E-commerce website

A Project Report for Industrial Training and Internship

submitted by

Sreyoshee Saha

In the partial fulfillment of the award of the degree of

B.TECH

In the department of

Electronics and Communication Engineering
OF
NARULA INSTITUE OF TECHOLOGY

At

Ardent Computech Pvt. Ltd.

CERTIFICATE FROM SUPERVISOR

This is to certify that"Sreyoshee Saha,430223020128"have completed the project titled "E-Commerce Website" under my supervision during the period from "14/07/2025" to "21/07/2025" which is in partial fulfillment of requirements for the award of the B.Tech degree and submitted to the Department of "Electronics and Communication Engineering" of "Narula Institute of Technology".

Soumavo Acharjee

Signature of the Supervisor

Date(DD/MM/YYYY):
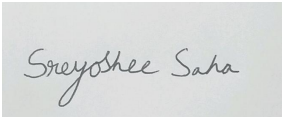
Name of the Project Supervisor: Soumavo Acharjee

BONA FIDE CERTIFICATE

Certified that this project work was carried out under my supervision

E-commerce website is a bonafide work of

Name of the student: Sreyoshee Saha   *Signature:* _Sreyoshee Saha_

SIGNATURE : _Sreyoshee Saha_

Name : Sreyoshee Saha

PROJECT MENTOR:

SIGNATURE : **Soumavo Acharjee**

Name: **Soumavo Acharjee**

EXAMINERS: **Soumavo Acharjee**

Ardent Original Seal

ACKNOWLEDGEMENT

The achievement that is associated with the successful completion of any task would be incomplete without mentioning the names of those people whose endless cooperation made it possible. Their constant guidance and encouragement made all our efforts successful.

We take this opportunity to express our deep gratitude towards our project mentor, *Soumavo Acharjee* , for giving such valuable suggestions, guidance and encouragement during the development of this project work.

Last but not the least, we are grateful to all the faculty members of **Ardent Computech Pvt. Ltd.** for their support.

1. <u>COMPANY PROFILE</u>

ARDENT (Ardent Computech Pvt. Ltd.), formerly known as Ardent Computech Private Limited, is an **ISO 9001:2015** certified Software Development and Training Company based in India. Operating independently since 2003, the organisation has recently undergone a strategic merger with ARDENT Technologies, enhancing its global outreach and service offerings.

ARDENT Technologies

ARDENT Technologies delivers high-end IT services across the UK, USA, Canada, and India. Its core competencies lie in the development of customised application software, encompassing end-to-end solutions including system analysis, design, development, implementation, and training. The company also provides expert consultancy and electronic security solutions. Its clientele spans educational institutions, entertainment companies, resorts, theme parks, the service industry, telecom operators, media, and diverse business sectors.

ARDENT Collaborations

ARDENT Collaborations, the Research, Training, and Development division of ARDENT (Ardent Computech Pvt. Ltd.), offers professional IT-enabled services and industrial training programs. These are tailored for freshers and professionals from B.Tech, M.Tech, MBA, MCA, BCA, and MSc backgrounds. ARDENT (Ardent Computech Pvt. Ltd.) provides Summer Training, Winter Training, and Industrial Training to eligible candidates. High-performing students may qualify for stipends, scholarships, and additional benefits based on performance and mentor recommendations.

Associations and Accreditations

ARDENT (Ardent Computech Pvt. Ltd.) is affiliated with the National Council of Vocational Training (NCVT) under the Directorate General of Employment & Training (DGET), Ministry of Labour & Employment, Government of India. The institution upholds strict quality standards under ISO 9001:2015 certification and is dedicated to bridging the gap between academic knowledge and industry skills through innovative training programs.

<u>INDIX</u>

2. Introduction

In today's technology-driven world, e-commerce platforms are essential for modern retail operations. This project presents a full-stack, role-based web application designed to manage online product sales and customer interactions. Built with Angular for the frontend, Node.js with Express for the backend, and MongoDB as the database, the platform facilitates seamless interactions between customers and administrators. Key features include product listings, user authentication, cart operations, order tracking, and admin controls for managing products and user activities. The platform is built for responsiveness, modularity, and scalability, supporting real-time transactions and role-specific dashboards.

The project bridges the gap between physical retail constraints and the growing demand for digital access. It offers a secure, mobile-friendly, and intuitive user experience that aligns with modern shopping trends. From dynamic product catalogs to secure checkout and admin-controlled stock management, the system is designed to mimic real-world e-commerce workflows while improving upon traditional inefficiencies.

Furthermore, this platform supports role-based user access, ensuring that only authorized users can perform sensitive operations. Users can track their order history and manage account details, while admins are provided with powerful tools to monitor, update, and fulfill customer needs. The RESTful API structure enables seamless integration with potential third-party services such as payment gateways or shipment tracking systems.

By adhering to best practices in software engineering, this project promotes scalability, security, and ease of maintenance. It follows modular architecture with clean separation of concerns between frontend, backend, and database layers, making it ideal for further feature extension and deployment in real-world scenarios.

2A. Objective

The primary goals of the e-commerce platform are:

- **Enable Easy Shopping:** The platform is built with a focus on providing an intuitive and seamless shopping experience. Users can easily navigate through product categories, apply dynamic filters such as price range and product ratings, and conduct keyword-based searches to find the products they need quickly. The responsive layout ensures usability across devices, enhancing accessibility.
- **Simplify Checkout:** To ensure a smooth purchasing process, the system supports real-time cart updates, detailed order previews, and a straightforward checkout mechanism. Users can add or remove items from the cart, view totals, and proceed to place an order with minimal steps. Once an order is confirmed, the backend handles processing, and the user receives updates regarding order status.
- **Support Admin Control:** The platform equips administrators with a powerful dashboard to manage all business operations effectively. Admins can perform CRUD (Create, Read, Update, Delete) operations on product listings, handle customer queries, monitor order logs, and view system-wide analytics. This allows for efficient inventory tracking, pricing updates, and real-time order monitoring.
- **Ensure Security:** Security is integral to the platform, and the system employs modern security practices such as hashed passwords, token-based authentication (JWT), and route protection to prevent unauthorized access. User roles are strictly enforced, ensuring that customers and admins have access only to features relevant to their permissions.
- **Promote Scalability:** The architecture is designed to support future expansion. Whether it's integrating third-party payment gateways, adding user features like wishlists or reviews, or scaling to handle thousands of users, the modular design enables flexible enhancements without disturbing existing functionality. The separation of concerns between the frontend, backend, and database ensures maintainability and performance optimization.

User (Customer) Features:

- **Authentication:** Customers can securely register and log in using email and password credentials. Upon successful login, users receive a token for session management and are redirected to their personalized dashboard. Profile management features allow users to update personal details, such as name, contact information, and password.
- **Product Interaction:** Users have access to a wide catalog of products. They can browse all available items or narrow down their search using category filters, price range sliders, and keyword-based queries. Each product page displays detailed information such as images, descriptions, pricing, availability, and user reviews.
- **Cart & Checkout:** The shopping cart allows users to add items, update quantities, or remove products. Cart totals are updated in real-time, showing applicable taxes or

discounts. During checkout, users can provide shipping information and select from available payment methods before finalizing the order.

Technical Scope:

- **Frontend (Angular):** The frontend is built with Angular, using components, services, and routing modules to structure the application efficiently. It ensures responsive design, form validation, and integration with backend APIs through Angular's HttpClient module.
- **Backend (Node.js):** The backend utilizes Node.js with Express.js to create RESTful APIs for all major functionalities, including authentication, product management, and order processing. Middleware is used to validate tokens and handle exceptions gracefully.
- **Database (MongoDB):** MongoDB serves as the NoSQL database, with collections for storing product details, user accounts, and orders. It allows for flexible data modeling, fast querying, and horizontal scalability.
- **Security:** User passwords are encrypted using hashing algorithms like bcrypt. JWT tokens are issued on login and used for secure communication between frontend and backend. Role-based access controls ensure that only authorized users can perform critical actions.

3. System Analysis

System analysis is a crucial phase in the software development life cycle that enables the team to investigate and understand the needs, limitations, and functionalities required for a successful software system. In the context of the e-commerce platform, system analysis involves evaluating the requirements of both customers and administrators to ensure the final solution meets expectations and delivers a seamless online shopping and management experience. Through this phase, the software is logically structured for implementation, identifying how different components will interact and how data will flow through the system.

System analysis involves multiple subcomponents such as the identification of needs, feasibility studies, workflow modeling, system structure, input/output requirements, software specifications, and the application of a suitable software engineering paradigm. The analysis aids in defining both the functional and non-functional requirements of the platform, ensuring that both user experience and backend efficiency are optimized.

This section outlines a detailed exploration of all the subtopics that form the foundation of the e-commerce system's design and development. Each component described below contributes to establishing a reliable, scalable, and secure platform for modern online commerce.

3A. Identification of Need

With the exponential growth of internet usage and digital transformation in businesses, the need for online commerce platforms has become increasingly important. Traditional brick-and-mortar stores are limited by location, operating hours, and manual operations that do not scale well with growing customer bases. Moreover, customers now expect seamless digital experiences with options to browse products, compare prices, place orders, and track shipments from the comfort of their homes.

The rise in demand for contactless transactions, especially post-pandemic, has accelerated the adoption of e-commerce solutions. Businesses require tools that allow them to manage products, handle customer queries, process payments, and update order statuses in real time. Similarly, customers want to experience quick navigation, filtered searches, and secure checkout processes.

This project addresses those pain points by creating a full-stack e-commerce web platform that connects consumers and businesses efficiently. It eliminates geographical and time barriers, offers 24x7 product access, and automates inventory and order processing. Additionally, it integrates user authentication and authorization to ensure secure access and role-specific functionality.

3B. Feasibility Study

A feasibility study evaluates the viability of a project from various perspectives to determine whether it is worth pursuing. It includes technical, economic, operational, and legal assessments to ensure that the e-commerce platform can be implemented successfully without undue risk or cost. This section provides a detailed examination of the four major feasibility areas relevant to this project:

- **Technical Feasibility:** The project leverages proven technologies such as Angular for the frontend, Node.js with Express for the backend, and MongoDB for data persistence. These technologies are widely adopted, well-documented, and supported by large developer communities. The choice of MERN-like architecture (Angular instead of React) ensures modularity, maintainability, and ease of integration with third-party APIs such as payment gateways and shipping services. The development team possesses the necessary skill set, and the technology stack is compatible with current hardware and deployment environments.
- **Economic Feasibility:** The cost of developing the platform is minimal due to the use of open-source tools and frameworks. There is no need for licensing fees, and deployment can be done using cloud services with flexible payment models. This makes the solution affordable for startups and small to medium-sized enterprises. In the long term, the platform reduces operational costs by digitizing inventory, billing, and order management processes, thus eliminating the need for excessive manpower or paperwork.
- **Operational Feasibility:** The platform is designed to be user-friendly and intuitive for both customers and administrators. A clean, responsive user interface ensures seamless interaction on desktops and mobile devices. The role-based access model ensures that different users (customers and admins) have access to only the relevant features, thereby reducing complexity and improving system efficiency. Customer support, sales monitoring, and inventory management are integrated into the platform, making daily operations streamlined and automated.
- **Legal Feasibility:** The system adheres to web development standards and follows industry best practices for security and privacy. It implements secure authentication protocols using JWT and encrypts sensitive data like passwords using hashing algorithms. The project avoids handling sensitive financial information directly, thus reducing exposure to compliance risks such as PCI-DSS. The privacy of user data is respected, aligning the system with data protection laws like the GDPR.

In conclusion, the project is highly feasible across all aspects and is suitable for deployment in real-world commercial environments.

3C. Workflow

The development of the e-commerce platform follows an **Iterative Waterfall Model**, combining the structure of the traditional waterfall method with the flexibility of iterative improvements. Each phase flows logically into the next, while allowing for feedback and refinements from previous stages. This model was selected because it supports clarity in planning, systematic tracking of progress, and structured delivery—especially suitable for academic and internship-based projects.

Below is a breakdown of the key phases in this model as applied to our project:

1. **Requirement Gathering:** This phase involves identifying the key functional and non-functional needs of the platform. User and admin expectations, such as product browsing, cart functionality, and inventory control, are documented through research, benchmarking, and consultations.
2. **System Design:** Based on the requirements, architectural designs and diagrams like DFDs, use case diagrams, and class schemas are created. The system design defines the structure of the frontend, backend, and database components to ensure proper integration and role-based access.
3. **Implementation:** Development begins in this phase. Angular is used to create the frontend components (e.g., product listing, cart), while Node.js/Express manages backend routes and APIs. MongoDB is configured for data persistence. Modular code practices and Git version control are used.
4. **Testing:** Each component is tested for both functionality and performance. Unit testing ensures individual modules work correctly, while integration testing checks the consistency between backend APIs and frontend calls. Bugs are identified and corrected before proceeding.
5. **Deployment:** Once the application is tested, it is deployed to a live server or cloud environment (such as Heroku or Vercel). Environment variables and production-level security practices (like HTTPS and database rules) are implemented.
6. **Maintenance:** Post-deployment, the application is monitored for issues, and updates or enhancements are rolled out as needed. This includes feature extensions, database scaling, and user feedback-based improvements.

The workflow ensures continuous validation and delivers a stable product ready for real-world usage while maintaining adaptability at each stage.1. Requirement Gathering 2. System Design 3. Implementation 4. Testing 5. Deployment 6. Maintenance

3D. Study of the System

The e-commerce platform is divided into distinct modules to ensure separation of concerns and improve maintainability. Each module is designed to handle specific functionalities and is integrated with the others through well-defined APIs and frontend services.

- **Login Module:** This module authenticates users by verifying credentials against the database. It supports role-based access, generating JWT tokens upon successful login. This ensures that users and admins are redirected to appropriate dashboards.
- **Product Module:** Displays a list of all products, filtered by categories, price, and keywords. It dynamically fetches product data using Angular services connected to backend APIs. Each product page includes detailed descriptions, images, and stock availability.
- **Cart Module:** Allows customers to add products to the cart, update quantities, and remove unwanted items. Real-time price calculation and total amount updates are performed, and the cart is persisted across sessions using local storage or user sessions.

- **Order Module:** Facilitates the checkout process. It collects address and payment details, confirms the order, and generates an order record in the database. Users can track their order status and view historical orders.
- **Admin Panel:** Offers administrative tools to manage users, products, and orders. Admins can add/edit/delete product listings, update stock, review customer orders, and perform account-level changes on users.** User, product, and order management.

3E. Input and Output

Input and output design defines how the system interacts with users and other systems. Properly structured input mechanisms and meaningful outputs improve system usability and user satisfaction.

- **Inputs:**
  - User registration forms (name, email, password)
  - Login credentials (email, password)
  - Product search queries and filter selections
  - Cart interactions (add/remove product, quantity updates)
  - Checkout form (address, contact details)
  - Admin panel inputs (product info, pricing, stock levels)
- **Outputs:**
  - Confirmation messages (registration, login, order)
  - Product lists with real-time filtering and search results
  - Cart total calculations and previews
  - Order tracking status (shipped, delivered, cancelled)
  - Admin dashboards displaying analytics, product inventory, and user activity logs

3F. Software Requirement Specifications

This section outlines the functional and non-functional software requirements necessary for implementing the e-commerce platform.

- **Frontend:**
  - Angular 15+ for component-based architecture and reactive programming
  - Bootstrap 5 for responsive and accessible UI
  - Angular Router for navigation
  - Angular HttpClient for RESTful API integration
- **Backend:**
  - Node.js 18+ and Express for server-side logic and routing
  - Mongoose ODM for MongoDB interaction
  - Middleware for authentication, validation, and error handling
- **Database:**
  - MongoDB Atlas for scalable cloud-based data storage
  - Collections: Users, Products, Orders, Carts

- **Testing Tools:**
  - Postman for API testing
  - Chrome DevTools for debugging UI interactions and inspecting network calls

3G. Software Engineering Paradigm Applied

The development of this e-commerce platform is based on a hybrid of established software engineering paradigms, ensuring modularity, scalability, and maintainability.

- **Programming Paradigm:**
  - The frontend follows a **component-based architecture** using Angular, where UI elements are broken into reusable components (e.g., ProductCard, CartItem, Navbar).
  - The backend follows a **modular programming** approach, with each module (auth, product, order) organized into separate routes and controllers.
- **Design Paradigm:**
  - Implements the **Model-View-Controller (MVC)** pattern on the backend. Models define schemas (e.g., User, Product), views are served via Angular, and controllers handle business logic.
  - The frontend uses **service-based abstraction**, allowing for separation between UI logic and data-fetching functions.
- **Development Model:**
  - The project uses the **Iterative Waterfall Model**, where each development phase (Requirements, Design, Implementation, Testing, Deployment, Maintenance) is completed in sequence with room for iteration and refinement based on feedback and testing.

4. System Design

System Design is the blueprint of your application. It defines **how** the system will function, how components interact, and how data flows between users, the application, and the database. This phase bridges the gap between the abstract requirements and concrete implementation.

4A. Data Flow Diagram (DFD)

Purpose:
  DFDs illustrate how data moves through the system. It identifies **external entities**, **processes**, and **data stores** in a visual format.

- **Level 0 (Context Diagram):**
   Shows a high-level view of the system with users (Customers/Admins), and how they interact with the main system.

   1. External Entities: `User`, `Admin`

   2. Central Process: `E-commerce System`

   3. Data Stores: `Product DB`, `Order DB`, `User DB`

- **Level 1 (Detailed View):**
   Breaks the system into smaller sub-processes:

   1. **Login Module** – Authenticates users

   2. **Product Browsing** – Retrieves product data from DB

   3. **Cart Update** – Adds/removes products

   4. **Checkout & Order Placement** – Finalizes orders

   5. **Admin Controls** – Admin manages users/products/orders

4B. Sequence Diagram

Purpose:
 Describes the **sequence of events** in a particular user interaction. It focuses on **timing and flow between system components.**

- **Use Case: Placing an Order**

    1. User logs in (Auth service)

    2. Views products (API fetch)

    3. Adds to cart (cart state updates)

    4. Proceeds to checkout (form data submitted)

    5. Order stored in MongoDB

    6. Admin receives notification or sees it on dashboard

This shows real-time interaction between frontend, backend, and database.

4C. Use Case Diagram

Purpose:
 Visually maps out **actors** and their **actions** within the system.

- **Actors:**

    - User

- **Use Cases:**

    - Register, Login

    - Browse Products, Add to Cart, Checkout,

Helps developers and stakeholders understand what functionalities each role has access to.
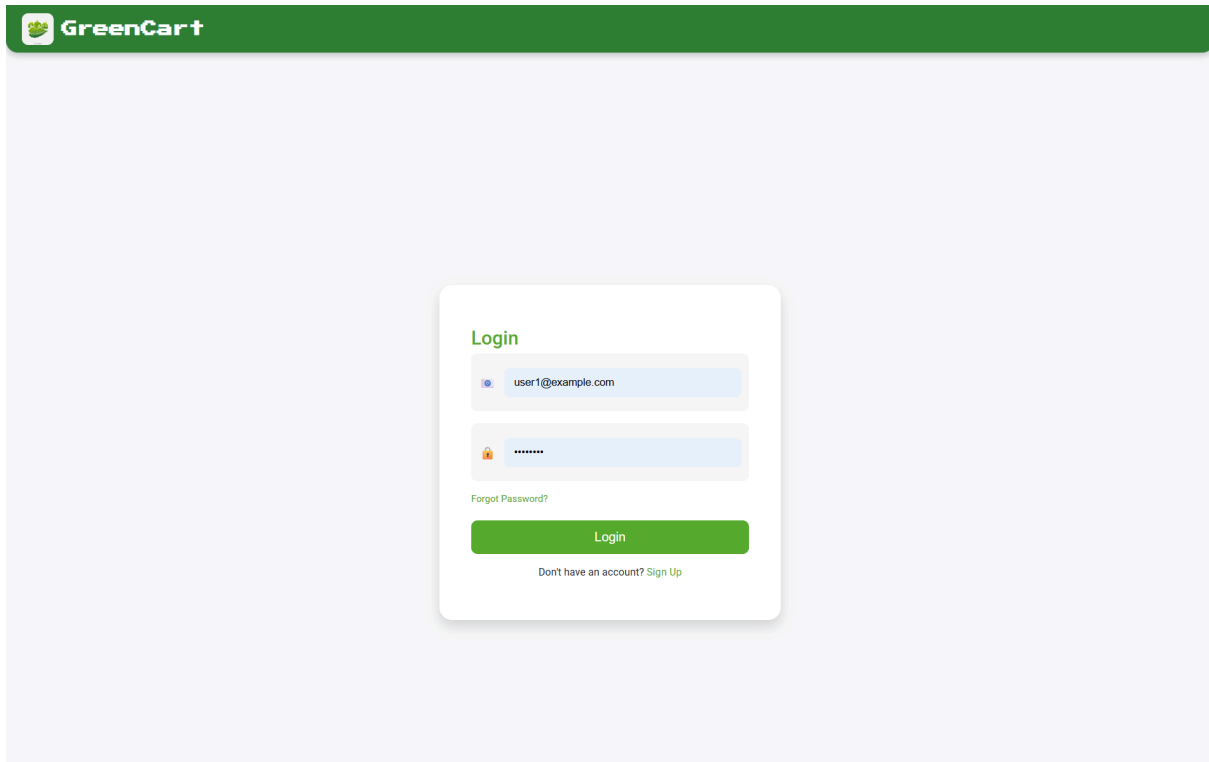
4D. Schema/Class Diagram

Purpose:
 Represents the **database structure** and the **relationships between entities**.

- **User:** `email`, `password`

- **Product:** `id`, `name`, `price`, `category`, `stock`

- **Order:** `id`, `userId`, `items[]`, `status`, `total`

- **Cart:** `userId`, `productId`, `quantity`

UI/UX

5. Frontend

- **Login page:-**

#code:-

Css

```css
body {

  background: linear-gradient(to right, #a8e063,
#56ab2f);

  font-family: 'Segoe UI', sans-serif;

}




.login-container {

  height: 100vh;
```

```css
  display: flex;

  justify-content: center;

  align-items: center;

}


.login-card {

  background: white;

  padding: 40px;

  border-radius: 16px;

  box-shadow: 0 8px 16px rgba(0,0,0,0.15);

  width: 350px;

}


.login-title {

  color: #56ab2f;

  margin-bottom: 5px;

}


.login-subtitle {

  color: #888;

  margin-bottom: 20px;
```

```css
}


.input-group {

  display: flex;

  align-items: center;

  margin-bottom: 15px;

  background: #f5f5f5;

  padding: 10px;

  border-radius: 8px;

}


.input-group .icon {

  margin-right: 10px;

  color: #56ab2f;

}


.input-group input {

  border: none;

  background: transparent;

  flex: 1;

  outline: none;
```

```css
    font-size: 14px;

}


.options {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-bottom: 20px;

}


.forgot-password {

    color: #56ab2f;

    text-decoration: none;

    font-size: 12px;

}


.login-button {

    width: 100%;

    padding: 12px;
```

```css
  background: #56ab2f;

  color: white;

  border: none;

  border-radius: 8px;

  font-size: 16px;

  cursor: pointer;

}


.login-button:hover {

  background: #4a9c28;

}


.create-account {

  margin-top: 15px;

  text-align: center;

  font-size: 13px;

}

.create-account a {

  color: #56ab2f;

  text-decoration: none;

}

}
```

HTML

```html
<div class="login-container">

  <div class="login-card">

    <h2 class="login-title">Login</h2>


    <form #loginForm="ngForm" (ngSubmit)="onLogin()"
novalidate>

      <!-- Email Field -->

      <div class="input-group">

        <span class="icon">📧</span>

        <input type="email" placeholder="Enter your
Email"
               [(ngModel)]="email" name="email"
               required email #emailRef="ngModel" />
      </div>

      <div class="error" *ngIf="emailRef.invalid &&
emailRef.touched">

        <span
*ngIf="emailRef.errors?.['required']">Email is
required.</span>

        <span
*ngIf="emailRef.errors?.['email']">Invalid email
format.</span>

      </div>
```

```html
      <!-- Password Field -->

      <div class="input-group">

        <span class="icon">🔒</span>

        <input type="password" placeholder="Enter
Password"

                [(ngModel)]="password" name="password"

                required minlength="7"
#passRef="ngModel" />

      </div>

      <div class="error" *ngIf="passRef.invalid &&
passRef.touched">

        <span
*ngIf="passRef.errors?.['required']">Password is
required.</span>

        <span
*ngIf="passRef.errors?.['minlength']">Password must be
at least 7 characters.</span>

      </div>


      <!-- Options -->

      <div class="options">

        <a href="#" class="forgot-password">Forgot
Password?</a>
```

```html
        </div>


        <!-- Submit -->

        <button type="submit" class="login-button"
[disabled]="loginForm.invalid">

          Login

        </button>

    </form>


        <p class="create-account"> Don't have an
account?  <a routerLink="/signup">Sign Up</a></p>



    </div>

</div>
```

Typescript:-

```typescript
import { Component } from '@angular/core';

import { Router } from '@angular/router';

import { AuthService } from '../auth.service';



@Component({{
```

```typescript
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  email = '';
  password= '';
  loginFailed = false;
  showPassword: boolean = false;


  constructor(private readonly authServices:
AuthService, private  readonly router: Router) {}


  onLogin(): void {
    console.log('Attempting login:', this.email);


    const isValid = this.authServices.login(this.email,
this.password);



    if (isValid) {
      console.log("Login successful");
```
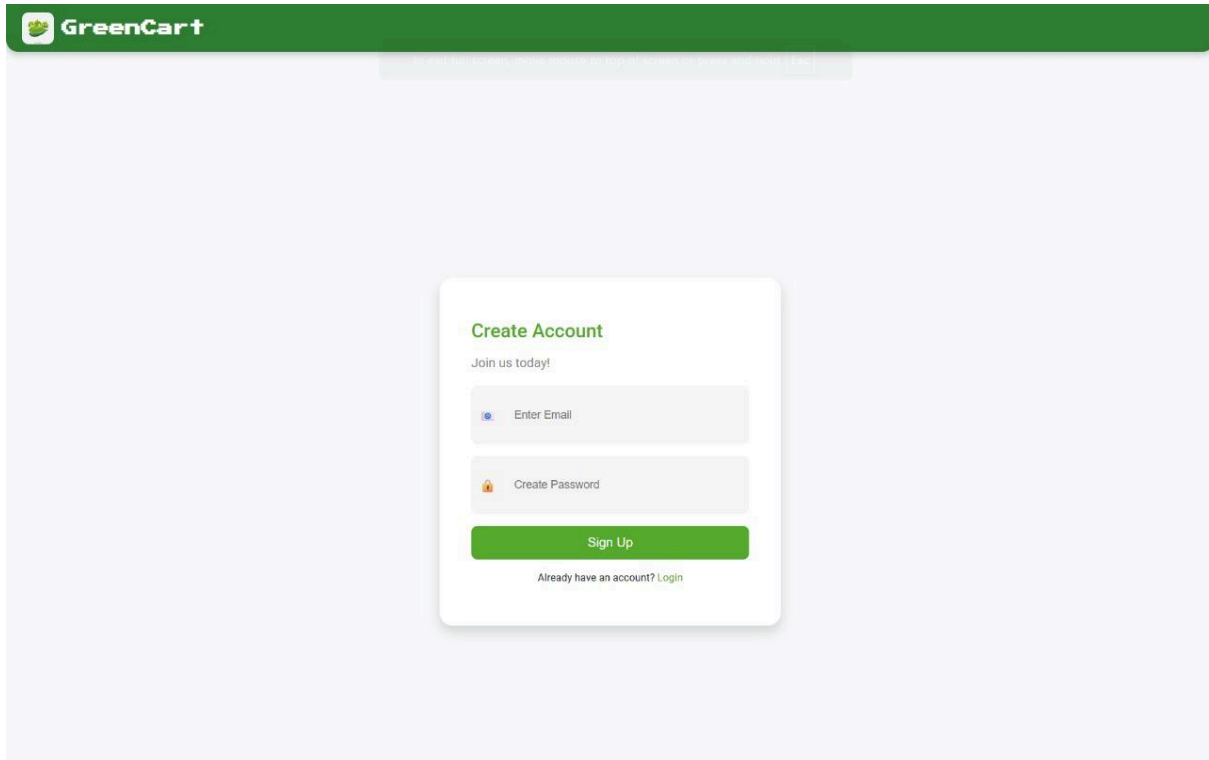
```
        this.router.navigate(['/home']);

    } else {

        console.log("Login failed");

        this.loginFailed = true;

    }

  }



onsignup(){

  this.router.navigate(['/signup']);

}



}
```

- **SignUp Page:-**

Code:-

css

```css
body {
  background: linear-gradient(to right, #a8e063, #56ab2f);

  font-family: 'Segoe UI', sans-serif;

}



.signup-container {

  height: 100vh;

  display: flex;

  justify-content: center;
```

```css
  align-items: center;

}


.signup-card {

  background: white;

  padding: 40px;

  border-radius: 16px;

  box-shadow: 0 8px 16px rgba(0,0,0,0.15);

  width: 350px;

}


.signup-title {

  color: #56ab2f;

  margin-bottom: 5px;

}


.signup-subtitle {

  color: #888;

  margin-bottom: 20px;

}
```

```css
.input-group {

  display: flex;

  align-items: center;

  margin-bottom: 15px;

  background: #f5f5f5;

  padding: 10px;

  border-radius: 8px;

}


.input-group .icon {

  margin-right: 10px;

  color: #56ab2f;

}


.input-group input {

  border: none;

  background: transparent;

  flex: 1;

  outline: none;

  font-size: 14px;

}
```

```css
.eye-button {

  background: transparent;

  border: none;

  cursor: pointer;

  font-size: 18px;

  margin-left: 8px;

}


.signup-button {

  width: 100%;

  padding: 12px;

  background: #56ab2f;

  color: white;

  border: none;

  border-radius: 8px;

  font-size: 16px;

  cursor: pointer;

}


.signup-button:hover {
```

```css
    background: #4a9c28;

}


.error {

  color: red;

  font-size: 12px;

  margin-top: -10px;

  margin-bottom: 10px;

}


.login-link {

  margin-top: 15px;

  text-align: center;

  font-size: 13px;

}


.login-link a {

  color: #56ab2f;

  text-decoration: none;

}
```

```html
<div class="signup-container">

  <div class="signup-card">

    <h2 class="signup-title">Create Account</h2>

    <p class="signup-subtitle">Join us today!</p>


    <form #signupForm="ngForm" (ngSubmit)="onSignup()" novalidate>

      <div class="input-group">

        <span class="icon">📧</span>

        <input type="email" placeholder="Enter Email"
               [(ngModel)]="email" name="email"
               required email #emailRef="ngModel" />
      </div>

      <div class="error" *ngIf="emailRef.invalid && emailRef.touched">

        <span *ngIf="emailRef.errors?.['required']">Email is required.</span>

        <span *ngIf="emailRef.errors?.['email']">Invalid email format.</span>

      </div>
```

```html
<div class="input-group">

    <span class="icon">🔒</span>

    <input [type]="showPassword ? 'text' :
'password'"

        placeholder="Create Password"

        [(ngModel)]="password" name="password"

        required minlength="7"
#passRef="ngModel" />

</div>

<div class="error" *ngIf="passRef.invalid &&
passRef.touched">

    <span
*ngIf="passRef.errors?.['required']">Password is
required.</span>

    <span *ngIf="passRef.errors?.['minlength']">Min
7 characters.</span>

</div>


<button type="submit" class="signup-button"
[disabled]="signupForm.invalid">

    Sign Up

</button>

</form>
```

```html
    <p class="login-link">Already have an account? <a
routerLink="/login">Login</a></p>

  </div>

</div>
```

```typescript
import { Component } from '@angular/core';

import { Router } from '@angular/router';

import { AuthService } from '../auth.service';




@Component({

  selector: 'app-signup',

  templateUrl: './signup.component.html',

  styleUrls: ['./signup.component.css']

})

export class SignupComponent {

  email = '';

  password = '';

  showPassword = false;
```

```typescript
  constructor(private readonly authServices:
AuthService, private readonly router: Router) { }


  togglePassword() {

    this.showPassword = !this.showPassword;

  }



  onSignup() {

    console.log('Signup:', this.email, this.password);

      const isValid =
this.authServices.login(this.email, this.password);



    if (isValid) {

      alert("Email Exist try to login");

      this.router.navigate(['/login']);

    } else {

      this.authServices.register(this.email,
this.password);



    }

  }

}
```
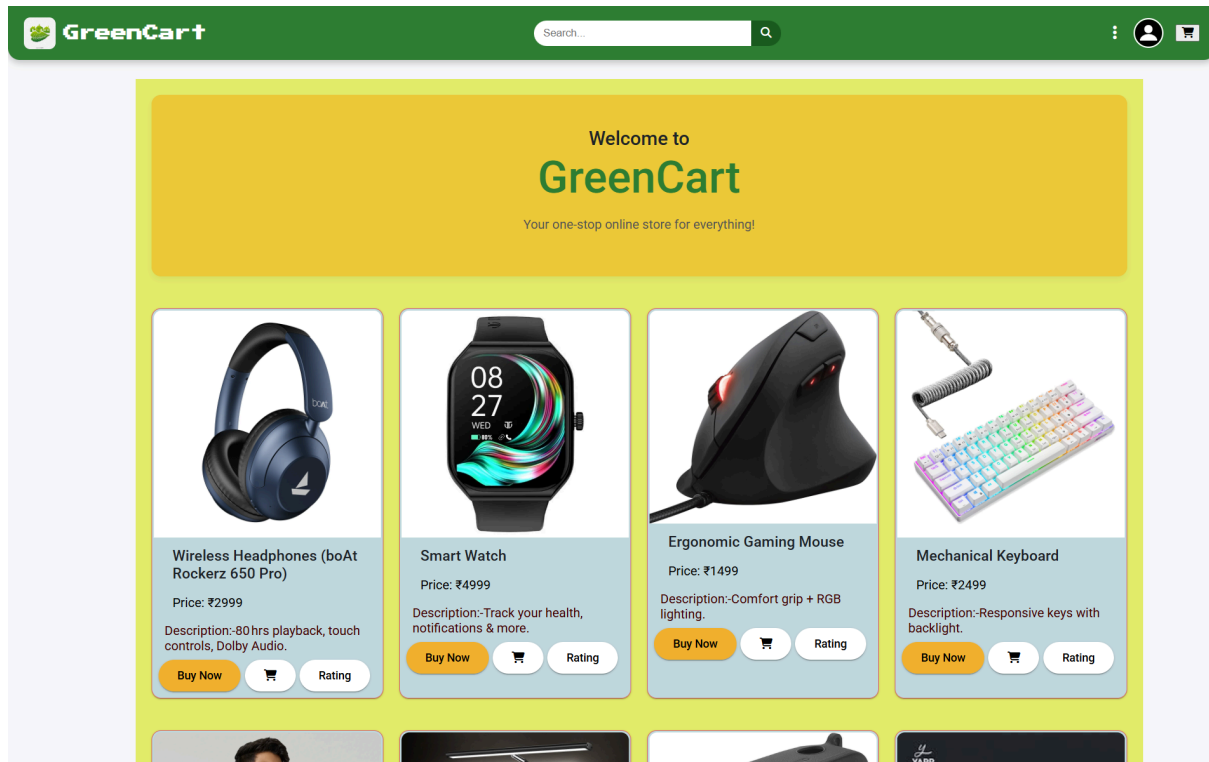
Home Page:-

Code:-
css:-

```css
html, body {

    height: 100%;

    margin: 0;

}



.home-container {

    background: #e4ec6b;

    padding: 20px;

    display: flex;

    flex-direction: column;
```

```css
  width:80vw;



}



.user-banner {

  text-align: center;

  background: #efb31aa0;

  padding: 40px 20px;

  border-radius: 12px;

  margin-bottom: 30px;

  box-shadow: 0 4px 8px rgba(0,0,0,0.05);

}



.user-banner h1 {

  font-size: 2.5rem;

  margin: 0;

}



.user-banner p {

  color: #555;

}
```

```css
.product-grid {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(250px,
1fr));

  gap: 20px;

  background-color: #e4ec6b;

}


.product-card {

  transition: transform 0.3s ease, box-shadow 0.3s
ease;

  background-color: #bad4f6d3;

  color: rgb(68, 2, 2);

  border: 1px solid rgb(219, 101, 101);

  border-radius: 8px;

  overflow: hidden;

  display: flex;

  flex-direction: column;

  justify-content: space-between;

  height: 100%; /* Make all cards the same height */

  padding-bottom: 10px;
```

```css
}

.product-card img{
  padding:2px;
  width: 100%;
  height: 200px;
  object-fit: cover;
}

mat-card-actions {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  gap: 8px;
  padding: 10px;
  box-sizing: border-box;
}

mat-card-action button{
  flex: 1 1 auto;
  min-width: 70px;
  white-space: nowrap;
  padding: 8px;
}
```

```css
.product-card button{

  background-color:#fff;

  color:black;

  margin-right: 5px;

  object-fit: cover;

}


.product-card:hover {

  transform: translateY(-5px);

  box-shadow: 14px 16px 12px rgba(0,0,0,0.1);

}

.mat-card-pName{

  margin-left:10px;

}

.pPrice{

  margin-left:10px;

  color: black;

}

.Dbtn{

    transition: transform 0.3s ease;

}
```

```css
.Dbtn:hover{

    transform: scale(1.1);



}

.info-box {

  position: absolute;

  bottom: 10px;

  left: 10px;

  right: 10px;

  background: rgba(255, 255, 255, 0.95);

  padding: 10px;

  border-radius: 8px;

  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);

  opacity: 0;

  transform: translateY(20px);

  transition: opacity 0.3s ease, transform 0.3s ease;
}

.Dbtn:hover .info-box {

  opacity: 1;

  transform: translateY(0);

}
```

Html

```html
<div class="home-container">



  <!-- User Banner -->

  <div class="user-banner">

    <h1>Welcome to GreenCart</h1>

    <p>Your one-stop online store for everything!</p>

  </div>



  <!-- Product Grid -->

  <div class="product-grid">

    <!-- ✅ Loop through filtered products -->

    <mat-card

      class="product-card"

      *ngFor="let product of filteredProducts"

      [@fadeInAnimation]

      style="width: 200px; height:auto; object-fit:
cover;"> <!-- ✅ Animation remains -->



        <!-- ✅ Product Image from DB -->

      <img mat-card-image [src]="product.image"
alt="Product Image" style="width: 200px; height: 200px;
object-fit: cover;"/>
```

```html
        <!-- ✅ Product Name -->

        <mat-card-title class="mat-card-pName">{{
product.name }}</mat-card-title>


        <!-- ✅ Product Price -->

        <mat-card-subtitle class="pPrice">Price:- ₹{{
product.price }}</mat-card-subtitle>


        <!-- ✅ Product Description -->

        <mat-card-content>

          <p>{{ product.description }}</p>

        </mat-card-content>



        <mat-card-actions>

        <button mat-raised-button
style="background-color: rgba(255, 166, 0, 0.798);">Buy
Now</button>



        <!-- ✅ Add to Cart action -->

        <button mat-raised-button
(click)="addToCart(product)">

          <i class="fas fa-shopping-cart"></i>
```

```html
        </button>


        <button mat-raised-button>Rating</button>

      </mat-card-actions>


    </mat-card> <!-- ✅ Each product loaded dynamically
-->



  </div>



</div>
```

Typescript

```typescript
import { Component, OnInit } from '@angular/core';

import { trigger, transition, style, animate } from
'@angular/animations';

import { CartService } from
'../../services/cart.service';

import { ProductService } from
'../../services/product.service';

import { SearchService } from
'../../services/serchService';
```

```typescript
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  animations: [
    trigger('fadeInAnimation', [
      transition(':enter', [
        style({ opacity: 0, transform: 'scale(0.9)' }),
        animate('600ms ease-out', style({ opacity: 1,
transform: 'scale(1)' }))
      ])
    ])
  ]
})
export class HomeComponent implements OnInit {
  products: any[] = [];
  filteredProducts: any[] = [];


  constructor(
    private readonly cartService: CartService,
    private readonly productService: ProductService,
    private readonly searchService: SearchService
```

```typescript
  ) {}


  ngOnInit(): void {

    this.productService.getProducts().subscribe({

      next: (res) => {

        this.products = res;

        this.filteredProducts = res;

      },

      error: (err) => console.error('Error loading
products:', err)

    });

    this.searchService.searchTerm$.subscribe(term => {

      this.filteredProducts =
this.products.filter(product =>

product.name.toLowerCase().includes(term.toLowerCase())

      );

    });

  }

  addToCart(product: any) {

    this.cartService.addItem(product);

  }

}
```

NAV-BAR:-

CSS:-

```css
/* === NAVBAR === */

.navbar {

  display: flex;

  justify-content: space-between;

  align-items: center;

  border-bottom-left-radius:15px ;

  border-bottom-right-radius:15px ;

  padding: 10px 25px;

  background-color: #56ab2f;

  position: relative;

  flex-wrap: wrap;

}


/* Left */

.nav-left {

  display: flex;

  align-items: center;

}

.logo {
```

```css
  height: 40px;

  cursor: pointer;

  margin-right: 10px;

  border-radius: 15px;

}

.site-name {

  font-size: 1.5rem;

  font-weight: bold;

  color: #fff;

  font-family: "Press Start 2P", monospace;

}


/* Center */

.nav-center {

  display: flex;

  align-items: center;

  flex: 1;

  justify-content: center;

}

.search-bar {

  padding: 5px 10px;
```

```css
  border-radius: 20px;

  border: 1px solid #ccc;

  width: 200px;

}

.search-btn {

  margin-left: 5px;

  background-color: #2196f3;

  color: white;

  border: none;

  border-radius: 20px;

  padding: 5px 10px;

  cursor: pointer;

}


/* Right */

.nav-right {

  display: flex;

  align-items: center;

  gap: 15px;

}

.icon-button {
```

```css
  background: none;

  border: none;

  font-size: 1.2rem;

  cursor: pointer;

}

.profile {

  height: 35px;

  width: 35px;

  border-radius: 50%;

  cursor: pointer;

}


/* === DROPDOWN MENU === */

.dropdown-menu {

  position: absolute;

  top: 60px;

  right: 100px;

  background-color: white;

  border: 1px solid #ccc;

  padding: 10px;

  z-index: 1000;

```

```css
  display: flex;

  flex-direction: column;

  gap: 10px;

}

.dropdown-menu button {

  border: none;

  background: none;

  cursor: pointer;

  text-align: left;

}


/* === CART === */

.cart-container {

  position: absolute;

  top: 60px;

  right: 10px;

  background: white;

  width: 300px;

  max-height: 400px;

  overflow-y: auto;

  border: 1px solid #ccc;
```

```css
    padding: 10px;

    z-index: 1000;

}

.item-card {

    display: flex;

    gap: 10px;

    margin-bottom: 10px;

}

.item-img {

    width: 60px;

    height: 60px;

    object-fit: cover;

}

.item-details {

    flex: 1;

}

.cart-total {

    text-align: right;

    font-weight: bold;

    margin-top: 10px;

}
```

```css
/* === SELLER FORM POPUP === */

.seller-container {

  position: fixed;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  width: 90%;

  max-width: 400px;

  max-height: 90vh;

  background-color: #fff;

  border: 2px solid #19d222;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

  padding: 25px;

  z-index: 9999;

  border-radius: 8px;

  overflow-y: auto;

}


.seller-container h2 {

  margin-top: 0;
```

```css
  color: #1976d2;

  text-align: center;

}


.seller-container p {

  text-align: center;

  color: #555;

  margin-bottom: 20px;

}


.seller-container form {

  display: flex;

  flex-direction: column;

  gap: 10px;

}


.seller-container label {

  font-weight: bold;

}


.seller-container input,
```

```css
.seller-container textarea,
.seller-container select {
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
}


.seller-container .submit {
  background-color: #1976d2;
  color: white;
  padding: 10px;
  font-weight: bold;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}


.seller-container .submit:disabled {
  background-color: #ccc;
```

```css
    cursor: not-allowed;

}



/* === RESPONSIVE === */

@media (max-width: 768px) {

  .nav-center {

    margin-top: 10px;

  }

  .cart-container {

    right: 0;

    width: 100%;

  }

}
```

HTML;-

```html
<nav class="navbar">

  <div class="nav-left">

    <img
src="https://encrypted-tbn0.gstatic.com/images?q=tbn:AN
d9GcQPgSfUAo_uWMMQ11mIb8QeLYEfdanxWWmZ0g&s" alt="Logo"
class="logo" (click)="navigate()"/>

    <span class="site-name">GreenCart</span>
```

```html
    </div>

    <!-- Search Bar -->
    <div class="nav-center" *ngIf="!isLoginPage">
      <input type="text" [(ngModel)]="searchValue"
placeholder="Search..." class="search-bar" />
      <button class="search-btn" (click)="onSearch()"><i
class="fas fa-search"></i></button>
    </div>

    <!-- Profile, Cart, Dropdown -->
    <div class="nav-right" *ngIf="!isLoginPage">
      <!-- Dropdown Menu -->
      <button class="icon-button"
(click)="toggleDropdown()">
        <i class="fas fa-ellipsis-v"></i>
      </button>

      <div class="dropdown-menu" *ngIf="isDropdownOpen">
        <button>Order History</button>
        <button (click)="sellerButtonClick()">Become a
Seller</button>
        <button>About Us</button>
```

```html
        <button (click)="logout()">Logout</button>

    </div>


    <!-- Profile Button -->

    <img
src="https://www.pngmart.com/files/23/Profile-PNG-Photo
.png" alt="Profile" class="profile"
(click)="profileBtnClick()"/>


    <!-- Cart Button -->

    <button class="icon-button" (click)="toggleCart()">

      <i class="fas fa-shopping-cart"></i>

      <span>{{ cartItemCount }}</span>

    </button>


    <!-- Cart Items -->

    <div class="cart-container" *ngIf="isCartOpen">

      <div *ngFor="let item of cartItems"
class="item-card">

        <img [src]="item.image" alt=""
class="item-img">

        <div class="item-details">

          <h4>{{ item.name }}</h4>
```

```html
        <p>{{ item.price }}</p>

        <button mat-button
(click)="decreaseQuantity(item)">-</button>

        <span>{{ item.quantity }}</span>

        <button mat-button
(click)="increaseQuantity(item)">+</button>

        <button mat-button
(click)="removeItem(item.id)">Delete</button>

      </div>

    </div>


    <div class="cart-total">

      <strong>Total: ₹{{ total }}</strong>

    </div>

  </div>

</div>


  <div class="seller-container" *ngIf="showSellerForm">

  <h2>Bcome a Seller</h2>

  <p>List Your Product Noww!!</p>


  <form (ngSubmit)="addProduct()"
#productForm="ngForm">
```

```html
    <!-- Image Upload -->

    <label>Product Image</label>

    <input type="file"
(change)="onImageSelected($event)" required />



    <!-- Product Name -->

    <label>Product Name</label>

    <input type="text" [(ngModel)]="newProduct.name"
name="name" required />



    <!-- Product Price -->

    <label>Price (₹)</label>

    <input type="number" [(ngModel)]="newProduct.price"
name="price" required />



    <!-- Product Description -->

    <label>Description</label>

    <textarea [(ngModel)]="newProduct.description"
name="description" required></textarea>



    <!-- Product Category -->

    <label>Category</label>
```

```html
    <select [(ngModel)]="newProduct.category"
name="category" required>

        <option value="" disabled selected>Select
Category</option>

        <option value="Fruits">Fruits</option>

        <option value="Vegetables">Vegetables</option>

        <option value="Dairy">Dairy</option>

        <option value="Bakery">Bakery</option>

        <option value="Others">Others</option>

    </select>


    <!-- Submit Button -->

    <button class="submit"
[disabled]="!productForm.form.valid">Add
Product</button>

  </form>

  </div>

</nav>
```

Typescript:-

```typescript
import { Component, OnInit, OnDestroy } from
'@angular/core';
```

```typescript
import { Router, NavigationEnd } from
'@angular/router';

import { filter, Subject } from 'rxjs';

import { takeUntil } from 'rxjs/operators';

import { AuthService } from '../auth.service';

import { CartService, CartItem } from
'../../services/cart.service';

import { HttpClient } from '@angular/common/http';

import { SearchService } from
'../../services/serchService';


@Component({

  selector: 'app-header',

  templateUrl: './header.component.html',

  styleUrls: ['./header.component.css']

})
export class HeaderComponent implements OnInit,
OnDestroy {

  isDarkTheme = false;

  isLoginPage = false;

  isAuthenticated = false;

  isDropdownOpen = false;

  isCartOpen = false;
```

```typescript
  isSeller = false;

  showSellerForm = false;

  showAddProductPopup = false;

  cartItemCount = 0;

  cartItems: CartItem[] = [];

  total = 0;

  searchValue = '';


  private readonly destroy$ = new Subject<void>();


  newProduct = {

    name: '',

    price: null,

    description: '',

    category: '',

    image: ''

  };


  constructor(

    private readonly cartService: CartService,

    private readonly authService: AuthService,
```

```typescript
    private readonly router: Router,
    private readonly http: HttpClient,
    private readonly searchService: SearchService
  ) {
    this.router.events.pipe(
      filter(event => event instanceof NavigationEnd),
      takeUntil(this.destroy$)
    ).subscribe((event: NavigationEnd) => {
      this.isLoginPage = event.urlAfterRedirects === '/login';
    });
  }


  ngOnInit() {

this.authService.authStatus.pipe(takeUntil(this.destroy$))
      .subscribe(status => this.isAuthenticated = status);



this.cartService.cartCount$.pipe(takeUntil(this.destroy$))
      .subscribe(count => this.cartItemCount = count);
```

```typescript
      this.loadCart();

  }


  ngOnDestroy() {

    this.destroy$.next();

    this.destroy$.complete();

  }


  toggleTheme() {

    this.isDarkTheme = !this.isDarkTheme;

    document.body.classList.toggle('dark-theme',
this.isDarkTheme);

  }



    onSearch() {

      console.log(this.searchValue);

    this.searchService.setSearchTerm(this.searchValue);

  }


  toggleDropdown() {

    this.isDropdownOpen = !this.isDropdownOpen;
```

```
  }


  toggleCart() {

    this.isCartOpen = !this.isCartOpen;

  }


  profileBtnClick() {

    this.router.navigate(['/profile']);

  }


  navigate() {

    this.router.navigate(['/home']);

  }


  logout() {

    this.authService.logout();

    this.router.navigate(['/login']);

  }


  loadCart() {

    this.cartItems = this.cartService.getItems();
```

```typescript
      this.total = this.cartService.getTotal();

  }



  increaseQuantity(item: CartItem) {

    this.cartService.addItem(item);

    this.loadCart();

  }



  decreaseQuantity(item: CartItem) {

    this.cartService.decreaseItem(item.id);

    this.loadCart();

  }



  removeItem(id: number) {

    this.cartService.removeItem(id);

    this.loadCart();

  }



  sellerButtonClick() {

    this.showSellerForm = !this.showSellerForm;

  }
```

```typescript
  onImageSelected(event: any) {

    const file = event.target.files[0];

    if (!file) return;


    const reader = new FileReader();

    reader.onload = () => {

      this.newProduct.image = reader.result as string;

    };

    reader.readAsDataURL(file);

  }


  addProduct() {

    console.log(this);

    if (!this.newProduct.name || !this.newProduct.price
|| !this.newProduct.description ||
!this.newProduct.category || !this.newProduct.image) {

      alert("! Please fill all fields");

      return;

    }
```
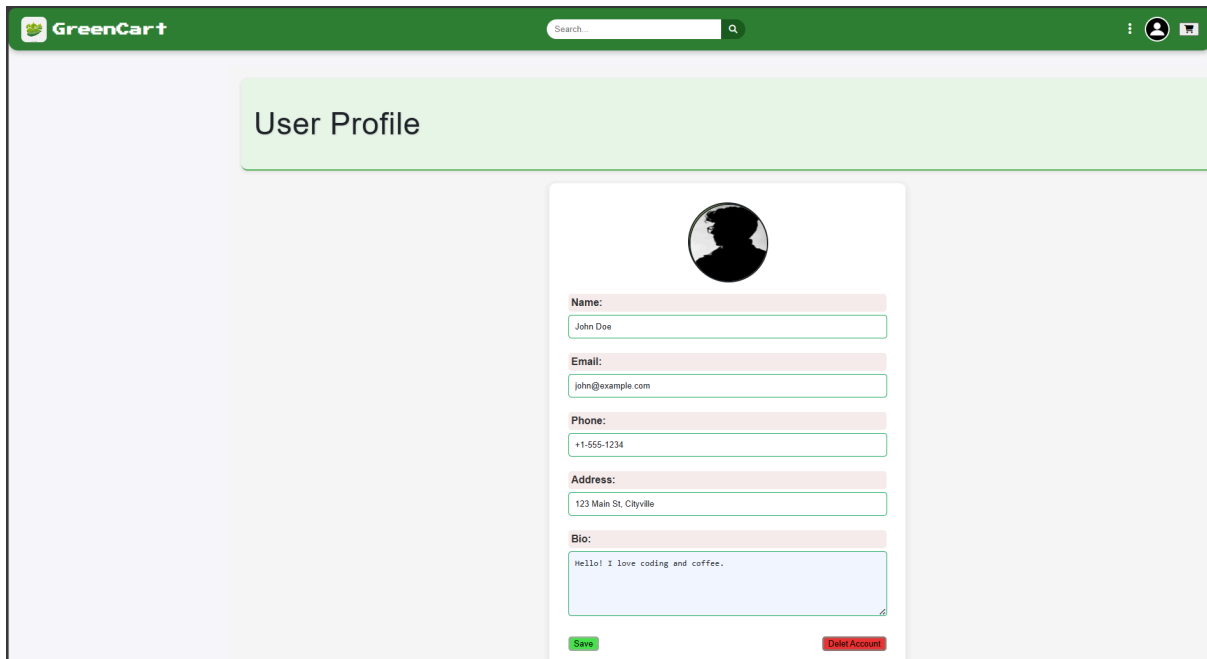
```
this.http.post('http://localhost:5000/api/products',
this.newProduct).subscribe({

    next: (res) => {

        console.log('✅ Product added:', res);

        alert('✅ Product added successfully!');

        this.resetProductForm();

    },

    error: (err) => {

        console.error('❌ Upload failed:', err);

        alert('❌ Failed to upload product');

    }

  });

}


resetProductForm() {

  this.showAddProductPopup = false;

  this.newProduct = {

    name: '',

    price: null,

    description: '',

    category: '',
```

```
        image: ''
    };
  }
}
```

USER PROFILE:-

Code:-

```css
.profile-container {

  font-family: Arial, sans-serif;

  padding: 20px;

  background-color: #f5f5f5;

  min-height: 100vh;

  min-width: 80vw;

}


.header {

  text-align: center;

  margin-bottom: 30px;
```

```css
  font-size: 24px;

  color: #333;

  font-weight: bold;

  border-bottom: 2px solid #4caf50;

  padding-bottom: 10px;

  margin-top: 0;

  margin-bottom: 20px;

  text-transform: castom;

  letter-spacing: 1px;

  font-family: 'Arial', sans-serif;

  text-shadow: 1px 1px 2px rgba(0,0,0,0.1);

  background-color: #e8f5e9;

  padding: 20px;

  border-radius: 10px;

  box-shadow: 0 2px 4px rgba(0,0,0,0.1);

  transition: background-color 0.3s ease;


}


.profile-card {
```

```css
  background-color: white;

  border-radius: 10px;

  max-width: 500px;

  margin: 0 auto;

  padding: 30px;

  box-shadow: 0 4px 8px rgba(0,0,0,0.1);

  transition: transform 0.2s ease;

}


.profile-image {

  width: 120px;

  height: 120px;

  border-radius: 50%;

  background-color: #a0c529ad;

  margin: 0 auto 20px auto;

  border-width: 2px;

  border-style: solid;

}


.profile-info label {

  display: block;
```

```css
    font-weight: bold;

    margin-top: 15px;

    background-color: #f9ecec;

    padding: 5px;

    border-radius: 5px;

    color: #333;

    font-size: 16px;


}


.profile-info input,

.profile-info textarea {

    width: 479px;

    padding: 10px;

    margin-top: 5px;

    border-radius: 5px;

    border: 1px solid #1aa65b;

}


.profile-info textarea {

    resize: vertical;

```

```css
  height: 80px;

  background-color: #f0f8ff;

  width: 479px;

}


.buttons {

  display: flex;

  justify-content: space-between;

  margin-top: 20px;

  gap: 15px;

}


.mat-button1{

  border-width: 3px;

  border-style: solid;

  border-color: #91b692;

  background-color: #4ae14f;

  border-radius: 5px;

  max-width: 100%;

}
```

```css
.mat-button {

  border-width: 3px;

  border-style: solid;

  border-color: #967471;

  background-color: #ef3633;

  border-radius: 5px;

}


.btn {

  padding: 10px;

  font-weight: bold;

  border: none;

  border-radius: 5px;

  flex: 1;

  cursor: pointer;

}


.mat-button1:hover {

  background-color: #347636;

  color: white;

  transform: scale(0.9);

}
```

```css
}


.mat-button:hover {

  background-color: #ac2f2d;

  color:white;

  transform: scale(0.9);



}
```

Html

```html
<div class="profile-container">

  <div class="header">

    <h1>User Profile</h1>

  </div>



  <div class="profile-card">

    <div class="profile-image"><img
src="https://tse4.mm.bing.net/th/id/OIP.p_yVJpMf6hUETy-
glD0MQQHaLG?r=0&rs=1&pid=ImgDetMain&o=7&rm=3" alt=""
class="profile-image"/></div>



    <div class="profile-info">
```

```html
        <label>Name:</label>

        <input [(ngModel)]="user.name" type="text" />



        <label>Email:</label>

        <input [(ngModel)]="user.email" type="email" />



        <label>Phone:</label>

        <input [(ngModel)]="user.phone" type="text" />



        <label>Address:</label>

        <input [(ngModel)]="user.address" type="text" />



        <label>Bio:</label>

        <textarea [(ngModel)]="user.bio"></textarea>



        <div class="buttons">

          <button class="mat-button1"
(click)="onSave()">Save</button>

          <button class="mat-button"
(click)="onLogout()">Delet Account</button>

        </div>

    </div>
```

```
    </div>

</div>
```

```typescript
import { Component } from '@angular/core';


@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.css']
})
export class ProfileComponent {
  profile = {
    photo: '',
  }
  user = {
    name: 'John Doe',
    email: 'john@example.com',
    phone: '+1-555-1234',
    address: '123 Main St, Cityville',
    bio: 'Hello! I love coding and coffee.'
  };
```

```
  onSave(): void {

    alert('Profile saved successfully!');

  }



  onLogout(): void {

    alert('Logging out...');

  }

}
```
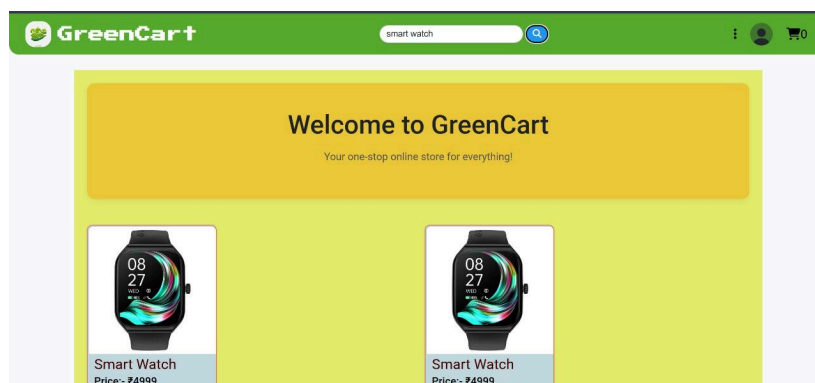
Outher snapes:-

Bcome a Seller

List Your Product Noww!!

Product Image

Choose File | img1.avif

Product Name

car

Price (₹)

60000000

Description

black car

Category

Others ⌄

Add Product

Backend

server.js:-

```javascript
const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

require('dotenv').config();



const productRoutes = require('./Routes/router');



const app = express();



const connectDB = require('./config/db');

connectDB();



// Middleware

app.use(cors());

app.use(express.json()); // Parses incoming JSON



// Routes

app.use('/api/products', productRoutes);



// MongoDB Connection
```

77

```js
mongoose.connect(process.env.MONGO_URI, {

  useNewUrlParser: true,

  useUnifiedTopology: true

})

.then(() => console.log('MongoDB Connected'))

.catch(err => console.log(err));


// Start Server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

router.js:-

```js
const express = require('express');

const router = express.Router();

const { getProducts, addProduct, deleteProduct } =
require('../controllers/productController');


// Get all products

router.get('/', getProducts);


// Add a product
```

```
router.post('/', addProduct);


// Delete a product by ID

router.delete('/:id', deleteProduct);



module.exports = router;
```

product.js:-

```
// models/Product.js

const mongoose = require('mongoose');



const productSchema = new mongoose.Schema({

  name: { type: String, required: true },

  price: { type: Number, required: true },

  image: { type: String, required: true },

  description: { type: String, required: true },

  category: { type: String, required: true } // changed from Category
to category

});
```

```
module.exports = mongoose.model('Product', productSchema);
```

productcontroller.js:-

```javascript
const Product = require('../modules/product');


// Get all products

const getProducts = async (req, res) => {

  try {

    const products = await Product.find();

    res.json(products);

  } catch (err) {

    res.status(500).json({ message: err.message });

  }

};


// Add new product

const addProduct = async (req, res) => {

  try {

    const { name, price, image, description, category } = req.body;


    // Input validation

    if (!name || !price || !image || !description || !category) {

      return res.status(400).json({ message: 'All fields are required.'
});
```

```javascript
  }

  const newProduct = new Product({ name, price, image, description,
category });

  const savedProduct = await newProduct.save();

  res.status(201).json(savedProduct);

} catch (err) {

  console.error('Error saving product:', err);

  res.status(500).json({ message: 'Server error' });

}

};


// Delete product

const deleteProduct = async (req, res) => {

try {

  await Product.findByIdAndDelete(req.params.id);

  res.json({ message: 'Product deleted' });

} catch (err) {

  res.status(500).json({ message: err.message });

}

};


module.exports = { getProducts, addProduct, deleteProduct };
```

db.js:-

```javascript
const mongoose = require('mongoose');


const connectDB = async () => {

  try {

    await mongoose.connect(process.env.MONGO_URI, {

      useNewUrlParser: true,

      useUnifiedTopology: true

    });

    console.log('MongoDB connected');

  } catch (err) {

    console.error(err);

    process.exit(1);

  }

};


module.exports = connectDB;
```

CONCLUSION

This project has been successfully implemented using modern web development technologies such as MongoDB, Angular, and other supporting tools. The use of MongoDB as a NoSQL database provided high scalability, flexibility, and performance, making it well-suited for handling complex data structures. Angular, being a robust frontend framework, contributed to creating a dynamic, responsive, and user-friendly interface.

The combination of these technologies resulted in a seamless and efficient system that meets user needs effectively. The system's modular design ensures maintainability and extensibility, allowing for future enhancements. Overall, the project demonstrates a practical application of the MEAN stack principles and highlights the benefits of modern full-stack development approaches in delivering scalable and interactive web solutions.

FUTURE SCOPE AND FURTHER ENHANCEMENTS

In future, we would like to continue enhancing this E-commerce website to make it more robust, scalable, and user-friendly. With advancements in technology and user expectations, we aim to integrate the following improvements:

Planned Enhancements:

1. **Mobile Application Integration**

   Develop Android and iOS apps using frameworks like Flutter or React Native to provide a seamless mobile shopping experience.

2. **AI-Based Recommendations**

   Implement machine learning algorithms to offer personalized product recommendations based on user behavior and purchase history.

3. **Advanced Search and Filtering**

   Improve the search functionality with autocomplete, filters, and natural language processing for better user experience.

4. **Real-time Chat Support**

   Add live chat or chatbot features using tools like Socket.io or Dialogflow for real-time customer assistance.

5. **Multi-Vendor Support**

   Enable third-party sellers to list and manage their products on the platform, making it a true marketplace.

6. **Improved Security Features**

   Add features like two-factor authentication, encrypted payment processing, and advanced role-based access control.

BIBLIOGRAPHY

Mongodb :- **https://docs.mongodb.com**

Nodejs :-https://nodejs.org/en/docs

Angular :-https://angular.io/docs

W3schools :-https://www.w3schools.com

Github :-https://github.com

Jwt :-https://jwt.io/introduction

Expressjs :-https://expressjs.com

Bcome a Seller

List Your Product Noww!!

Product Image

Choose File | img1.avif

Product Name

car

Price (₹)

60000000

Description

black car

Category

Others ⌄

Add Product

76

Backend

server.js:-

```javascript
const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

require('dotenv').config();



const productRoutes = require('./Routes/router');



const app = express();



const connectDB = require('./config/db');

connectDB();



// Middleware

app.use(cors());

app.use(express.json()); // Parses incoming JSON



// Routes

app.use('/api/products', productRoutes);



// MongoDB Connection
```