eCommerce Product Showcase using MERN Stack

Nithin Linga, Kartikeya Vinay Deepak Jakkinapalli, Revanth Ganta, Eeshaan Timmanapalli, Akanksha Singh

Chandigarh University, India

Corresponding author: Kartikeya Vinay Deepak Jakkinapalli, Email: jkartikeyavd2003@gmail.com

The MERN stack is a used technology, for building web applications. It consists of MongoDB, Express, React, and Node. MongoDB is a NoSQL database that can handle data structures with ease. Express simplifies the process of developing web apps and APIs using Node. React on the other hand is a JavaScript library that aids in creating user interfaces. Node effectively handles server-side logic. Serves as the runtime environment. The process of building a web project with MERN involves setting up the development environment de- signing a frontend using React connecting it with the backend and deploying the application. MERNs scalability and agility make it suitable, for applications, which is why an increasing number of developers are opting for it.

Keywords: MERN stack, MongoDB, Express.js, React.js, Node.js, HTML, CSS.

in: *Advancements in Communication and Systems*. Ed. by Ashish Kumar Tripathi and Vivek Shrivastava. Computing and Intelligent Systems, SCRS, India., 2023, pp. 253–272. DOI: https://doi.org/10.56155/978-81-955020-7-3-23

1. Introduction

The modern period has greatly benefited from web development since it is nearly always utilized everywhere, whether we are using a PC or a smartphone. The primary area that online development has the greatest impact on is shopping; these days, no one wants to go shopping; instead, they prefer to shop in between their everyday activities. This research paper is based on an e-commerce product showcase website that facilitates apparel shopping [1].

This online store mostly sells clothing rather than goods and other accessories [3]. We primarily used HTML, CSS, react.js or react for front-end development, node.js, and express.js or express for back-end development, and Mongo DB, which stores customer and admin data in JSON format, for back-end development.

Both frontend and backend development are required to establish a website; however, before we can work on the frontend, which is what users see, we must first create websites in HTML.HTML, sometimes referred to as "Hypertext Markup Language" is a markup language used to produce and organize material for websites. These websites provide text, photos, or other media with information about a certain product [2].

Once the webpage is created we need to incorporate “Cascading Style Sheets" (CSS) to enhance its appeal and structure. CSS allows us to manage aspects of the websites appearance, such, as background, spacing, placement, font selection and color schemes. In the phase we can add interactivity, to the web pages using JavaScript (JS). For this website we utilize React, a front end web development library based on JS that was developed by Facebook back in 2011.

Back-end web development begins once front-end web development is complete. Backend web development is done by web developers; it is not visible to customers. They ensure that there are no bugs and, if there are, they are fixed right away. Node.js is the run-time environment used for this, and express.js is used to work on node.js' instructions. We choose MongoDB to store the data since we need to make sure that the information that our customers require is not lost [5].

2. Features

Our website's primary features include the ability for users to register for free online. and to look for a particular product that interests them. He or she may place an online order using the "Cash on delivery," "Paypal," and "Card" payment options. The "Card" option accepts a variety of cards, including VISA, Master Card, and others. A few fields, including "Card Number," "Expiry Date," and "CVV," must be filled out; once these are done, the money will be taken out of the appropriate bank and the order will be validated. The sole authority to add, modify, or remove any product is held by the administrator [9]. Clients have the ability to update their personal data at any moment. Customers can order everything they want without repeatedly providing their billing information after they have logged into the system. The products on this website are arranged according to brands and categories. Simply dragging their cursor over the product image in the product details section allows customers to view any product in great detail. (see Figure 1)

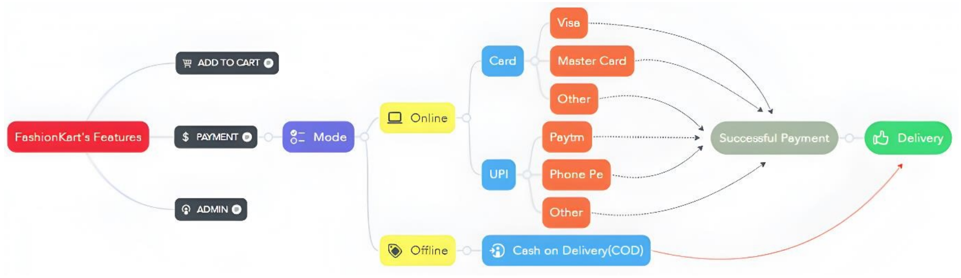


Figure 1. FlowChart

3. Related Work

The project was built on a B2C E-commerce web application with two sorts of users: administrators and users. Administrators are in charge of a wide range of administration duties, including the creation, modification, and deletion of goods from databases. On the other side, the user can peruse and examine the product's information before purchasing it by adding it to the shopping cart and completing the transaction. Some pages and web routes are available to the general public, while others are only accessible to administrators and logged-in users [1].

Full-stock developers are in higher demand than ever before. Full-stock builders are in higher demand than ever before. According to a Real research, the finest calls from the United States generate an incredible average profit of $ 110,770. Time Frame A Full Stack Developer is someone who can work on both the front and back ends of a dynamic Internet site or Internet-based apps. Web development is frequently based on the LAMP stack (Linux, Apache, MySQL, PHP, or Perl) and Java (Java EE, Spring), which incorporates some PC languages. JavaScript contributes to the improvement of the Internet by increasing the visibility of the MERN stack, allowing it to record every user aspect and server dimension [6].

A Node program operates in its own process with event looping, so no new thread is required for each activity. This contrasts with typical servers, which use a limited number of threads to handle requests. Node is a scalable, non-blocking, asynchronous, and event-driven engine. In general, Node libraries are built in a non-blocking fashion. Instead of pausing while waiting for a response, the application calls a request and then moves on to the next activity. When the request is finished, a callback function returns the results to the application. This enables the execution of several connections or requests to a server at the same time, which is useful when growing applications. Because MongoDB was designed to be utilized asynchronously, it is compatible with Node.js [5].

4. Models Used

B2C companies sell to customers directly [7]. Every purchase you make as a customer from clothing and home goods to entertainment in an internet store is a business-to-consumer transaction. When it comes to lower-value purchases, in particular, the decision-making process for a business-to-business (B2B) transaction is far shorter [8]. B2C companies often spend less on marketing to close a deal than their B2B counterparts, despite having a lower average order value and fewer recurring purchases due to this shorter sales cycle [8]. B2C encompasses both goods and services. B2C innovators have made direct marketing to their clients easier by utilizing technologies like native advertising, remarketing, and mobile apps [9]. (see Figure 2)

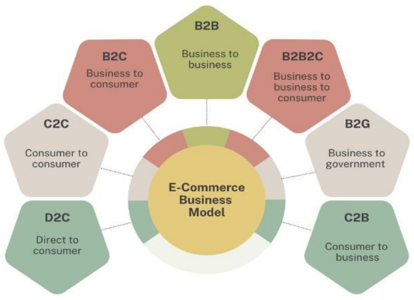


Figure 2. Business types

5. Planning

Our aim was to build a website that was easy to use, presented in a polished manner and had adequate beauty. for individuals of all ages to be its ultimate consumers. First, we had to define our objectives and divide the job into smaller, more doable tasks. An indicator of the percentage of work that is actually finished and a success story would be the benchmarks.

The following steps take place when planning the project

5.1 Use Case Model

Composing use cases, or tales of system usage, is a great way to comprehend and articulate needs. An end user who has allowed internet browsing registers and logs in to our website. uses the search function to locate products that interest him. Once the online copy of the bill is generated, it adds the items to the cart, where the products that are bought are placed and places the online order [4]. (see Figure 3)

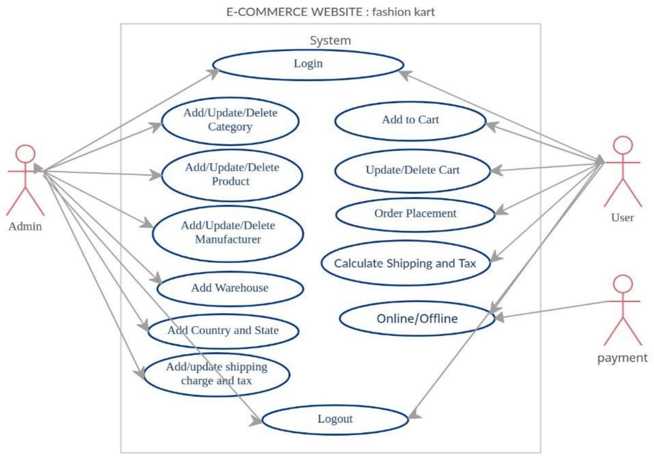

E-COMMERCE WEBSITE : fashion kart
System
Login
Add/Update/Delete Category
Add to Cart
Add/Update/Delete Product
Update/Delete Cart
Order Placement
Add/Update/Delete Manufacturer
Calculate Shipping and Tax
Add Warehouse
Online/Offline
Add Country and State
Add/update shipping charge and tax
Logout
Admin
User
payment

Figure 3. UseCase Model

5.2 Domain Modelling

We did the same as with the majority of web apps created with object-oriented programming (OOP). Therefore, we proceeded with the Object Oriented (OO) analysis. Which places an emphasis on locating and characterizing the concepts or objects within the issue domain. For instance, in our system, a product is an item [4].

5.3 Architectural Pattern

Our application uses an 'event-driven architecture'. In this architecture when an event occurs, for example when a login button is pressed, it performs the action [4]. (see Figure 4)

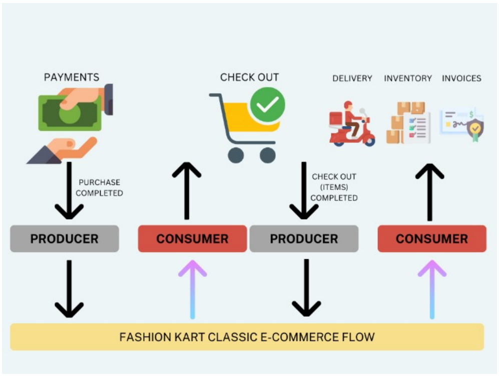


Figure 4. Architectural Pattern

6. Selection of Model

Differences between MERN and MEAN
In the MERN stack, React.js takes the spotlight. React.js is a JavaScript library known for its ability to create user interfaces. It follows a component-based architecture and stands out for its virtual DOM, which enhances rendering performance. On the other hand, in the MEAN stack, Angular is the frontrunner. Angular is a comprehensive front-end framework specially designed for building dynamic and single-page applications. It comes bundled with a set of conventions, making it a slightly more opinionated choice. Next, is the learning curve. When it comes to React.js, it is often considered more beginner-friendly. Its component-based nature and the extensive support from its vibrant community make it easier for beginners to get started. In contrast, Angular has a steeper learning curve. This is mainly because Angular is a fullfledged framework with more in-built features and established conventions.

Moving on to flexibility. React.js offers developers greater flexibility by allowing them to choose additional libraries or tools based on their project requirements. It is frequently described as more lightweight and adaptable. On the other hand, Angular's opinionated nature may limit flexibility in certain aspects. While this can be beneficial in terms of maintaining consistency, it may prove to be a drawback for specific use cases.

Reasons why we preferred the MERN Stack
Firstly, the component-based architecture of React.js promotes a modular and reusable code structure. This is highly appealing to developers and organizations looking for maintainable and scalable code. Additionally, the React.js community is substantial and highly active, resulting in a robust ecosystem of

libraries, tools, and resources. This proves advantageous for developers in search of solutions, troubleshooting guidance, and community support. Another key advantage of React.js is its virtual DOM, which enables rendering optimization by updating only the parts of the real DOM that have changed. This can lead to improved performance compared to updating the entire DOM. Lastly, by leveraging JavaScript across the entire stack, MERN allows developers to combine Node.js for the backend and React.js for the front end seamlessly. This fosters a consistent and seamless development experience.

It is important to note that the choice between MERN and MEAN ultimately depends on the specific needs of the project, the expertise of the development team, and personal preferences. While MERN is often preferred for its flexibility and perceived ease of learning, MEAN may be a better fit for projects where an opinionated framework like Angular brings advantages or where a comprehensive, integrated solution is preferred.

7. Technology Used

This e-commerce website is made using MERN stack where M-MongoDB, E-Ex- press.js, R-React.js, and N- Node.js [3]. For creating an e-commerce product showcase website, we need 3 things, frontend web development, backend web development, and a database to store the data of the users so that, the users can access the data later on, this information may have the login information, payment information, which will not be shared to any other users, and the products that are frequently viewed by the users [6]. (see Figure 5)

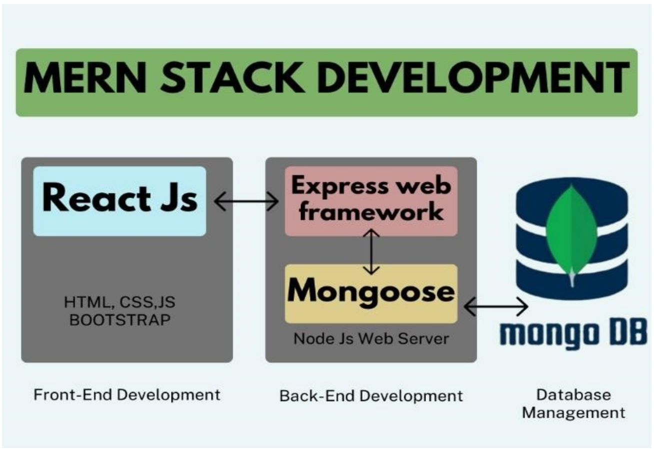


Figure 5. MERN Stack Development Flow

So, in order to build a good user interface, we use three languages

HTML (Hypertext Markup Language)

The first step in creating a user-interactive front end, the first step is creating web pages using HTML, sometimes referred to as "Hypertext Markup Language," which is a markup language used to produce and organize material for websites. These websites provide text, photos, or other media with information about a certain product [1]. These web pages are further modified to be colorful and user-interactive in the next 2 steps. The general structure of HTML is

<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<body>
</html>
Where <head></head> is the part that is visible on the website tab.
<body></body> part is the information that is present inside the webpage. Some of the basic operations that can be performed using HTML are

CSS (Cascading Style Sheets)

After successfully creating web pages using HTML, we need to make the pages user- friendly and colorful and for this, we need to use Cascading Style Sheets, also known as CSS [9]. This is the second step, in which we need to beautify the pages so that the customers or the users find the website to their liking. In this step, we add font colors, In order to style a HTML file we need to link the CSS file to the HTML file using <link rel='stylesheet' href='style.css'>

After linking the css file we can perform functions such as adding background, ani- mations, changing font family, font color; creating a shadow, creating navigation bars, etc.,.

React.js

Using JavaScript, or simply JS, we can make the web pages interactive in the next phase. React is what we utilize for this website. React is a JS-based User Interactive development library used in front-end web development, which was created by Face- book in 2011 [1].

We created an index.js file in which we will import the React, ReactDOM, css file, App.js file which imports other javascript files and their paths such as 'login', 'Home', 'About', 'PageNotFound', 'Contact' and many other javascript files..

As we know that react is an open-source library so, in order to install react into your system first you need to install Nodejs in your system. Nodejs is used to run the javascript at client side i.e. in your system. After installing nodejs you need the Node Package Manager (NPM). The command to install react is npx create-next-app@latest after installing react app in your specified directory if you run npm start to run the react app in your browser in which pre-written code displays in your browser. (see Figure 6)

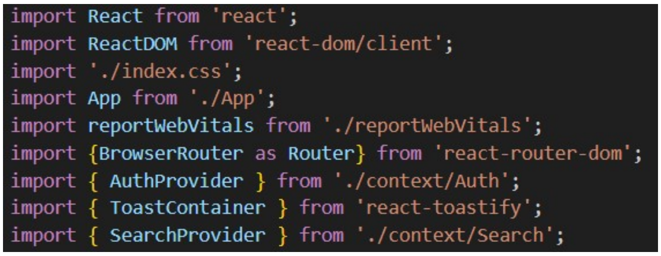
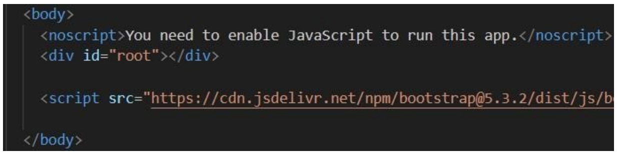

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import {BrowserRouter as Router} from 'react-router-dom';
import { AuthProvider } from './context/Auth';
import { ToastContainer } from 'react-toastify';
import { SearchProvider } from './context/Search';
```

Figure 6. importing js files into index.js

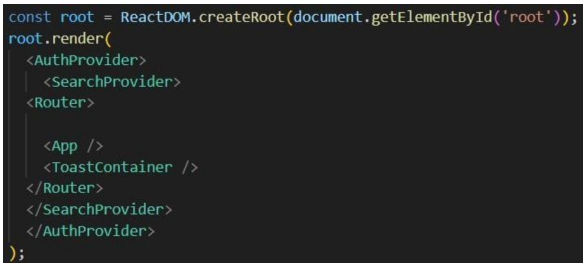
Index.js file is responsible for routing(react-router-dom), for authentication between user and admin and third party npm packages. And also index.js is the root file at client side in which the development starts from index file. Using the ReactDOM library, we will render the user interface and access the HTML file, in which we already created a div element with id as 'root'. (see Figure 7)



```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/b

</body>
```

Figure 7. creating div element with id 'root'

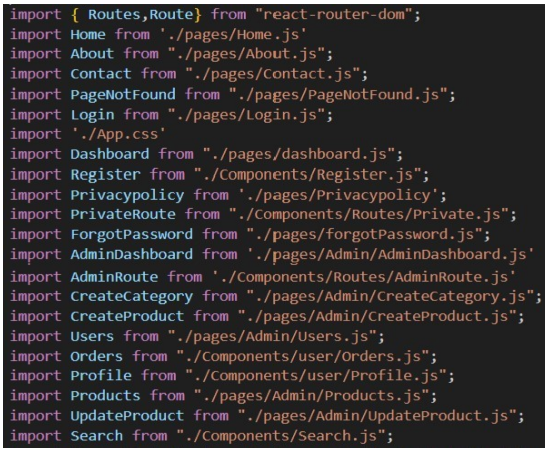
In Index.js we will create a root in which we will render the other javascript files directly into the html file which is displayed to the user. Whatever the changes we made in the different file all these should be rendered through index.js file. In a way we can say that index file is the heart for client side. (see Figure 8)



```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <AuthProvider>
    <SearchProvider>
  <Router>

    <App />
    <ToastContainer />
  </Router>
  </SearchProvider>
  </AuthProvider>
);
```

Figure 8. creating a root using DOM in index.js

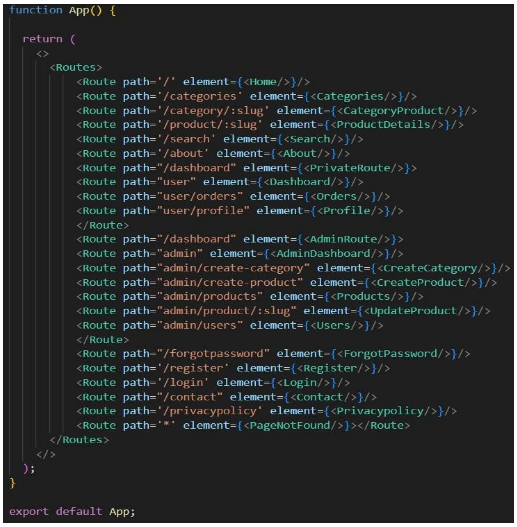
App.js file is used for routing between different pages. First, we have to import that component with path and in the function we can use that component for routing. To implement routing we need to import package named react-router-dom. (see Figure 9)



```
import { Routes,Route} from "react-router-dom";
import Home from './pages/Home.js'
import About from "./pages/About.js";
import Contact from "./pages/Contact.js";
import PageNotFound from "./pages/PageNotFound.js";
import Login from "./pages/Login.js";
import './App.css'
import Dashboard from "./pages/dashboard.js";
import Register from "./Components/Register.js";
import Privacypolicy from './pages/Privacypolicy';
import PrivateRoute from "./Components/Routes/Private.js";
import ForgotPassword from "./pages/forgotPassword.js";
import AdminDashboard from './pages/Admin/AdminDashboard.js'
import AdminRoute from './Components/Routes/AdminRoute.js'
import CreateCategory from "./pages/Admin/CreateCategory.js";
import CreateProduct from "./pages/Admin/CreateProduct.js";
import Users from "./pages/Admin/Users.js";
import Orders from "./Components/user/Orders.js";
import Profile from "./Components/user/Profile.js";
import Products from "./pages/Admin/Products.js";
import UpdateProduct from "./pages/Admin/UpdateProduct.js";
import Search from "./Components/Search.js";
```

Figure 9. importing other js files into App.js

This App.js file also consists of the path or the routes of the javascript files (see Figure 10)

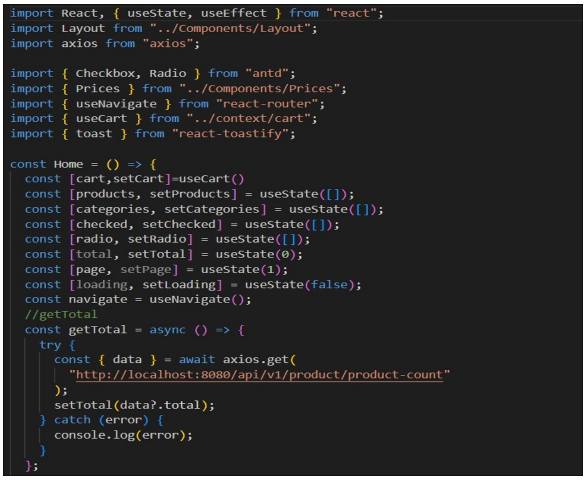


```
function App() {

  return (
    <>
      <Routes>
          <Route path='/' element={<Home/>}/>
          <Route path='/categories' element={<Categories/>}/>
          <Route path='/category/:slug' element={<CategoryProduct/>}/>
          <Route path='/product/:slug' element={<ProductDetails/>}/>
          <Route path='/search' element={<Search/>}/>
          <Route path='/about' element={<About/>}/>
          <Route path="/dashboard" element={<PrivateRoute/>}>
          <Route path="user" element={<Dashboard/>}/>
          <Route path="user/orders" element={<Orders/>}/>
          <Route path="user/profile" element={<Profile/>}/>
          </Route>
          <Route path="/dashboard" element={<AdminRoute/>}>
          <Route path="admin" element={<AdminDashboard/>}/>
          <Route path="admin/create-category" element={<CreateCategory/>}/>
          <Route path="admin/create-product" element={<CreateProduct/>}/>
          <Route path="admin/products" element={<Products/>}/>
          <Route path="admin/product/:slug" element={<UpdateProduct/>}/>
          <Route path="admin/users" element={<Users/>}/>
          </Route>
          <Route path="/forgotpassword" element={<ForgotPassword/>}/>
          <Route path='/register' element={<Register/>}/>
          <Route path='/login' element={<Login/>}/>
          <Route path="/contact" element={<Contact/>}/>
          <Route path='/privacypolicy' element={<Privacypolicy/>}/>
          <Route path='*' element={<PageNotFound/>}></Route>
      </Routes>
    </>
  );
}

export default App;
```

Figure 10. App.js

As you can see in the above figure each component is given a path, it is given in such a way that the specified component should render corresponding to the path provided.

Every element that is specified in the App.js file is its own JavaScript file with its own functions [9] For example, if we take Homepage element, its function is to renders products in an order. Every element as functionality. (see Figure 11)



```
import React, { useState, useEffect } from "react";
import Layout from "../Components/Layout";
import axios from "axios";

import { Checkbox, Radio } from "antd";
import { Prices } from "../Components/Prices";
import { useNavigate } from "react-router";
import { useCart } from "../context/cart";
import { toast } from "react-toastify";

const Home = () => {
  const [cart,setCart]=useCart()
  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState([]);
  const [checked, setChecked] = useState([]);
  const [radio, setRadio] = useState([]);
  const [total, setTotal] = useState(0);
  const [page, setPage] = useState(1);
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();
  //getTotal
  const getTotal = async () => {
    try {
      const { data } = await axios.get(
        "http://localhost:8080/api/v1/product/product-count"
      );
      setTotal(data?.total);
    } catch (error) {
      console.log(error);
    }
  };
```

Figure 11. Home page

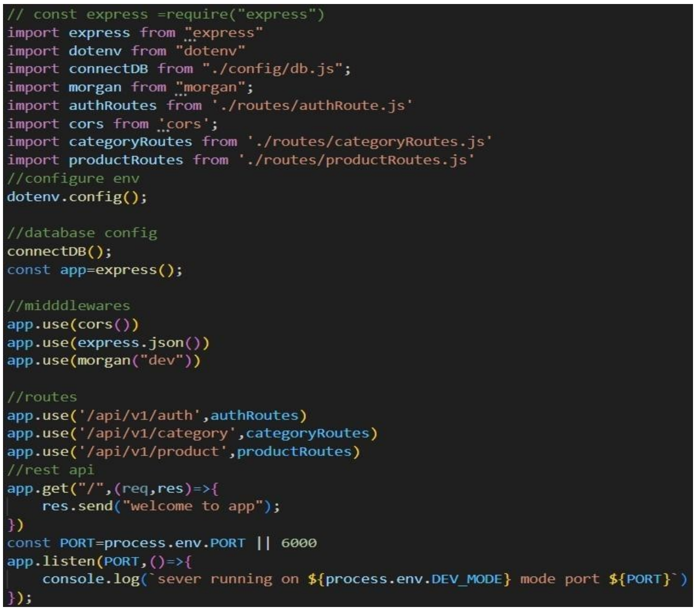
Home page is the landing page of our application in which it consists of navigation bar, products, radio buttons for price filtering and check boxes for category filtering. Using restfull api we fetch the data about the products which are mainly name of the product ,description, price and each product which is displayed in the home page consist of add to cart and more details button. In home.js file we use react hook which is useState for various functionalities like radio buttons, check boxes, no of items on Cart icon in navigation bar. useState is used for storing the previous state and current state if any object.

In this way each and every element has its own javascript file with its own function.
In order to ensure that there are no issues in the server side and there are no issues for the users we need two major backbones for development of project

Node.js
The benefit with node.js is that it can be used to write the code outside the website. It is modified on the Chrome's V8 JavaScript Engine.

Express.js
It is a simple, quick online framework for Node.js. The Express framework is intended for the development of reliable APIs and web applications. It has a simple structure, quick speed, and a wealth of functionality that can be added as plugins.

First we need to create a server.js file in order to call the APIs (see Figure 12)



```
// const express =require("express")
import express from "express"
import dotenv from "dotenv"
import connectDB from "./config/db.js";
import morgan from "morgan";
import authRoutes from './routes/authRoute.js'
import cors from 'cors';
import categoryRoutes from './routes/categoryRoutes.js'
import productRoutes from './routes/productRoutes.js'
//configure env
dotenv.config();

//database config
connectDB();
const app=express();

//midddlewares
app.use(cors())
app.use(express.json())
app.use(morgan("dev"))

//routes
app.use('/api/v1/auth',authRoutes)
app.use('/api/v1/category',categoryRoutes)
app.use('/api/v1/product',productRoutes)
//rest api
app.get("/",(req,res)=>{
    res.send("welcome to app");
})
const PORT=process.env.PORT || 6000
app.listen(PORT,()=>{
    console.log(`sever running on ${process.env.DEV_MODE} mode port ${PORT}`)
});
```

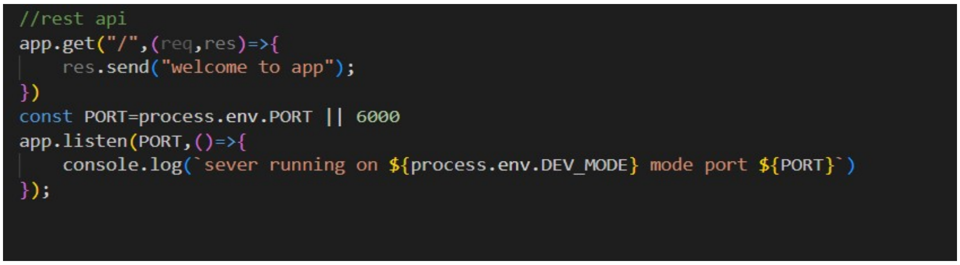
Figure 12. Server.js

Express js gives a functionality to simplify the API calling. To use express we need to import it in the server.js file and further we have to specify the API routes for individual routes like auth route for authentication, category route for categories and product routes for products.

Creating Rest API

Rest API is an interface which is used to communicate or exchange the information between client and server.

Exchange information means to input the data into the database, get the data from the database and update the data in the database. API calls are used to perform these actions. (see Figure 13)



```
//rest api
app.get("/",(req,res)=>{
    res.send("welcome to app");
})
const PORT=process.env.PORT || 6000
app.listen(PORT,()=>{
    console.log(`sever running on ${process.env.DEV_MODE} mode port ${PORT}`)
});
```

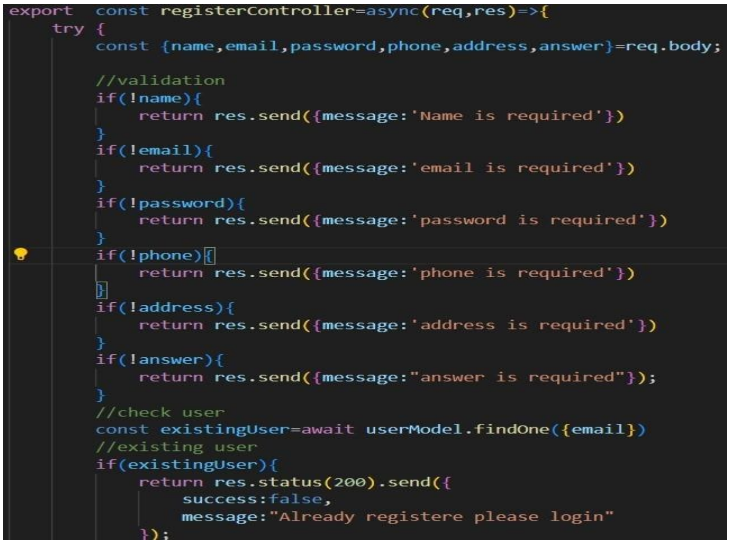
Figure 13. Rest API

Controllers

We have mainly three controllers which are auth controller, product controller and category controller which are used to handle the data

Auth Controller

Auth controller is used to handle user information and check for the redundant data and store the user data into the database in the form of documents. As you can see in below figure it is checking for every input field and checking if there is an existing user in the database if there is an existing user it produces an error if not then the register Controller successfully creates a new user. (see Figure 14)

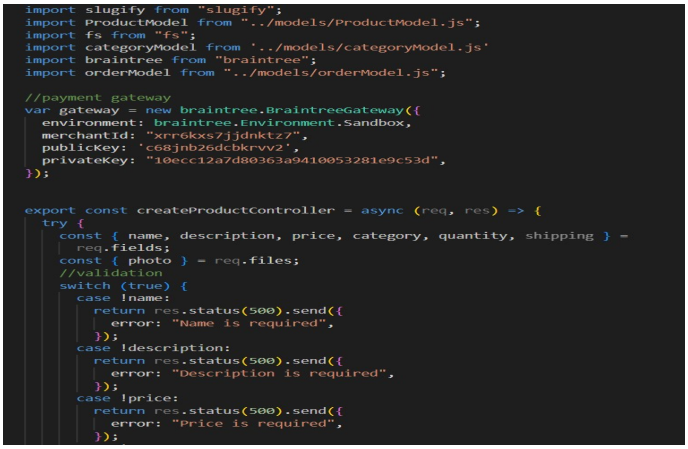


```
export  const registerController=async(req,res)=>{
    try {
        const {name,email,password,phone,address,answer}=req.body;

        //validation
        if(!name){
            return res.send({message:'Name is required'})
        }
        if(!email){
            return res.send({message:'email is required'})
        }
        if(!password){
            return res.send({message:'password is required'})
        }
        if(!phone){
            return res.send({message:'phone is required'})
        }
        if(!address){
            return res.send({message:'address is required'})
        }
        if(!answer){
            return res.send({message:"answer is required"});
        }
        //check user
        const existingUser=await userModel.findOne({email})
        //existing user
        if(existingUser){
            return res.status(200).send({
                success:false,
                message:"Already registere please login"
            });
```

Figure 14. authController.js

Product Controller
Product controller is used to manage the products by filtering the products like rendering only one product, rendering similar products and rendering the products according to the search result. The main functionality of product controller is to manage products in the database and render on the page.

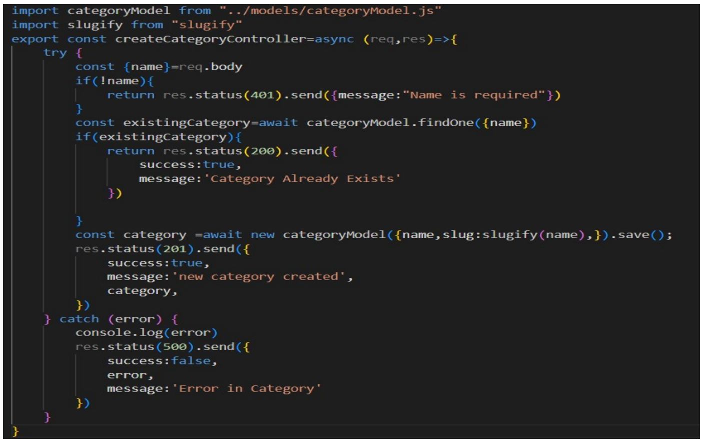
Product controller also contain payment gateway. In this project only card through payment is allowed. Card can be any visa for example VISA, MASTERCARD, RUPAY etc... (see Figure 15)



```javascript
import slugify from "slugify";
import ProductModel from "../models/ProductModel.js";
import fs from "fs";
import categoryModel from '../models/categoryModel.js'
import braintree from "braintree";
import orderModel from "../models/orderModel.js";

//payment gateway
var gateway = new braintree.BraintreeGateway({
  environment: braintree.Environment.Sandbox,
  merchantId: "xrr6kxs7jjdnktz7",
  publicKey: 'c68jnb26dcbkrvv2',
  privateKey: "10ecc12a7d80363a94100532 81e9c53d",
});


export const createProductController = async (req, res) => {
  try {
    const { name, description, price, category, quantity, shipping } =
      req.fields;
    const { photo } = req.files;
    //validation
    switch (true) {
      case !name:
        return res.status(500).send({
          error: "Name is required",
        });
      case !description:
        return res.status(500).send({
          error: "Description is required",
        });
      case !price:
        return res.status(500).send({
          error: "Price is required",
        });
```

Figure 15. productController.js

Category Controller
Category controller is used to manage category which means to create a category, delete a category and also rendering the products based on the category. Creating a category or deleting a category is done by only admin. A user can only fetch the products based on categories. This is where auth controller and middleware comes into, it differentiates the user and admin and provide the privileges to the individual users. The user can do the actions according to the privileges they have. (see Figure 16)

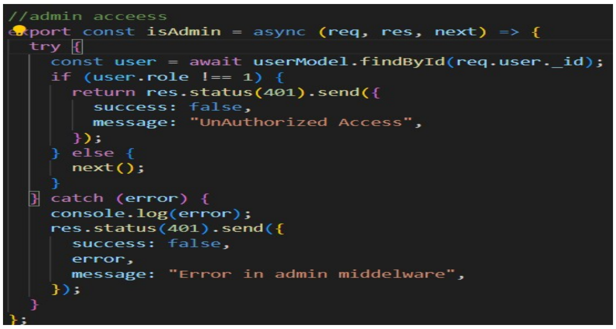


```javascript
import categoryModel from "../models/categoryModel.js"
import slugify from "slugify"
export const createCategoryController=async (req,res)=>{
    try {
        const {name}=req.body
        if(!name){
            return res.status(401).send({message:"Name is required"})
        }
        const existingCategory=await categoryModel.findOne({name})
        if(existingCategory){
            return res.status(200).send({
                success:true,
                message:'Category Already Exists'
            })

        }
        const category =await new categoryModel({name,slug:slugify(name),}).save();
        res.status(201).send({
            success:true,
            message:'new category created',
            category,
        })
    } catch (error) {
        console.log(error)
        res.status(500).send({
            success:false,
            error,
            message:'Error in Category'
        })
    }
}
```

Figure 16. categoryController.js

Middleware

Middleware acts as a bridge between the client side and database to differentiate between user and admin. It means to allow only admin to enter protected routes which are admin dashboard, creating a new product, creating a new category which should only accessible by admin. It every check for the user authorization For security reasons. (see Figure 17)



```
//admin access
export const isAdmin = async (req, res, next) => {
  try {
    const user = await userModel.findById(req.user._id);
    if (user.role !== 1) {
      return res.status(401).send({
        success: false,
        message: "UnAuthorized Access",
      });
    } else {
      next();
    }
  } catch (error) {
    console.log(error);
    res.status(401).send({
      success: false,
      error,
      message: "Error in admin middelware",
    });
  }
};
```

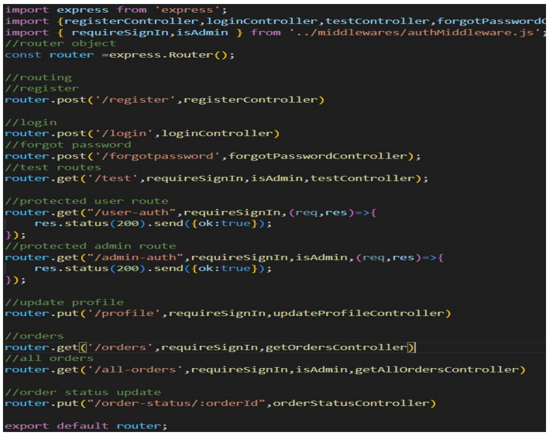
Figure 17. authMiddleware.js

Routes

Routes are the paths that are assigned with some privileges and controller to handle the data as well as to check for the authorization. These are also used to connect the front-end with backend and also get data from client and manage the data through controllers. We have mainly three types of routes they are,

Auth Routes

These routes are used in registering the new user, allowing the admin to access the admin dashboard using is Admin controller and requireSignIn controllers from auth controllers. It is also used to allow only registered users to place order and allow non-registered and registered user to view products and its details. (see Figure 18)

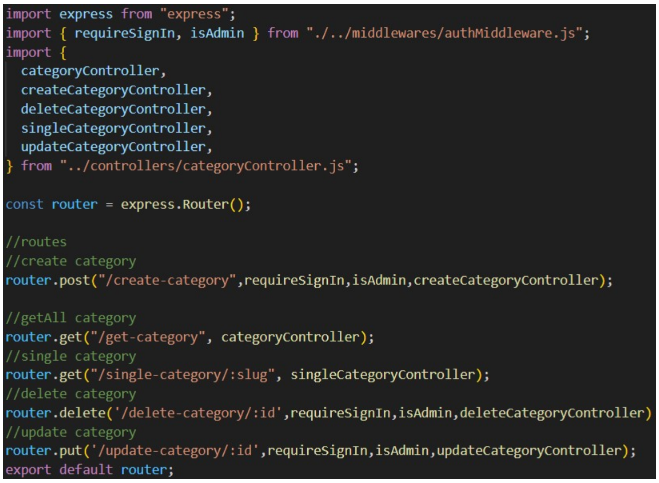


```
import express from 'express';
import {registerController,loginController,testController,forgotPasswordC
import { requireSignIn,isAdmin } from '../middlewares/authMiddleware.js';
//router object
const router =express.Router();

//routing
//register
router.post('/register',registerController)

//login
router.post('/login',loginController)
//forgot password
router.post('/forgotpassword',forgotPasswordController);
//test routes
router.get('/test',requireSignIn,isAdmin,testController);

//protected user route
router.get("/user-auth",requireSignIn,(req,res)=>{
    res.status(200).send({ok:true});
});
//protected admin route
router.get("/admin-auth",requireSignIn,isAdmin,(req,res)=>{
    res.status(200).send({ok:true});
});

//update profile
router.put('/profile',requireSignIn,updateProfileController)

//orders
router.get('/orders',requireSignIn,getOrdersController)
//all orders
router.get('/all-orders',requireSignIn,isAdmin,getAllOrdersController)

//order status update
router.put("/order-status/:orderId",orderStatusController)

export default router;
```

Figure 18. authRoutes.js

Category Routes

These routes are used assigning the controllers based on the requirement for the particular path, for example, create-category requires isAdmin controller, create Category Controller and requireSignIn controller which is mentioned in Category controller file. (see Figure 19)

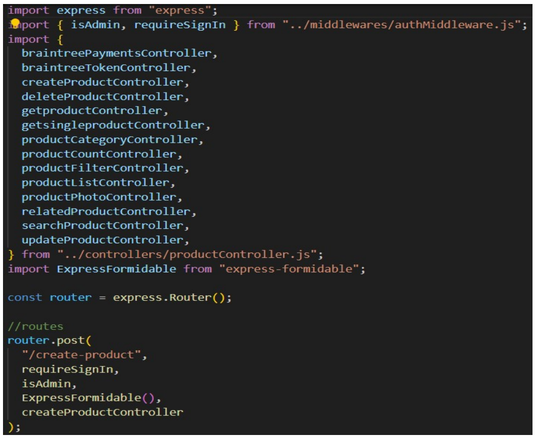


```javascript
import express from "express";
import { requireSignIn, isAdmin } from "./../middlewares/authMiddleware.js";
import {
  categoryController,
  createCategoryController,
  deleteCategoryController,
  singleCategoryController,
  updateCategoryController,
} from "../controllers/categoryController.js";

const router = express.Router();

//routes
//create category
router.post("/create-category",requireSignIn,isAdmin,createCategoryController);

//getAll category
router.get("/get-category", categoryController);
//single category
router.get("/single-category/:slug", singleCategoryController);
//delete category
router.delete('/delete-category/:id',requireSignIn,isAdmin,deleteCategoryController)
//update category
router.put('/update-category/:id',requireSignIn,isAdmin,updateCategoryController);
export default router;
```

Figure 19. categoryRoutes.js

Product Routes

These routes are used to assign the controllers based on the path that is for example, product-filter path requires product Filter Controller and search needs search controller which is written in product controller. (see Figure 20)



```javascript
import express from "express";
import { isAdmin, requireSignIn } from "../middlewares/authMiddleware.js";
import {
  braintreePaymentsController,
  braintreeTokenController,
  createProductController,
  deleteProductController,
  getproductController,
  getsingleproductController,
  productCategoryController,
  productCountController,
  productFilterController,
  productListController,
  productPhotoController,
  relatedProductController,
  searchProductController,
  updateProductController,
} from "../controllers/productController.js";
import ExpressFormidable from "express-formidable";

const router = express.Router();

//routes
router.post(
  "/create-product",
  requireSignIn,
  isAdmin,
  ExpressFormidable(),
  createProductController
);
```

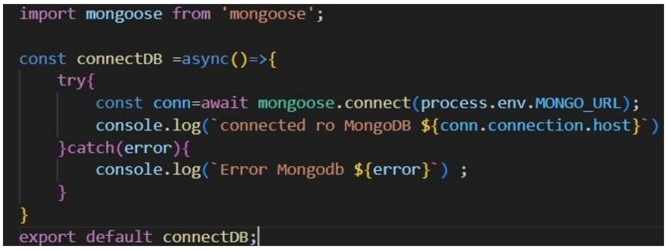
Figure 20. productRoutes.js

Database

MongoDB

MongoDB is basically a NoSQL database that stores the data from the website in the form of documents. Using routes we assign controllers and through controller we get the data and then we insert those data into the database through API calls. As we know POST-to send the data, GET-to get the data, PUT-to update the data and DELETE-to delete the data.

Below figure shows the database connection using the user which we have provided in the environmental variable in the server side. (see Figure 21)

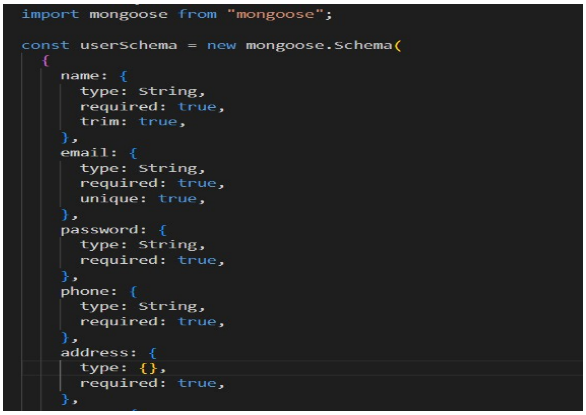


```
import mongoose from 'mongoose';

const connectDB =async()=>{
    try{
        const conn=await mongoose.connect(process.env.MONGO_URL);
        console.log(`connected ro MongoDB ${conn.connection.host}`)
    }catch(error){
        console.log(`Error Mongodb ${error}`) ;
    }
}
export default connectDB;
```

Figure 21. database(db.js)

User Model

User model is used to the details of every user including the admin details such as name, email Id, password, address and answer for password reset in future. Usermodel is imported in every controller which involves in managing the data about the user. (see Figure 22)

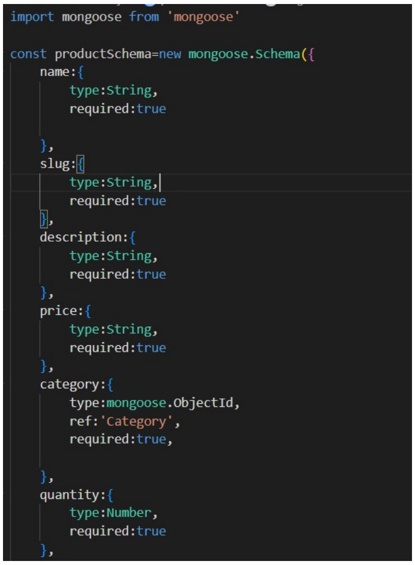


```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
      trim: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    phone: {
      type: String,
      required: true,
    },
    address: {
      type: {},
      required: true,
    },
```

Figure 22. userModel.js

Product Model

Product model is used to store the details about the project like name, price, description, type of category, image of the of product and no of products available. (see Figure 23)



```
import mongoose from 'mongoose'

const productSchema=new mongoose.Schema({
    name:{
        type:String,
        required:true

    },
    slug:{
        type:String,
        required:true
    },
    description:{
        type:String,
        required:true
    },
    price:{
        type:String,
        required:true
    },
    category:{
        type:mongoose.ObjectId,
        ref:'Category',
        required:true,

    },
    quantity:{
        type:Number,
        required:true
    },
```

Figure 23. productModel.js

8. Results

This phase describes how the product works in order of different steps and what admin can do and what freedom does the user have while operating the website.

Home Page

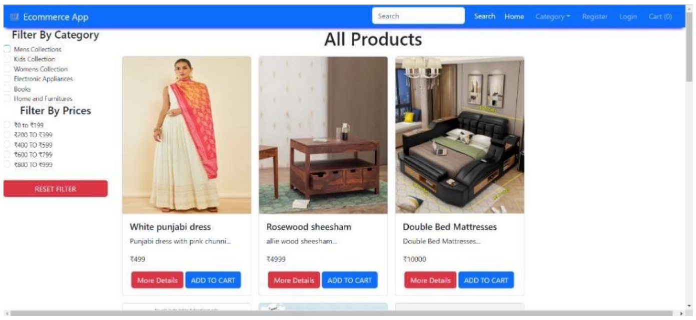


Figure 24. Home page

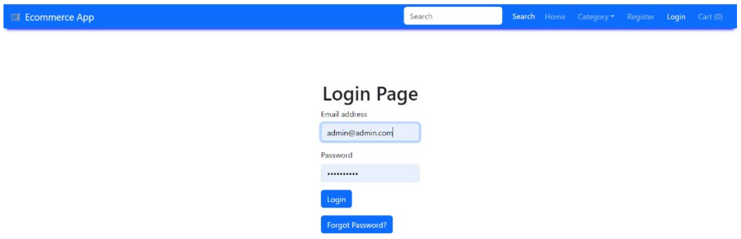
This is the landing page where we can see all the products are rendered in rows and columns.

To access this page the user need not be registered any user if he is authorized or unauthorized can access this page. (see Figure 24)

Login Page

In this login page, only the Verified admin and a verified user can log in through this page and the admin and user are differentiated at the backend. After logging in the user gets the specified rights like admin and user rights.

The login can be done in two ways, by user and by admin User and Admin login: (see Figure 25)



Login Page

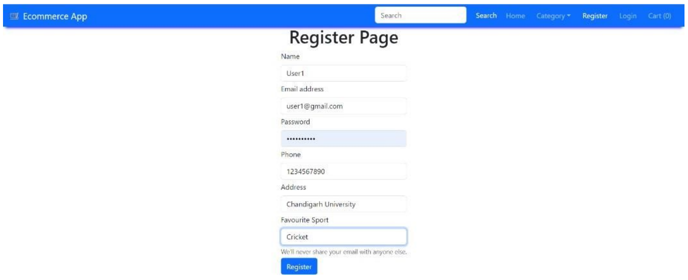
Email address: admin@admin.com

Password: ••••••••••

Figure 25. User and Admin Login

Registration Page

Anyone can register on our website using this registration page by giving the required details like name, email, password, phone number, and favorite sport for resetting the password. (see Figure 26)



Register Page

Name: User1

Email address: user1@gmail.com

Password: ••••••••••

Phone: 1234567890

Address: Chandigarh University

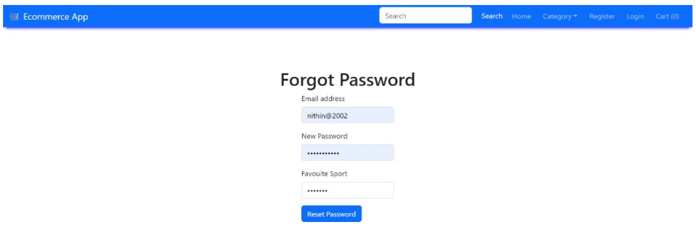
Favourite Sport: Cricket

We'll never share your email with anyone else.

Figure 26. Registration page

Forget Password

On this Page, the user or admin can change their password by providing the email address, new password, and favorite which was asked at the start. (see Figure 27)



Forgot Password

Email address: nithin@2002

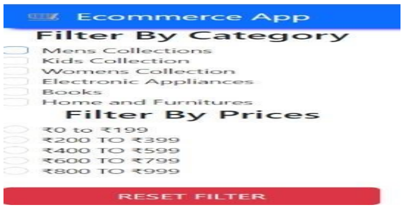
New Password: •••••••••••

Favouite Sport: •••••••

Figure 27. Forget Password Viewing

Products Based on Filters

We can filter the products by prices and categories. As we can see below when clicking on a particular category or particular price the related products are rendered according to their price and categories. We created filter by category and by prices: (see Figure 28)



Ecommerce App

Filter By Category

Mens Collections
Kids Collection
Womens Collection
Electronic Appliances
Books
Home and Furnitures

Filter By Prices

₹0 to ₹199
₹200 TO ₹399
₹400 TO ₹599
₹600 TO ₹799
₹800 TO ₹999

RESET FILTER

Figure 28. Filter types

By Category

As below figure shows only mens collection as we have clicked the mens collection checkbox. (see Figure 29)

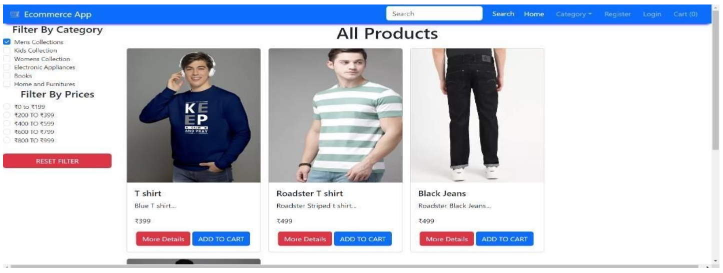


Figure 29. filtering by categories

By Prices

Below figure Renders the products between 200 to 399 rupees as we have clicked the radio button which consists 200-399. (see Figure 30)

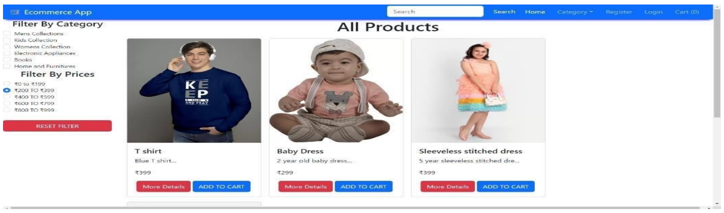
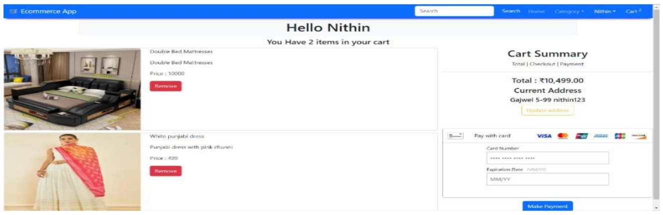


Figure 30. filtering by prices

Add to Cart and Payment Gateway

When you add products to the cart they are added to this page and rendered them a first come first basis on the hand side there will be the total price and user address which was given at the start (see Figure 31). The Payment option is a debit or credit card of any bank account and it offers seamless transactions.



Hello Nithin

You Have 2 items in your cart

Double Bed Mattresses

Double Bed Mattresses

Price : 10000

White punjabi dress

Punjabi dress with pink chunni

Price : 499

Cart Summary

Total | Checkout | Payment

Total : ₹10,499.00

Current Address

Gajwel 5-99 nithin123

Card Number

**** **** **** ****

Expiration Date (MM/YY)

MM/YY

Figure 31. payment

Admin Access

Since this website is based on the admin, the admin is the only one who can create products, update products, categorize the product Create Product: (see Figure 32)

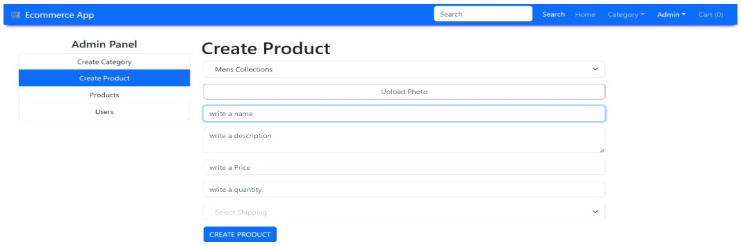
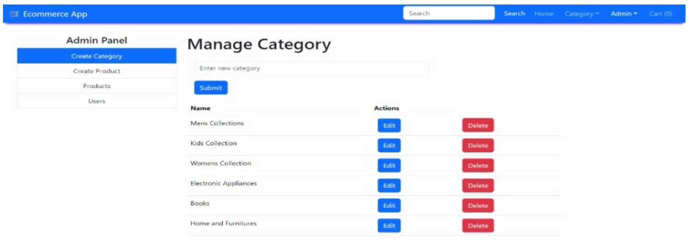


Figure 32. create product

Manage Categories

Only the admin has the right to add products or manage categories the below figures show how the admin can create/add a product by providing the required details and managing the categories which include adding, deleting, and renaming of categories. (see Figure 33)



Name	Actions	
Mens Collections	Edit	Delete
Kids Collection	Edit	Delete
Womens Collection	Edit	Delete
Electronic Appliances	Edit	Delete
Books	Edit	Delete
Home and Furnitures	Edit	Delete

Figure 33. Categories

The above screenshots provided are the outcome of our website, firstly the user land on home page (figure (25), any user can view products but in order to place an order he/she needs to be authorized. Our website consists of two users one is admin and other is user where user can only view products, place the order and check the status of orders, where as an admin can manage users, products, categories and status of orders. A user can register by providing the required details and can login and needs to set an answer for resetting password in future. After logging in successfully we will be redirected to home page, in home page we can see products in which each product contain two buttons one is for adding the product to cart and the other is to view more details about the product. In figure

30 and 31 we see that the products are filtered based on categories and prices respectively. After that admin dashboard has three functionalities first Create category for creating new category, then Create Product for adding new products and final orders which is used to update the status of every order. Then last and final step that is Payment gateway in our website we accept debit and credit cards for placing orders, after making the payment the user is redirected to orders page. We made sure throughout the project the website should run smoothly without any interruption to the user and admin.

9. Conclusion

The main goal is to build an online application for e-commerce electronics item sales using front-end development. E-commerce managed to persevere and generate a substantial amount of transactions in spite of the fact that the commodities and stock markets were in free collapse. Adding new methods and approaches to a transaction is also crucial. Bringing the kindness of a person or a state to the world through the huge E-Commerce available on the Internet is truly far greater [8]. Our project is merely a side endeavour to assist those who must oversee their project activity. Additional user- friendly coding methods have also been developed. Your demands will all be success- fully met by this package. The e-commerce industry as a whole is anticipated to flourish in India. Nonetheless, the global internet industry is growing rapidly [9]. It's official: in the near future, e-commerce will be the main way to advertise goods and services. In- deed, ecommerce has become more and more common in our society. Future success will go to businesses which embrace e-commerce and make sufficient investments in its expansion. E-commerce is a fully fledged economic endeavour, not just a technical problem. Businesses who use it as justification to overhaul their operations are likely to pick up on it. Additionally, e-commerce could be a helpful tool that lets customers communicate with companies and enterprises globally.

References

[1] Thi Thu Hien Tran, "THE DEVELOPMENT OF AN ECOMMERCE WEB APPLICATION USING MERN STACK", Bachelor’s Thesis ,Turku University of Ap- plied Sciences Information and Communications Technology, 2022

[2] Hung Viet Nguyen, "End-to-end E-commerce web application, a modern approach using MERN stack", Metropolia University of Applied Sciences, Bachelor’s Thesis, 21 May 2020

[3] Monika, Mehra, Manish Kumar, Anjali Maurya, Charu Sharma and Shanu, "MERN Stack Web Development", Annals of R.S.C.B., 08 May 2021

[4] Syed Emdad Ullah, Tania Alauddin and Hasan U. Zaman, "Developing an E-Commerce Website", IEEE, 2016

[5] Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate, "MERN: A Full-Stack Development", International Journal for Research in Applied Sci- ence & Engineering Technology (IJRASET), 10 Jan 2022

[6] Mrunmayee Vaibhav Kulkarni, "Social Media Web Application using MERN", International Research Journal of Engineering and Technology (IRJET), 02 Feb 2022

[7] Akarsh Shrivas, Aniket Pawar, Pratham Mishra, Prof. Satish Chadokar, "E-Commerce Website Using MERN Stack" ,IJIRMPS , May-June 2023

[8] Md Touhidul Islam, "Building an End-to-End E-commerce solution Designing and Imple- menting a Full-stack E-commerce website", BACHELOR’S THESIS, May 2023, Tampere University of Applied Sciences

[9] Avnish Kumar Sharma, BIG BUY(E- Commerce website) by using Frontend Web Devel- opment, International Journal for Modern Trends in Science and Technology