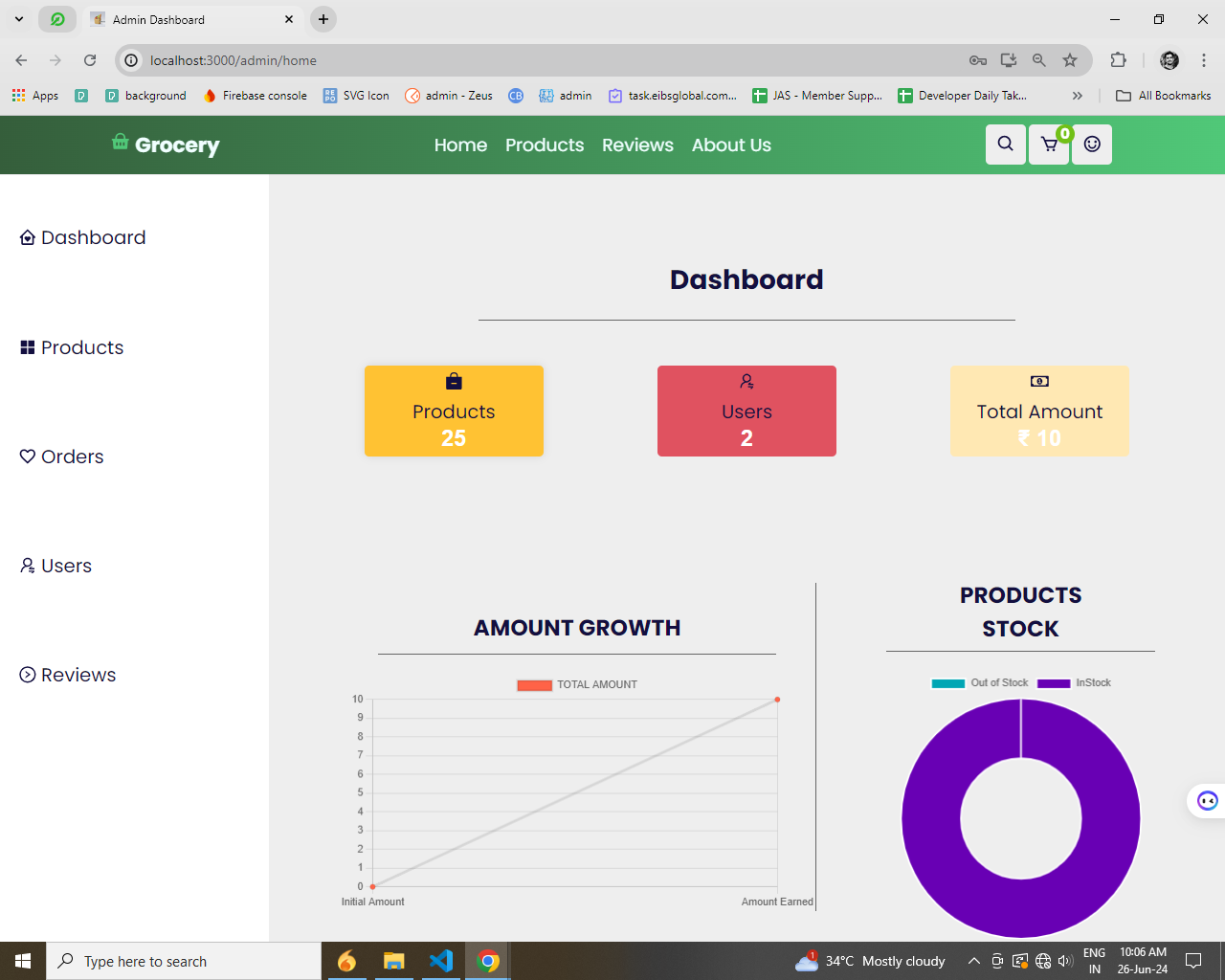
****

**Register**

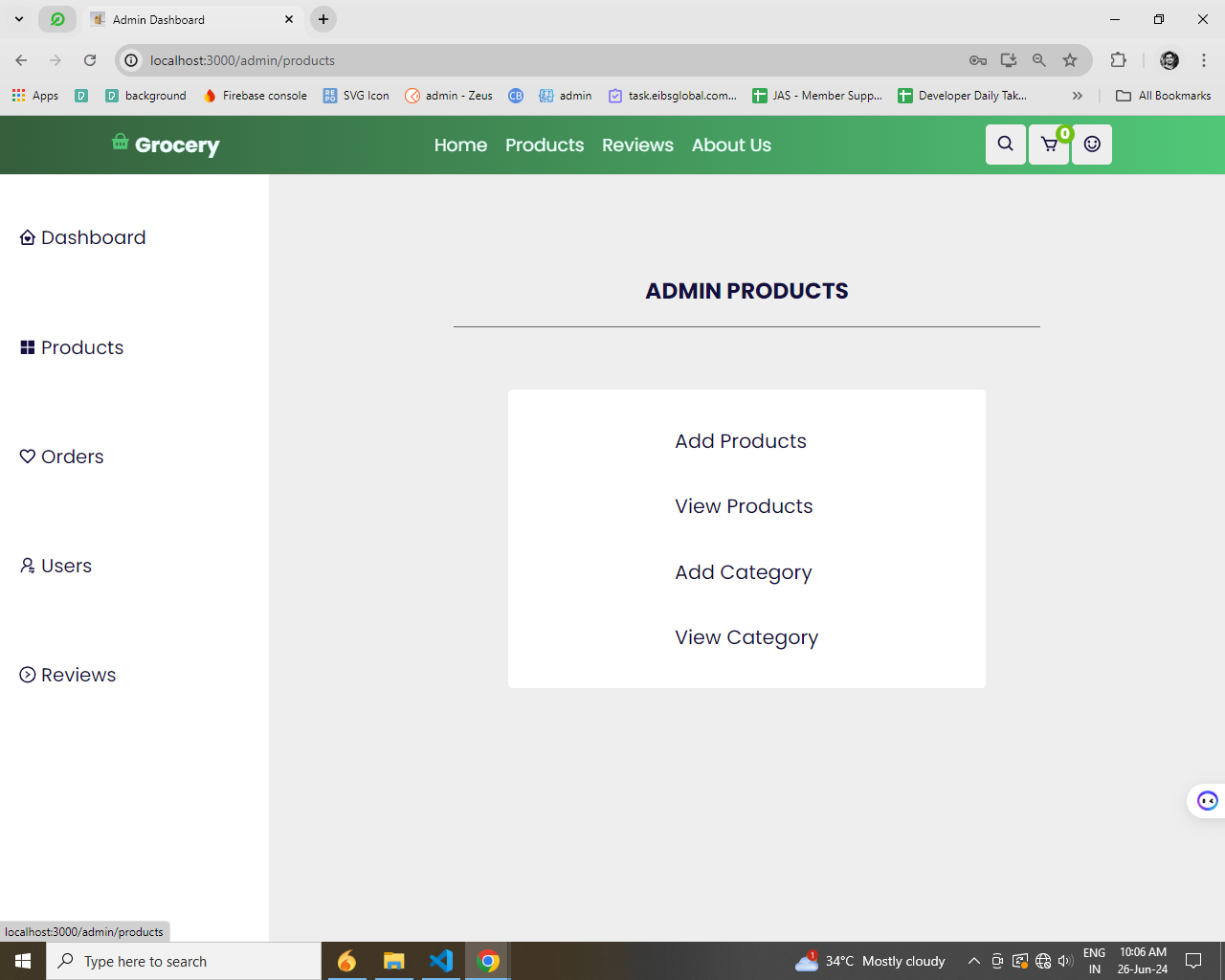
**Login**

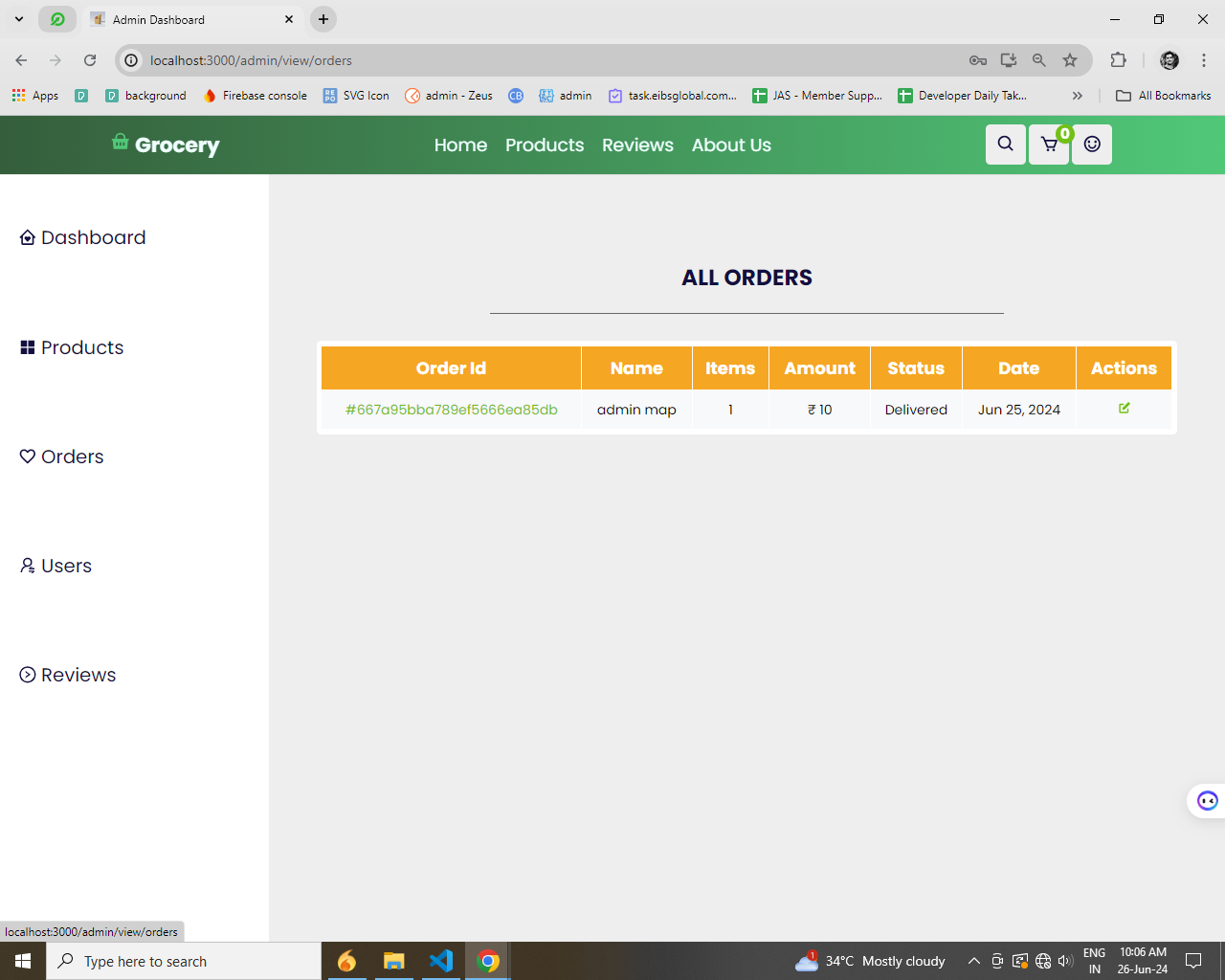
****

**Admin Dashboard**

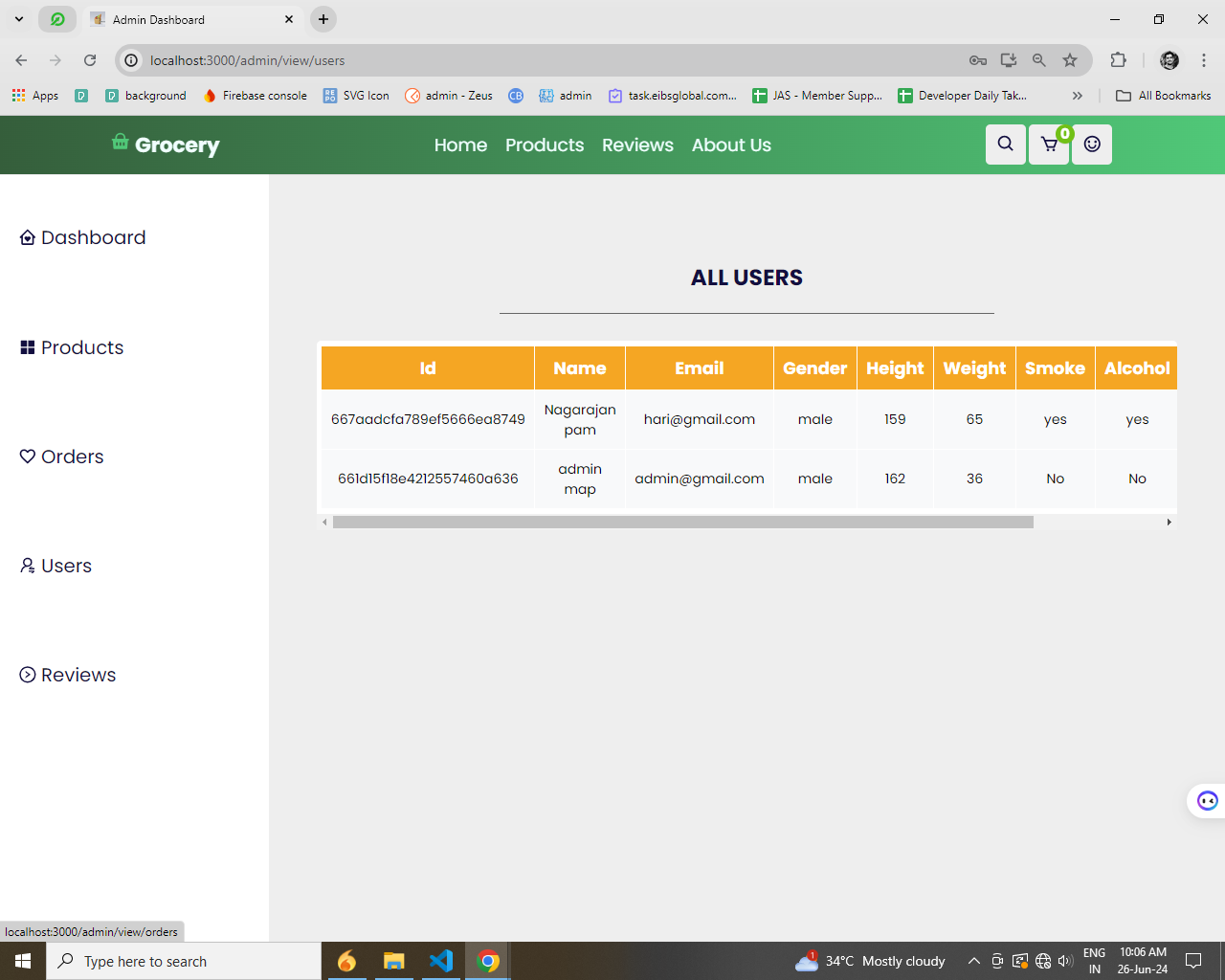
****

**Product dashboard:**

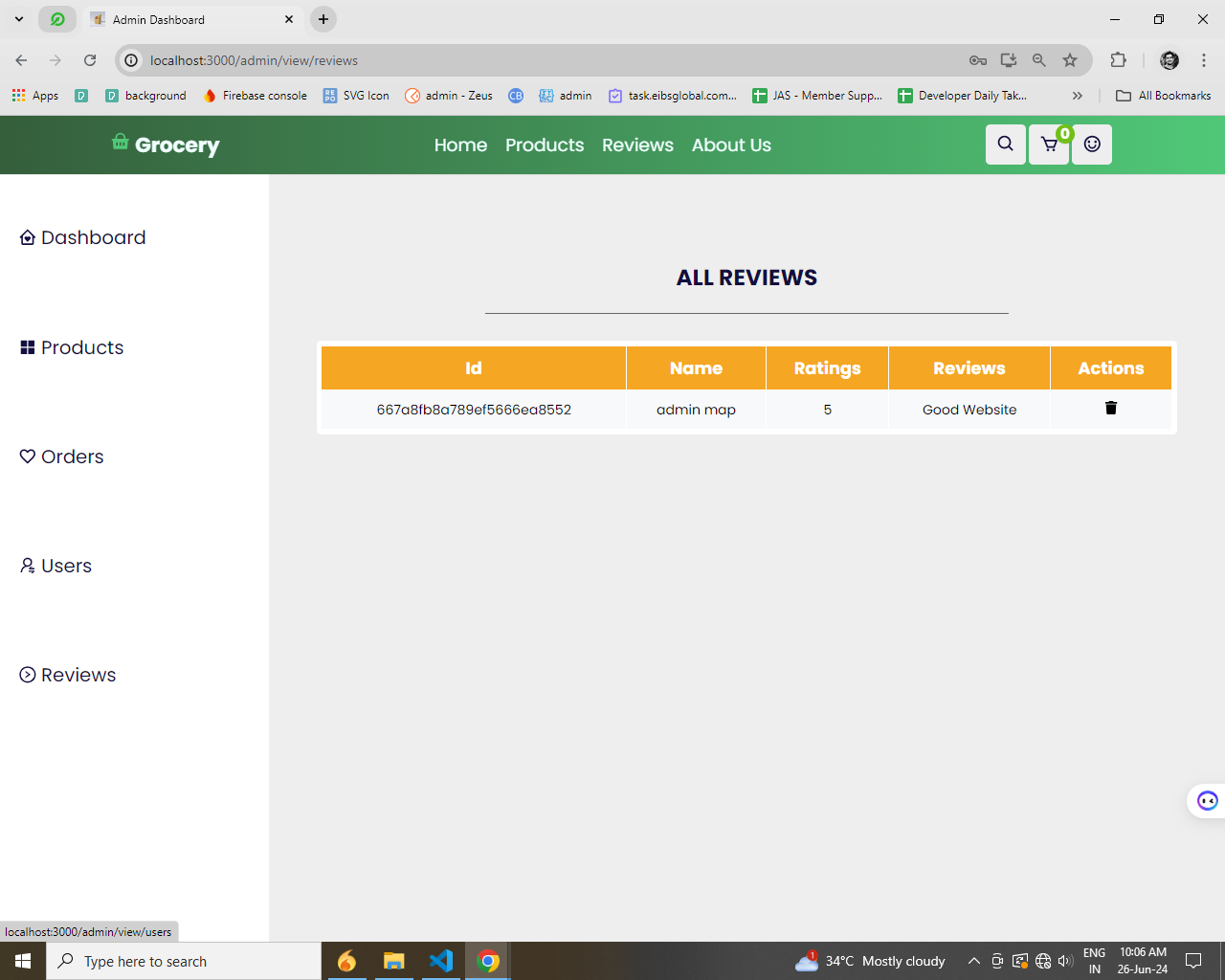
****

**Order List **

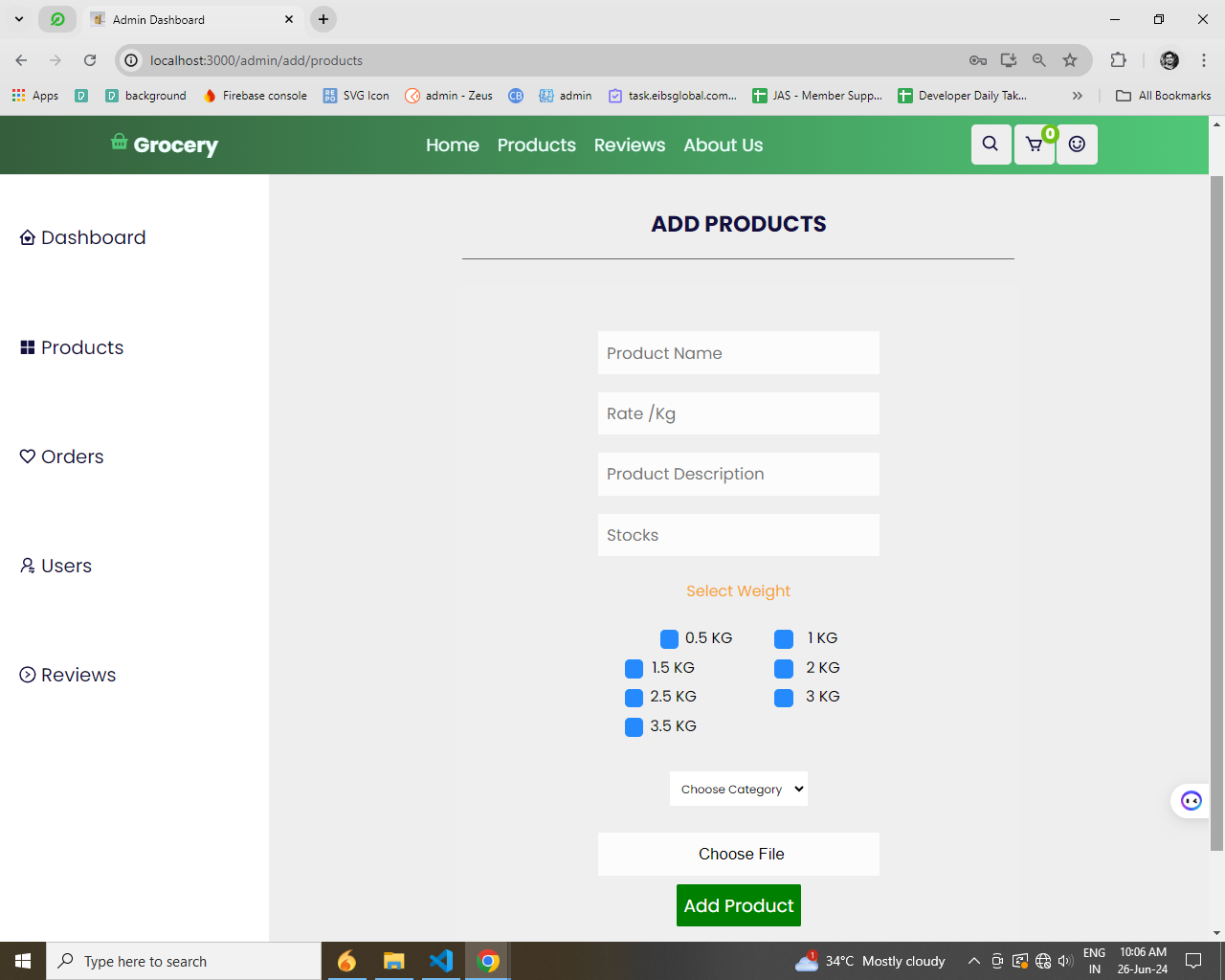
**User list**

****

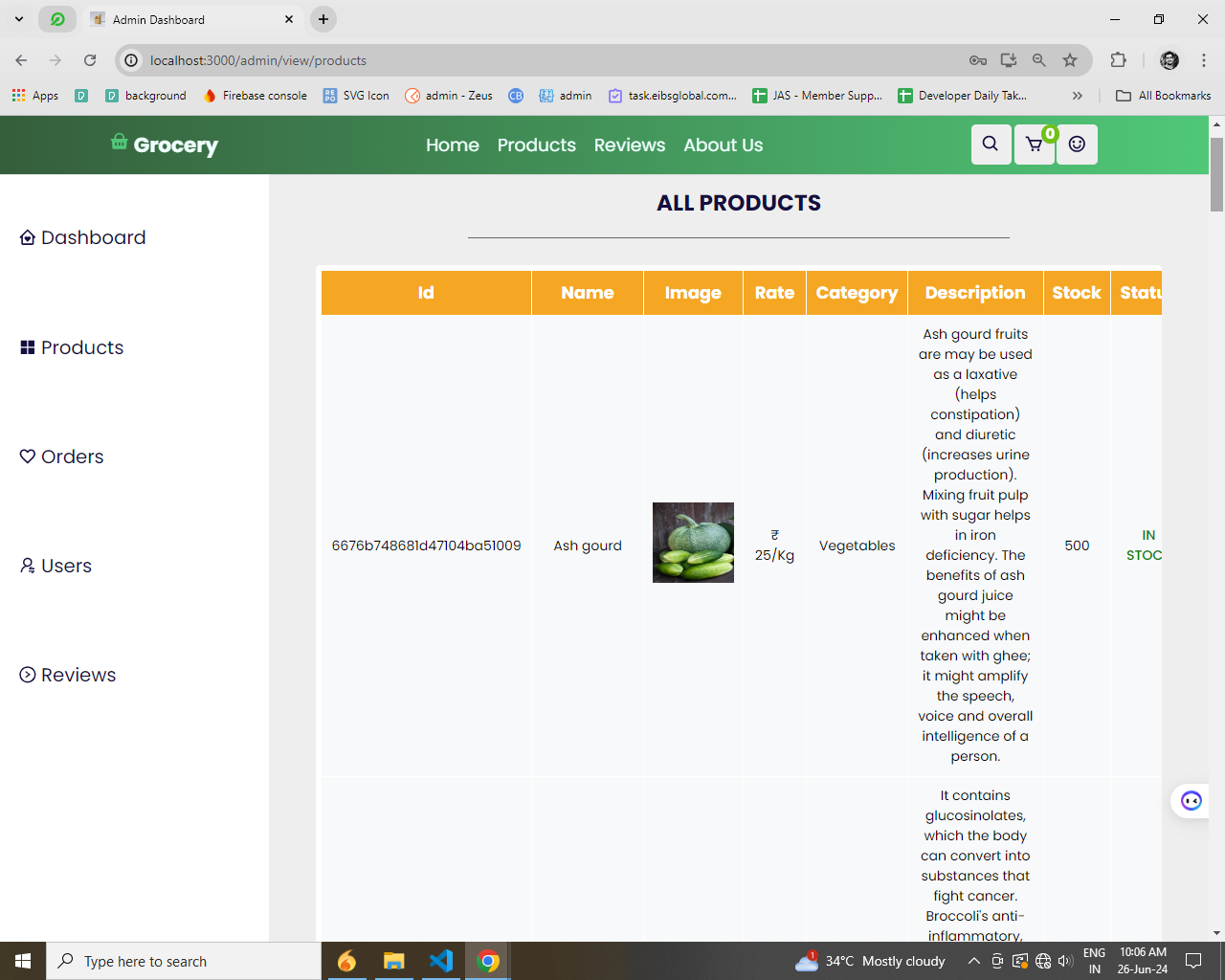
**All Reviews:**

****

**Add product:**

****

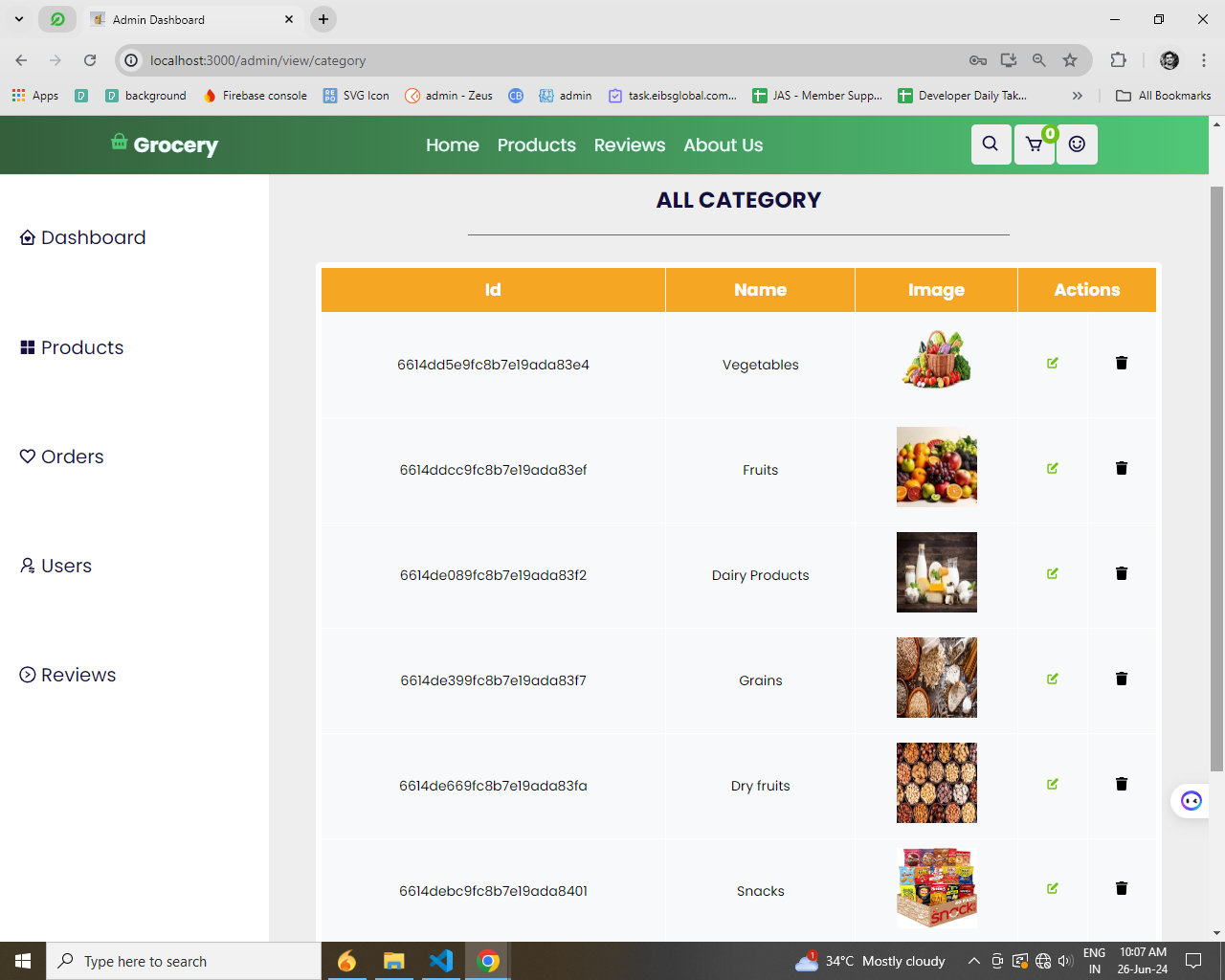
**All Product**

****

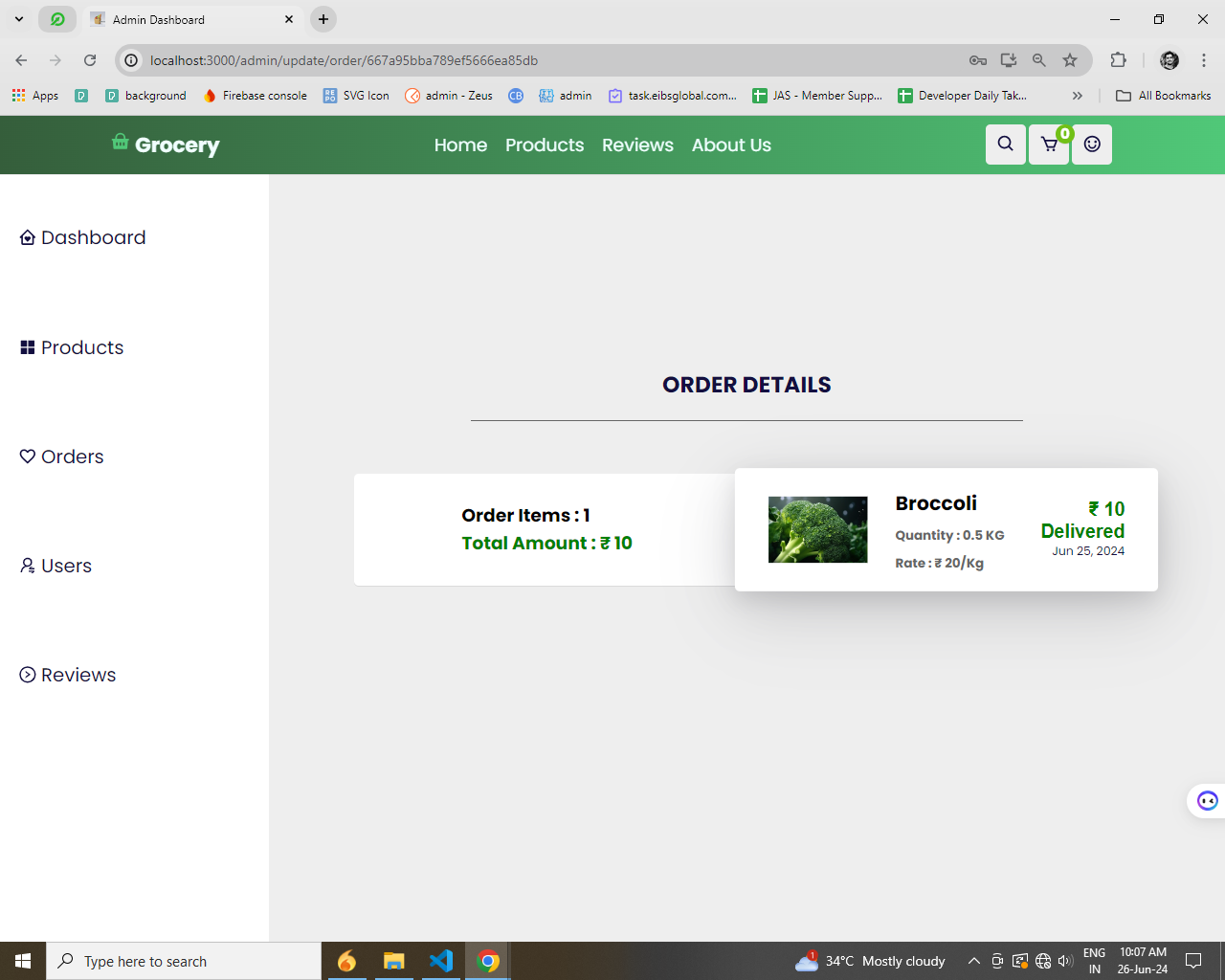
**Add category**

****

**All category**

****

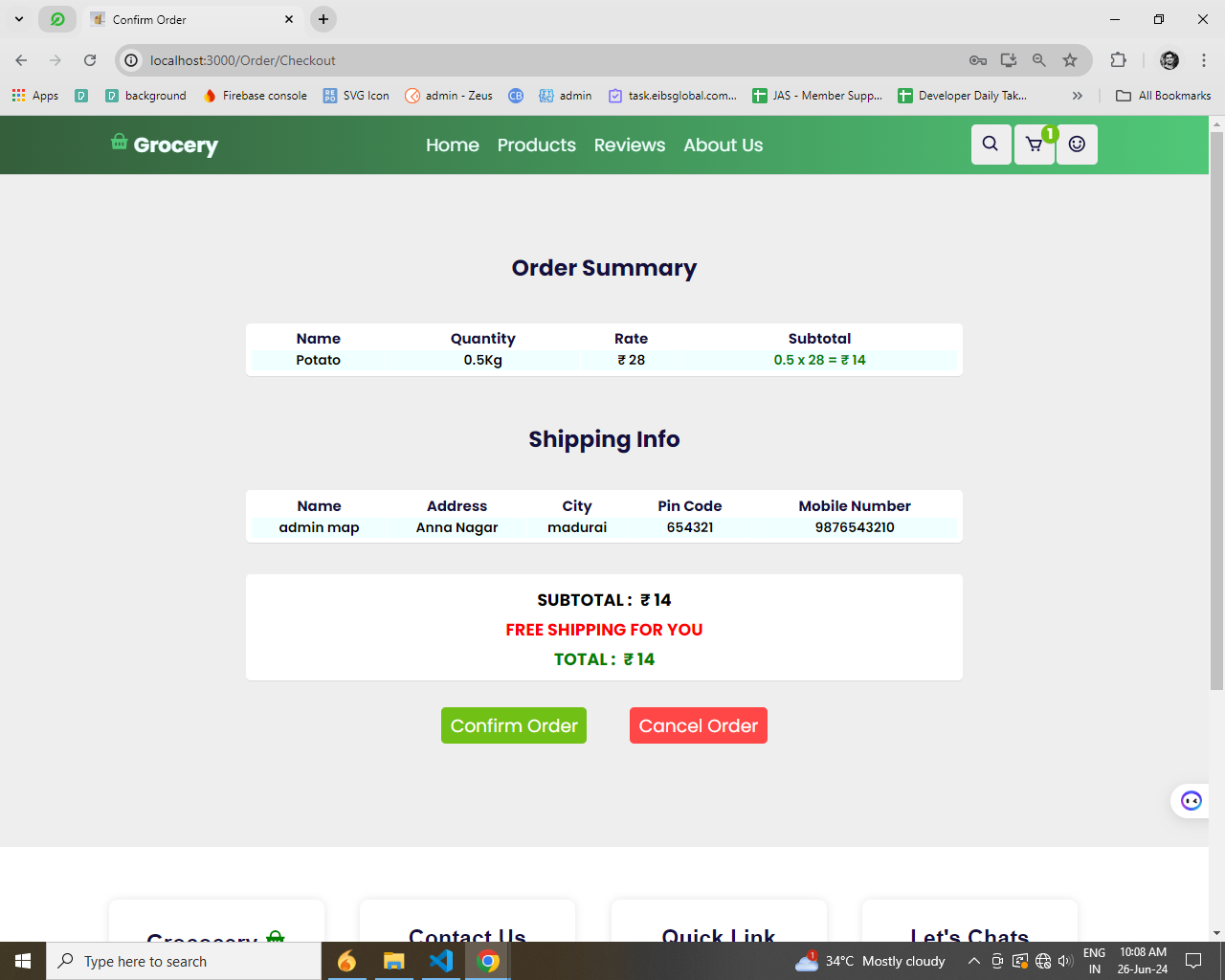
**Order details**

****

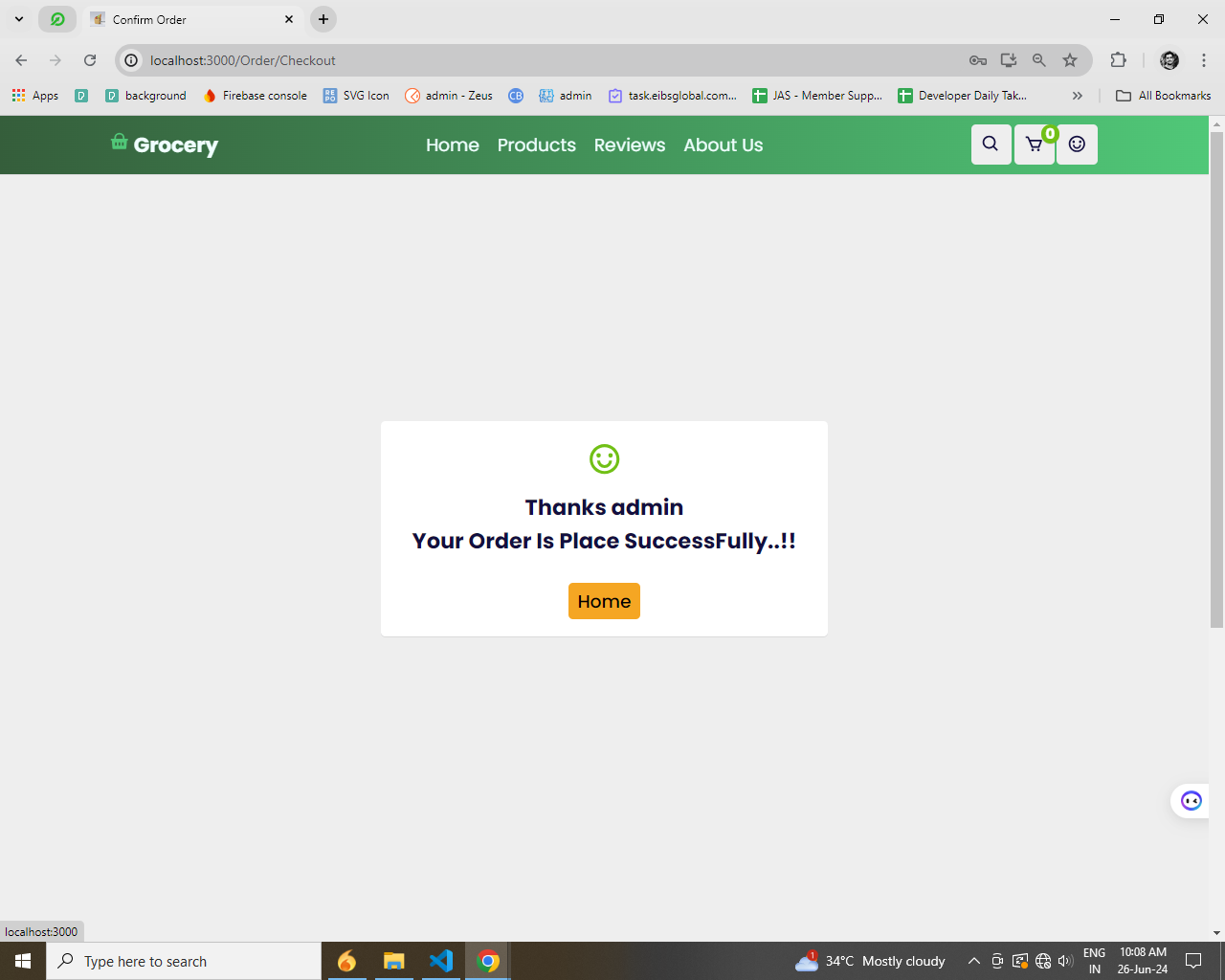
**Shipping details:**

****

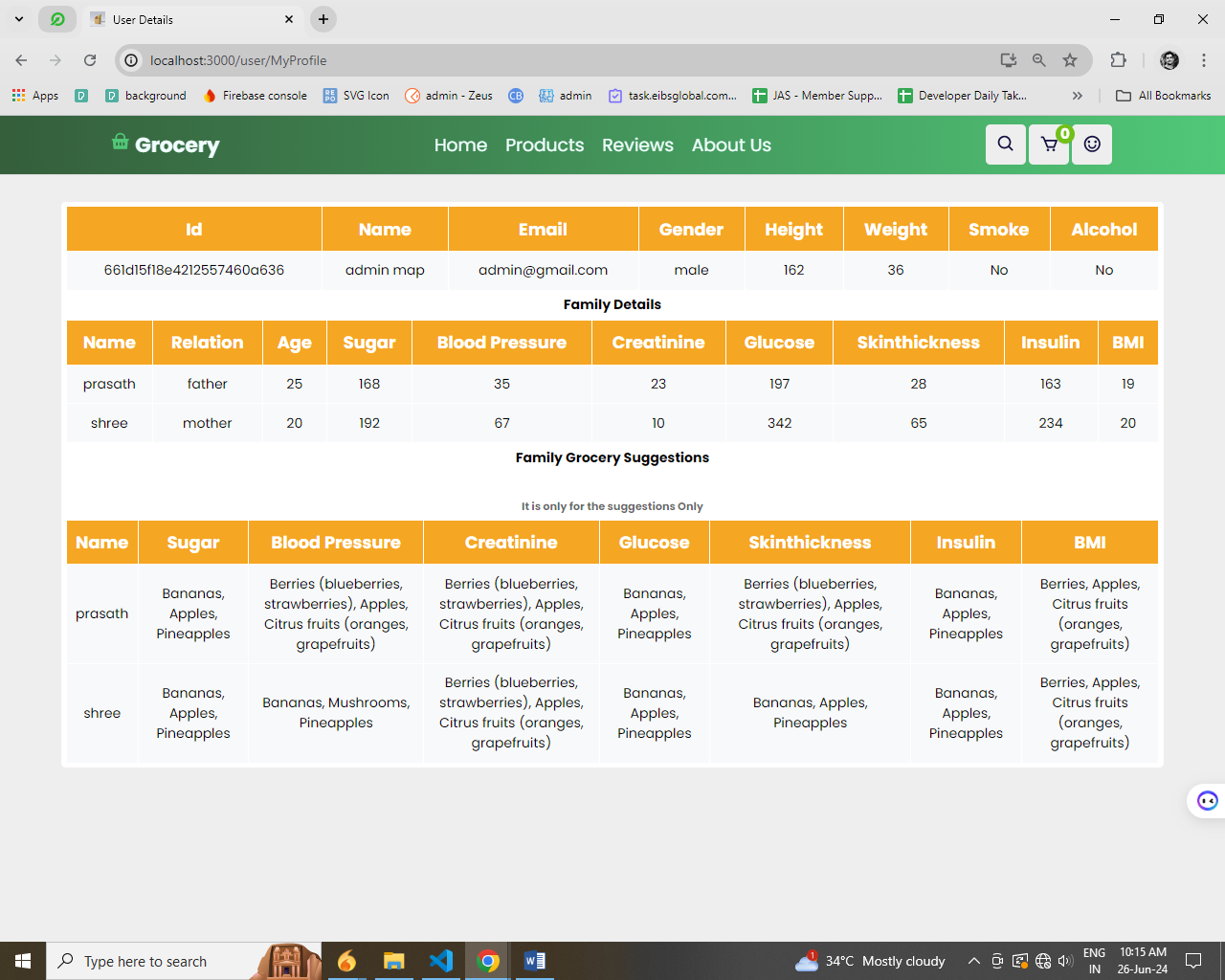
**Checkout**

****

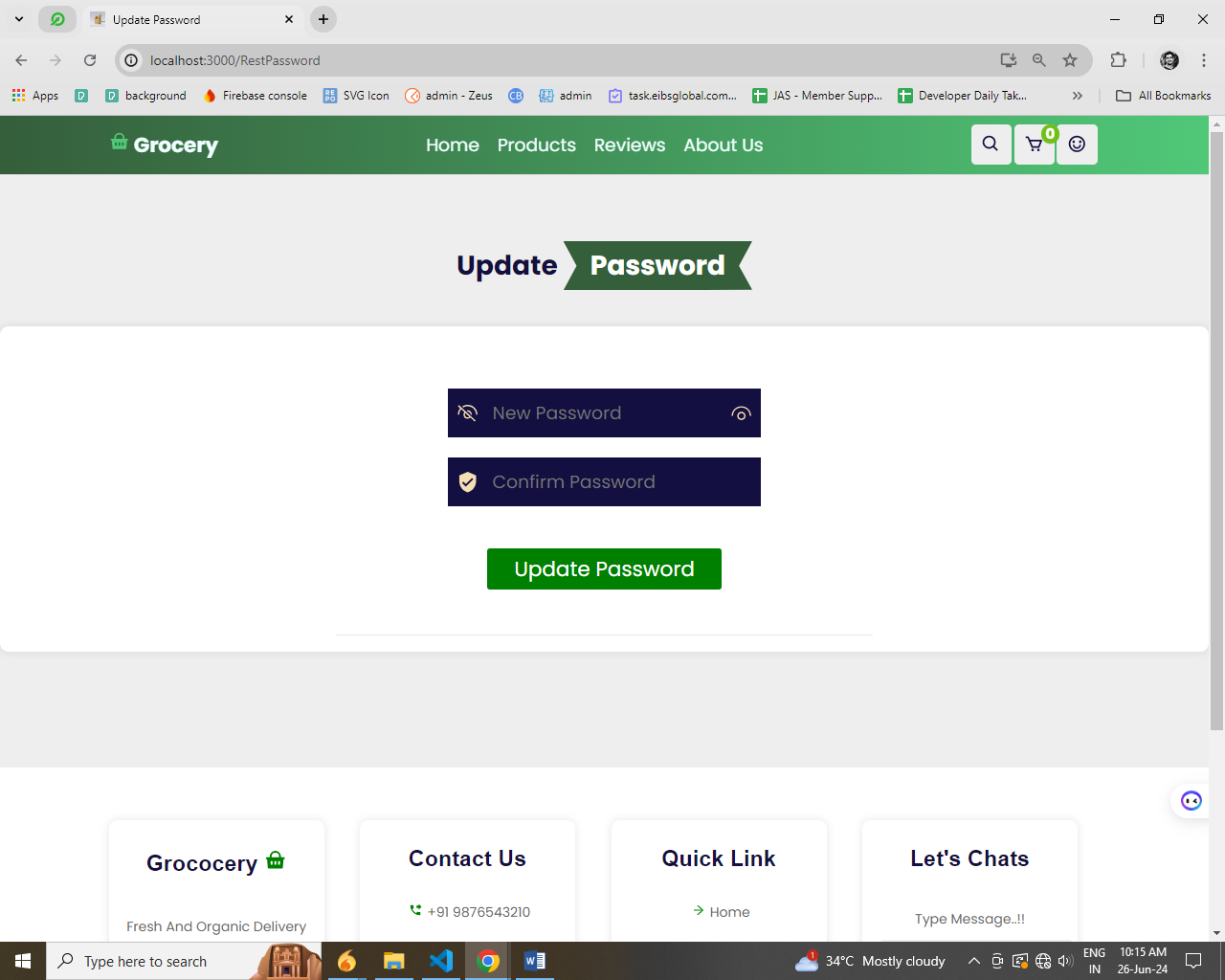
**Order successfully**

****

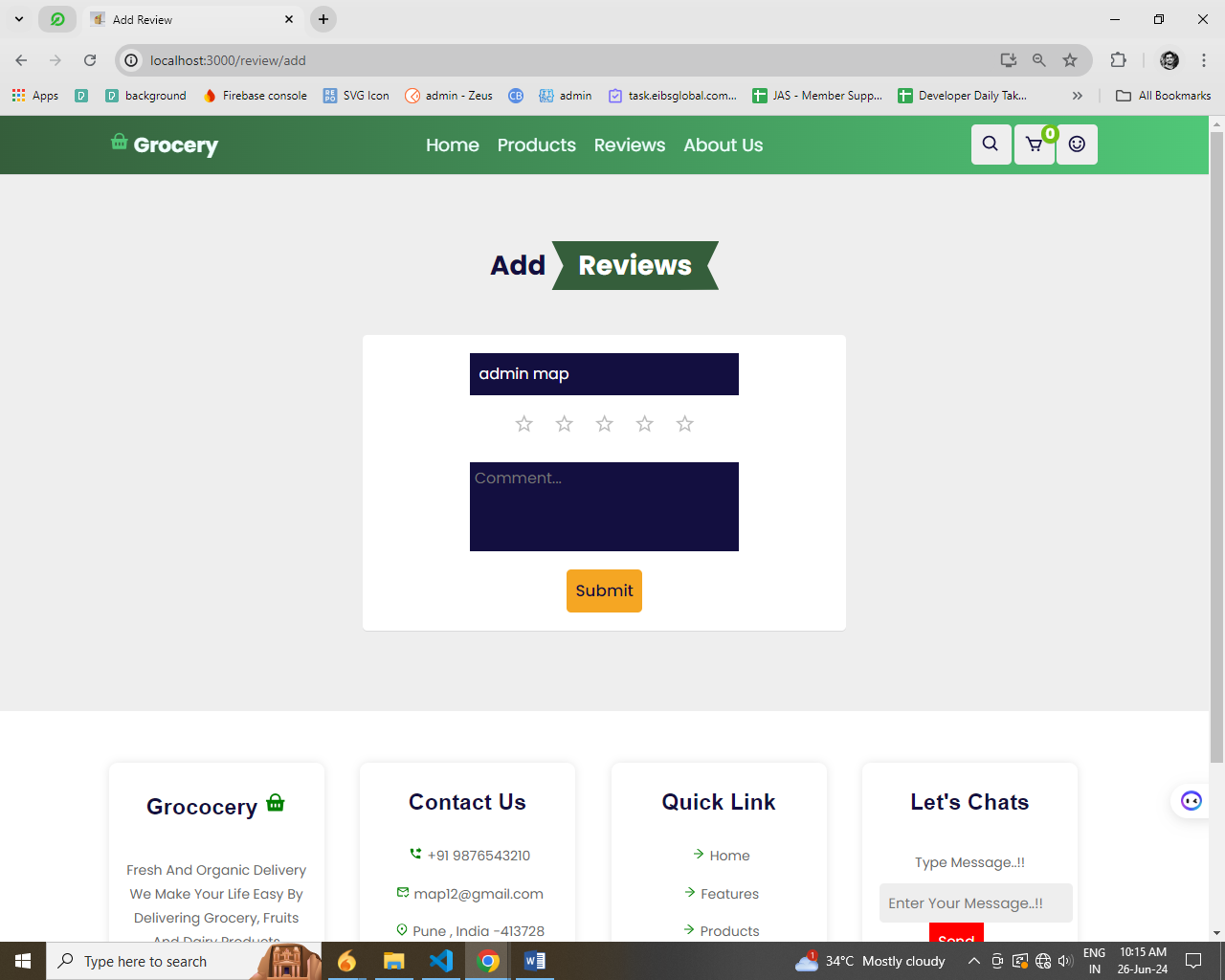
**Health Suggestions**

****

**Update the password:**

****

**Add review**

****

**CHAPTER-9**

**REFERENCE**

* https://nodejs.org/
* https://react.dev/
* https://www.npmjs.com/
* https://www.w3schools.com/nodejs/
* https://www.w3schools.com/reactjs/
* https://expressjs.com/
* https://www.mongodb.com/
* https://www.javatpoint.com/mern-stack