Tech Saksham

Final project Report

"Full Stack Web Development"

"E-Store The RKV E-commerce Website"

Submitted By
Nagalakshmi Vepamjeri
R170403-CSE

Mr.Poova Raghavan
Master Triner

Index

Abstract

In the existing Manual store , The customer has to manually visit the store, purchase items, stand in the queue for billing and complete billing manually, which is a time consuming process and increase the crowd in the store. So to avoid these problems this E-Store is introduced. E-Store will help users to book the groceries ,Food ,Cakes for Birthdays etc in Advance so that the Crowd and Rush Problem will be reduced. Instead of Physically Approaching the store and Standing in the Queue for Billing,This Site will help users to Book items in Bulk and Also make the Billing Easy through Online. This Web-Application is Much User-friendly and Secure.

INTRODUCTION

- The main objective of the proposed system is to avoid the manual visiting of the store .By using this E-store website purchasing and billing of the items will be done through online.
- This E-STORE automates the entire process of Purchasing and billing of items in the university, saving time and resources by letting Customers focus on the important tasks before them and eliminating the traditional need to visit the store manually.
- This approach basically deals with customer registering for the website and they can add their items to the cart or directly purchase the items from this site, and they can pay for their items through the their cards online .
- Through this website the Admin can view the orders placed by the users and he can either ship the product, deliver the product or reject it.

Purpose

E-Store is a E-commerce site especially designed for the purpose of our College Store. The Estore will help users to book the groceries ,Food ,Cakes for Birthdays etc in Advance so that the Crowd and Rush Problem will be reduced. Instead of Physically Approaching the store and Standing in the Queue for Billing, This Site will help users to Book items in Bulk and Also make the Billing Easy through Online. This Web-Application is Much User-friendly and Secure.

Intended Audience

1).Customers
2).Admin

Product Vision

The main vision of the project is automation of the store. Instead of Physically Approaching the store and Standing in the Queue for Billing ,This Site will help users to Book items in Bulk and Also make the Billing Easy through Online.

- **Existing System :**

As of Today, Students in our College has to Visit the Store and Buy Items Manually and the Billing Process is also Manual.This is a Time Consuming and Traditional Process. It will be Hard for the Store Management to Control the Crowd. It is also Hard For the Students to Buy item during Rush-days and Stand in the Queue for Billing. It is also difficult for the students to book cakes for Birthdays .Students also face-difficulties in Canteen.

- **Proposed System**

We Propose System:

➢ Will allow users to Book the Groceries through Online from our Store
➢ Will allow users to Book Cakes through Online in Advance.
➢ Will reduce difficulties in the Canteen.
➢ Billing through Online Which will reduce the Time of Billing.
➢ User-friendly.
➢ Easy to Use With Better UI

Requirement Specification

Hardware Configuration:
Client side:

Ram	512 MB

Hard disk	10GB
Processor	1.0 GHz

Server side:

Ram	1GB
Hard disk	20GB
Processor	2.0GHz

Software Requirements:

Front end	ReactJS,Boostrap
Server side Language	NodeJs
Database Server	Mongo db
Web Browser	Firefox,Chrome or compatible any Browser
Operating System	Ubuntu,Windows or Compatible any Browser
Software	EPASS

MODULES

- ADMIN
  - Admin Login Page
  - Admin Home Page
  - Add Category
  - Manage Category
  - View Category
  - Edit Category

- Delete Category
- Add Product
- Manage Product
- View Product
- Edit Product
- Delete Product
- Manage Order
- View Order
- Edit Order Status

- CUSTOMER INTERFACE
  - Customer Login and Registration Page

- CART
  - Add To Cart Page
  - Cart Product Before Check Out
  - Final Checkout And Add Shopping Information

- E-COMMERCE PROCESS
  - Select Payment Method
  - Smart Card

Modules Description

- **ADMIN**
  - **Admin Login Page**

Admin part in so important part of the system and he take cares all the part the shopping system. When the system developed an admin user created and using admin user email id and password he can login to the system figure given below.

- **Admin Home Page**

Admin will get different menu in his panel after login with valid user id and password. The menu admin will get are Category, under category will get submenu Add Category, Manage Category and Archive Category. He will also get Manufacturer menu, under Manufacturer menu will get submenu Add

Manufacturer, Manage Manufacturer. He will get Add product, Manage Product and Archive Product submenu under product menu. The customer request for product will show on Manager Order menu.

- **Add Category**

From admin panel admin add category that will show in home page if admin select the publication status as published, and will not show if he select publication status as unpublished. Figure of Add Category given below.

- **Manage Category**

Manage category is an important part of admin panel, for set status that the category will show or not in home page admin can select this. He can view, edit and delete the category from Manage Category menu. Figure of Manage category given below.

- **View Category**

In view category submenu customer can show the details of the category. Figure of View Category given below.

- **Edit Category**

If mistake done when added category, by editing the page admin can correct the information that updated when it created. Figure of edit category given below.

- **Delete Category**

Unused category or the product category the company decided to stop sell permanently can delete, before deleting it, a notification will come to reconfirm that admin sure he wants to delete it. Figure of Delete category given below**.**

- **Add Product**

From admin panel admin Add Product that will show in home page if admin select the publication status as published, and will not show if he select publication status as unpublished. Figure of Add Product given below.

- **Manage Product**

Manage Product is an important part of admin panel, for setting status that the Product will show or not in home page, admin can select this. He can view, edit and delete the product from Manage Product submenu. Figure of Manage Product given below.

- **View Product**

In View option customer can show the details of the product Figure of view product given below.

- **Edit Product**

If mistake done when added product, by editing the page admin can correct the information that updated when it created. Figure of Edit Product given below.

- **Delete Product**

Unused Product or the product the company decided to stop sell permanently can delete, before deleting it, a notification will come to reconfirm that admin sure he wants to delete it. Figure of Delete Product given below.

➢ Manage Order

The product which were ordered by the customer can manage from this menu. When a product delivered to the customer, and customer paid for this, then need to change product order status to delivered and paid, that can be done from this menu. Figure of Manage Order given below

➢ View Order

In view category submenu customer can show the details of the category. Figure of View Order given below.

➢ Edit Order Status

The product that delivered the customer have to change the product deliver status as delivered and as a result the product will not display at admin.

- CUSTOMER INTERFACE

➢ CUSTOMER LOGIN AND REGISTRATION PAGE

Customer have to login before adding product in cart. In this page existing customer can login to buy product and new user can create an account for buying the product.
Figure of user login and registration page given below.

- CART

➢ Add To Cart Page

To buy a product the customer have to add product to cart. Also customer can view the product details, as well as large view by putting the cursor over the product image. The figure of adding product to cart given

➢ CART PRODUCT BEFORE CHECK OUT

The customer can view the product, update number of product, delete product from cart and send request to check out and given figure below.

➢ FINAL CHECKOUT AND ADD SHIPPING INFORMATION

From the customer panel before adding shipping information he can see the product details, also have to add.
Shipping information. The figure of Final Checkout and Add Shipping Information method given below.

- E-COMMERCE PROCESS

➢ Select Payment Method

There are several method of payment to pay to product cost, customer select any of them. After selecting the payment method customer have to confirm the order. The figure of payment method given below.

➢ Smart Cart

Smart card is again similar to a credit card or a debit card in appearance, but it has a small microprocessor

Smart cards are also used to store money and the amount gets deducted after every transaction.

Technologies Used

- FRONTEND : Reactjs,Bootsrap .
- BACKEND : Nodejs,Mongodb.
- ENVIRONMENT : Visual Studio Code.

REACT JS

React is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta and a community of individual developers and companies. React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js.

BOOTSTRAP

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Node JS

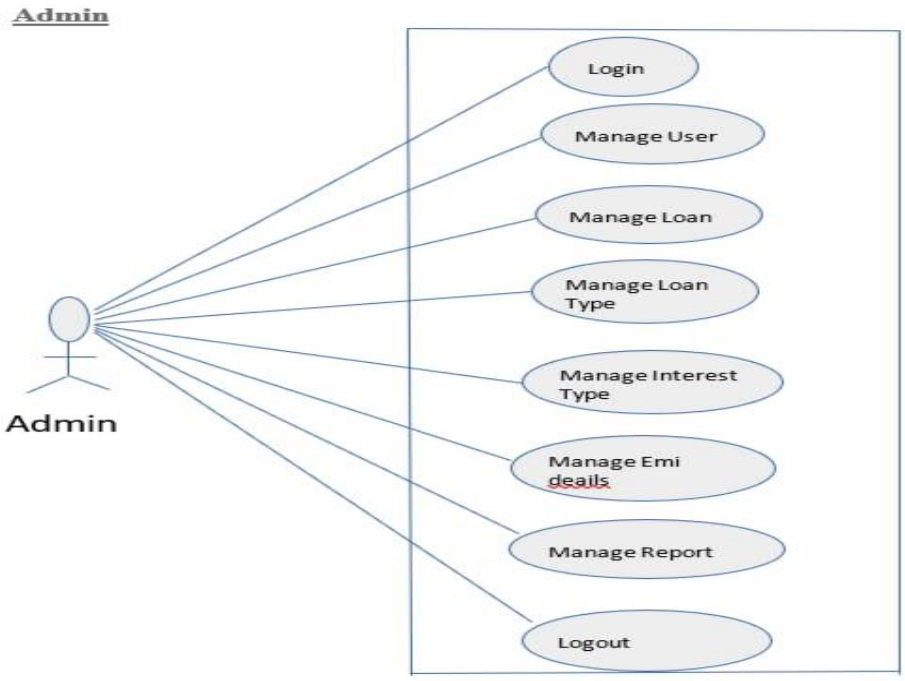
Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

MONGO DB

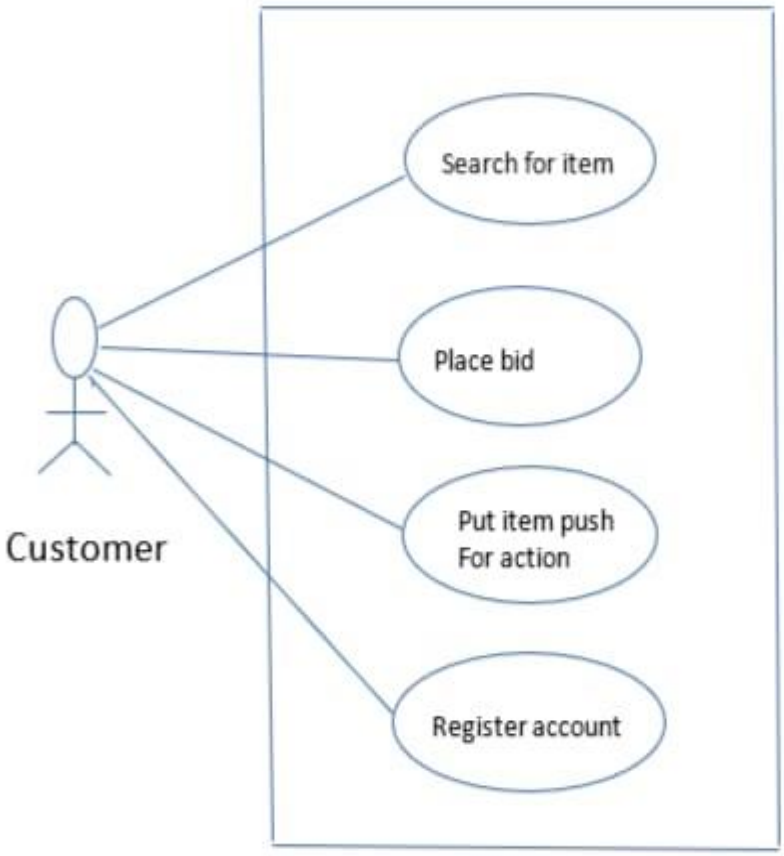
MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License which is deemed non-free by several distributions.

Use case Diagram

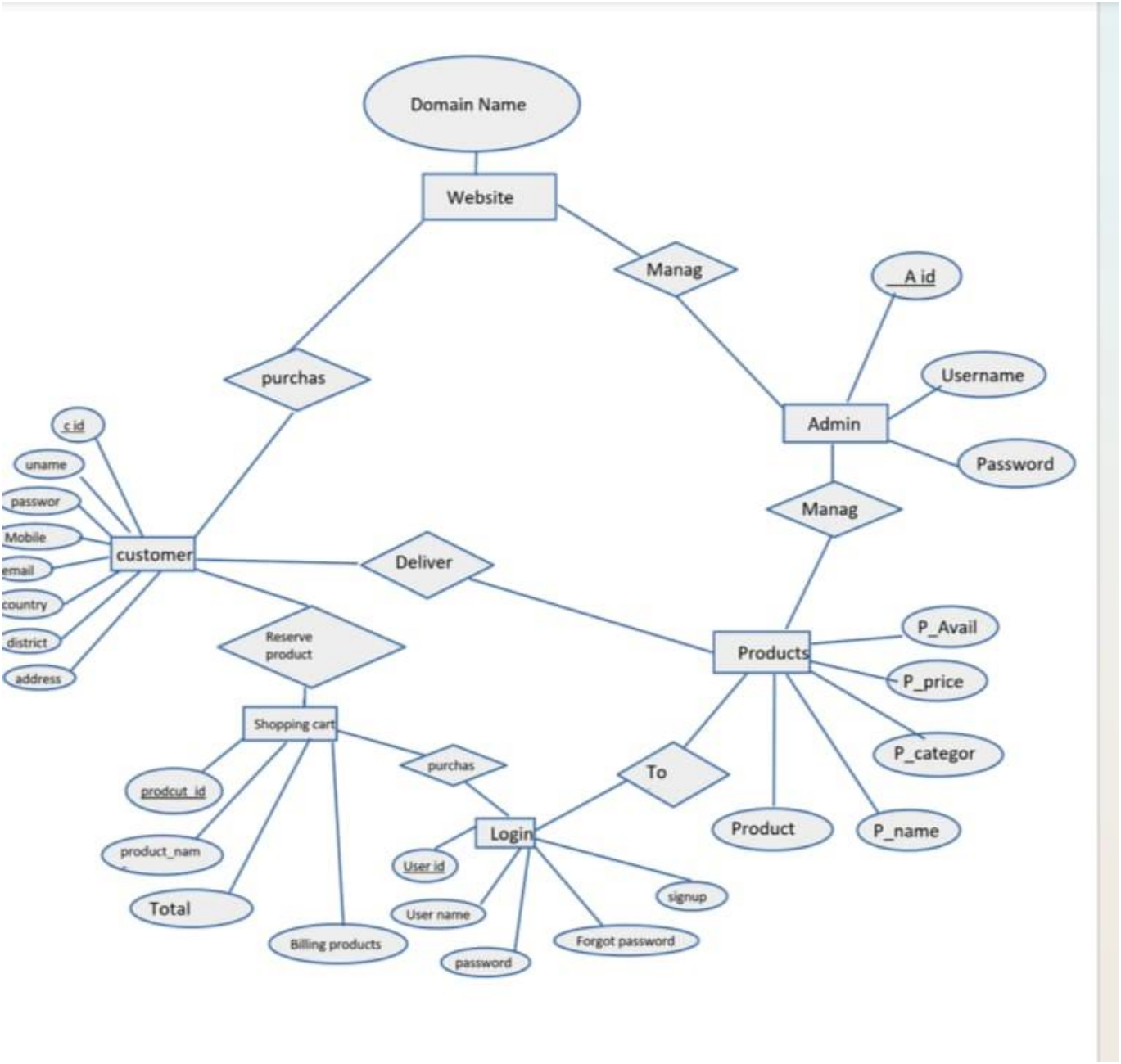
Admin


Admin
Admin
Login
Manage User
Manage Loan
Manage Loan Type
Manage Interest Type
Manage Emi deails
Manage Report
Logout

Customer


Search for item
Place bid
Put item push
For action
Register account
Customer

ER Diagram


Domain Name
Website
Manag
A id
purchas
Username
Admin
Password
c id
uname
passwor
Mobile
email
country
district
address
customer
Deliver
Manag
Reserve product
Products
P_Avail
P_price
P_categor
Shopping cart
purchas
To
prodcut_id
product_nam
Total
Billing products
Login
User id
User name
password
Forgot password
signup
Product
P_name

SOURCE CODE:

Home.js:

```
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { Checkbox, Radio } from "antd";
import { Prices } from "../components/Prices";
import { useCart } from "../context/cart";
import axios from "axios";
import toast from "react-hot-toast";
import Layout from "./../components/Layout/Layout";
import { AiOutlineReload } from "react-icons/ai";
import "../styles/Homepage.css";

const HomePage = () => {
  const navigate = useNavigate();
  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);

  //get all cat
  const getAllCategory = async () => {
    try {
      const { data } = await axios.get("/api/v1/category/get-category");
      if (data?.success) {
        setCategories(data?.category);
      }
    } catch (error) {
      console.log(error);
    }
  };

  useEffect(() => {
    getAllCategory();
    getTotal();
  }, []);
  //get products
```

```
const getAllProducts = async () => {
  try {
    setLoading(true);
    const { data } = await axios.get(`/api/v1/product/product-list/${page}`);
    setLoading(false);
    setProducts(data.products);
  } catch (error) {
    setLoading(false);
    console.log(error);
  }
};

//getTOtal COunt
const getTotal = async () => {
  try {
    const { data } = await axios.get("/api/v1/product/product-count");
    setTotal(data?.total);
  } catch (error) {
    console.log(error);
  }
};

useEffect(() => {
  if (page === 1) return;
  loadMore();
}, [page]);
//load more
const loadMore = async () => {
  try {
    setLoading(true);
    const { data } = await axios.get(`/api/v1/product/product-list/${page}`);
    setLoading(false);
    setProducts([...products, ...data?.products]);
  } catch (error) {
    console.log(error);
    setLoading(false);
  }
};

// filter by cat
const handleFilter = (value, id) => {
  let all = [...checked];
  if (value) {
    all.push(id);
  } else {
```

```
    all = all.filter((c) => c !== id);
  }
  setChecked(all);
};
useEffect(() => {
  if (!checked.length || !radio.length) getAllProducts();
}, [checked.length, radio.length]);

useEffect(() => {
  if (checked.length || radio.length) filterProduct();
}, [checked, radio]);

//get filterd product
const filterProduct = async () => {
  try {
    const { data } = await axios.post("/api/v1/product/product-filters", {
      checked,
      radio,
    });
    setProducts(data?.products);
  } catch (error) {
    console.log(error);
  }
};
return (
  <Layout title={"E-RKV STORE"}>
    <div className="container-fluid row mt-3 home-page">
      <div className="col-md-3 filters">
        <h4 className="text-center">Filter By Category</h4>
        <div className="d-flex flex-column">
          {categories?.map((c) => (
            <Checkbox
              key={c._id}
              onChange={(e) => handleFilter(e.target.checked, c._id)}
            >
              {c.name}
            </Checkbox>
          ))}
        </div>
        {/* price filter */}
        <h4 className="text-center mt-4">Filter By Price</h4>
        <div className="d-flex flex-column">
          <Radio.Group onChange={(e) => setRadio(e.target.value)}>
            {Prices?.map((p) => (
              <div key={p._id}>
```

```
            <Radio value={p.array}>{p.name}</Radio>
          </div>
        ))}
      </Radio.Group>
    </div>
    <div className="d-flex flex-column">
      <button
        className="btn btn-danger"
        onClick={() => window.location.reload()}
      >
        RESET FILTERS
      </button>
    </div>
  </div>
  <div className="col-md-9 ">
    <h1 className="text-center">All Products</h1>
    <div className="d-flex flex-wrap">
      {products?.map((p) => (
        <div className="card m-2" key={p._id}>
          <img
            src={`/api/v1/product/product-photo/${p._id}`}
            className="card-img-top"
            alt={p.name}
          />
          <div className="card-body">
            <div className="card-name-price">
              <h5 className="card-title">{p.name}</h5>
              <h5 className="card-title card-price">
                {p.price.toLocaleString("en-US", {
                  style: "currency",
                  currency: "USD",
                })}
              </h5>
            </div>
            <p className="card-text ">
              {p.description.substring(0, 60)}...
            </p>
            <div className="card-name-price">
              <button
                className="btn btn-info ms-1"
                onClick={() => navigate(`/product/${p.slug}`)}
              >
                More Details
              </button>
              <button
```

```
                    className="btn btn-dark ms-1"
                    onClick={() => {
                      setCart([...cart, p]);
                      localStorage.setItem(
                        "cart",
                        JSON.stringify([...cart, p])
                      );
                      toast.success("Item Added to cart");
                    }}
                  >
                    ADD TO CART
                  </button>
                </div>
              </div>
            </div>
          ))}
        </div>
        <div className="m-2 p-3">
          {products && products.length < total && (
            <button
              className="btn loadmore"
              onClick={(e) => {
                e.preventDefault();
                setPage(page + 1);
              }}
            >
              {loading ? (
                "Loading ..."
              ) : (
                <>
                  {" "}
                  Loadmore <AiOutlineReload />
                </>
              )}
            </button>
          )}
        </div>
      </div>
    </div>
  </Layout>
 );
};

export default HomePage;
```

Admin Dashboard:

```
import React from "react";
import AdminMenu from "../../components/Layout/AdminMenu";
import Layout from "./../../components/Layout/Layout";
import { useAuth } from "../../context/auth";
const AdminDashboard = () => {
  const [auth] = useAuth();
  return (
    <Layout>
      <div className="container-fluid m-3 p-3 dashboard">
        <div className="row">
          <div className="col-md-3">
            <AdminMenu />
          </div>
          <div className="col-md-9">
            <div className="card w-75 p-3">
              <h3> Admin Name : {auth?.user?.name}</h3>
              <h3> Admin Email : {auth?.user?.email}</h3>
              <h3> Admin Contact : {auth?.user?.phone}</h3>
            </div>
          </div>
        </div>
      </div>
    </Layout>
  );
};

export default AdminDashboard;
```

Backend:

Authcontroller.js:

```
import userModel from "../models/userModel.js";
import orderModel from "../models/orderModel.js";

import { comparePassword, hashPassword } from "./../helpers/authHelper.js";
import JWT from "jsonwebtoken";
```

```
export const registerController = async (req, res) => {
  try {
    const { name, email, password, phone, address, answer } = req.body;
    //validations
    if (!name) {
      return res.send({ error: "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 no is Required" });
    }
    if (!address) {
      return res.send({ message: "Address is Required" });
    }
    if (!answer) {
      return res.send({ message: "Answer is Required" });
    }
    //check user
    const exisitingUser = await userModel.findOne({ email });
    //exisiting user
    if (exisitingUser) {
      return res.status(200).send({
        success: false,
        message: "Already Register please login",
      });
    }
    //register user
    const hashedPassword = await hashPassword(password);
    //save
    const user = await new userModel({
      name,
      email,
      phone,
      address,
      password: hashedPassword,
      answer,
    }).save();

    res.status(201).send({
```

```
      success: true,
      message: "User Register Successfully",
      user,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      message: "Errro in Registeration",
      error,
    });
  }
};

//POST LOGIN
export const loginController = async (req, res) => {
  try {
    const { email, password } = req.body;
    //validation
    if (!email || !password) {
      return res.status(404).send({
        success: false,
        message: "Invalid email or password",
      });
    }
    //check user
    const user = await userModel.findOne({ email });
    if (!user) {
      return res.status(404).send({
        success: false,
        message: "Email is not registerd",
      });
    }
    const match = await comparePassword(password, user.password);
    if (!match) {
      return res.status(200).send({
        success: false,
        message: "Invalid Password",
      });
    }
    //token
    const token = await JWT.sign({ _id: user._id }, process.env.JWT_SECRET, {
      expiresIn: "7d",
    });
    res.status(200).send({
```

```
      success: true,
      message: "login successfully",
      user: {
        _id: user._id,
        name: user.name,
        email: user.email,
        phone: user.phone,
        address: user.address,
        role: user.role,
      },
      token,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      message: "Error in login",
      error,
    });
  }
};

//forgotPasswordController

export const forgotPasswordController = async (req, res) => {
  try {
    const { email, answer, newPassword } = req.body;
    if (!email) {
      res.status(400).send({ message: "Emai is required" });
    }
    if (!answer) {
      res.status(400).send({ message: "answer is required" });
    }
    if (!newPassword) {
      res.status(400).send({ message: "New Password is required" });
    }
    //check
    const user = await userModel.findOne({ email, answer });
    //validation
    if (!user) {
      return res.status(404).send({
        success: false,
        message: "Wrong Email Or Answer",
      });
    }
```

```
    const hashed = await hashPassword(newPassword);
    await userModel.findByIdAndUpdate(user._id, { password: hashed });
    res.status(200).send({
      success: true,
      message: "Password Reset Successfully",
    });
  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      message: "Something went wrong",
      error,
    });
  }
};

//test controller
export const testController = (req, res) => {
  try {
    res.send("Protected Routes");
  } catch (error) {
    console.log(error);
    res.send({ error });
  }
};

//update prfole
export const updateProfileController = async (req, res) => {
  try {
    const { name, email, password, address, phone } = req.body;
    const user = await userModel.findById(req.user._id);
    //password
    if (password && password.length < 6) {
      return res.json({ error: "Passsword is required and 6 character long" });
    }
    const hashedPassword = password ? await hashPassword(password) : undefined;
    const updatedUser = await userModel.findByIdAndUpdate(
      req.user._id,
      {
        name: name || user.name,
        password: hashedPassword || user.password,
        phone: phone || user.phone,
        address: address || user.address,
      },
      { new: true }
```

```
    );
    res.status(200).send({
      success: true,
      message: "Profile Updated SUccessfully",
      updatedUser,
    });
  } catch (error) {
    console.log(error);
    res.status(400).send({
      success: false,
      message: "Error WHile Update profile",
      error,
    });
  }
};

//orders
export const getOrdersController = async (req, res) => {
  try {
    const orders = await orderModel
      .find({ buyer: req.user._id })
      .populate("products", "-photo")
      .populate("buyer", "name");
    res.json(orders);
  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      message: "Error WHile Geting Orders",
      error,
    });
  }
};
//orders
export const getAllOrdersController = async (req, res) => {
  try {
    const orders = await orderModel
      .find({})
      .populate("products", "-photo")
      .populate("buyer", "name")
      .sort({ createdAt: "-1" });
    res.json(orders);
  } catch (error) {
    console.log(error);
    res.status(500).send({
```

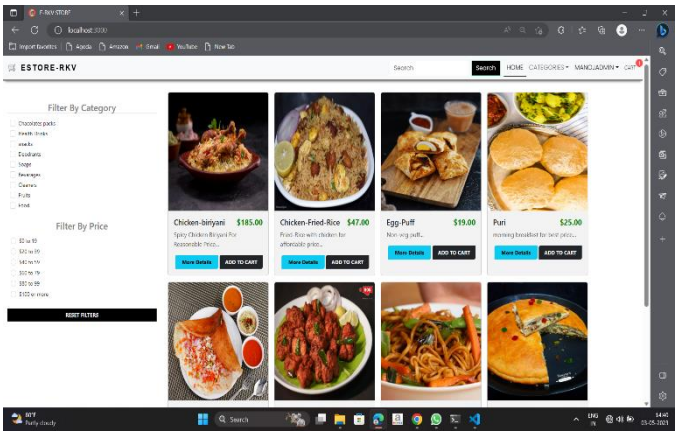
```
      success: false,
      message: "Error WHile Geting Orders",
      error,
    });
  }
};

//order status
export const orderStatusController = async (req, res) => {
  try {
    const { orderId } = req.params;
    const { status } = req.body;
    const orders = await orderModel.findByIdAndUpdate(
      orderId,
      { status },
      { new: true }
    );
    res.json(orders);
  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      message: "Error While Updateing Order",
      error,
    });
  }
};
```

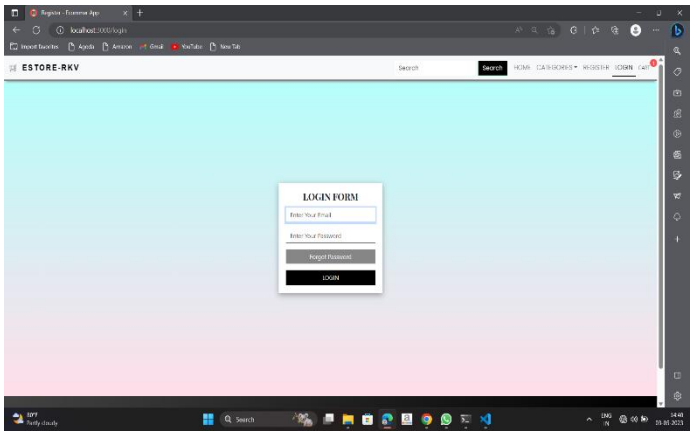
middleware.js:

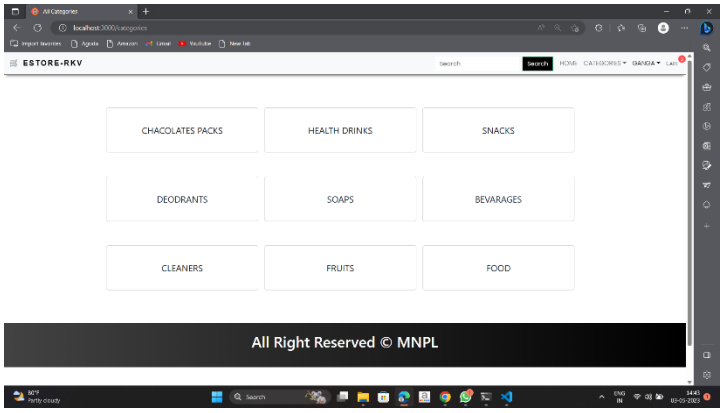
```
import JWT from "jsonwebtoken";
import userModel from "../models/userModel.js";

//Protected Routes token base
export const requireSignIn = async (req, res, next) => {
  try {
    const decode = JWT.verify(
      req.headers.authorization,
      process.env.JWT_SECRET
    );
    req.user = decode;
    next();
```

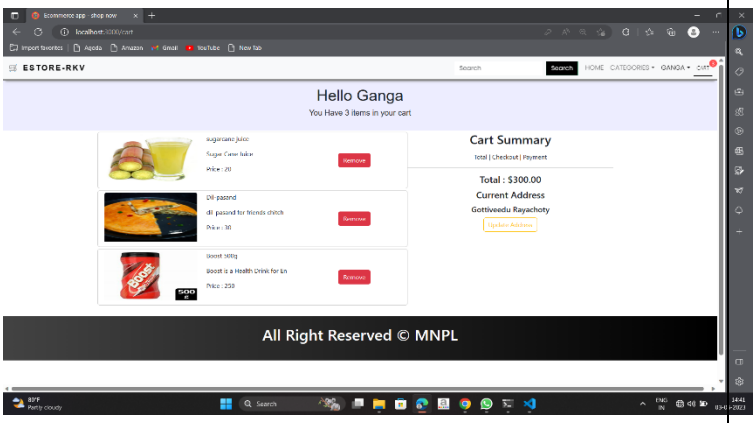
```
  } catch (error) {
    console.log(error);
  }
};

//admin acceess
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",
    });
  }
};
```

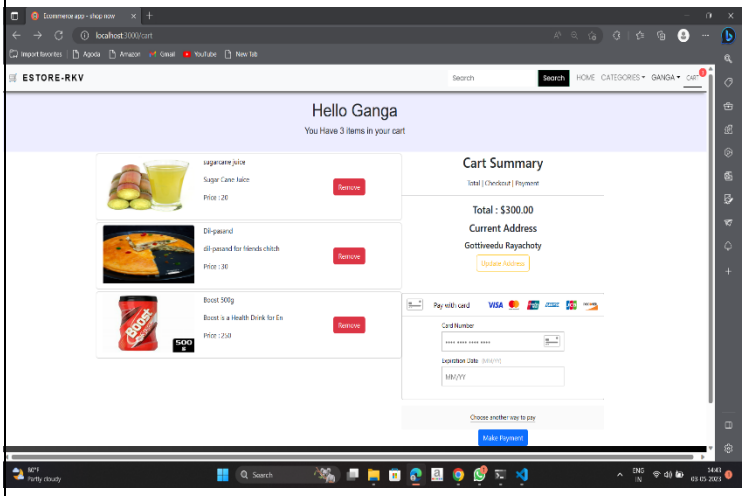
OUTPUT RESULTS:

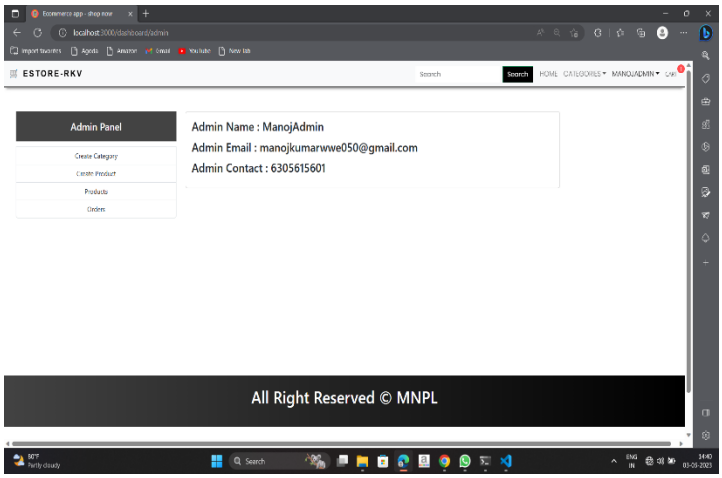












Conclusion:

This web Application provides Costumers to Purchase items through online without standing in the Queue for billing and improve Crowd management in the Store. It saves time and also makes the sellers life also easy and helps them to increase their sales. The E-store RKV e-commerce site need to be tested thoroughly tested before implementation to find any security gaps.

Future Enhancement:

Introduce fast delivery system

Introduce Coupan and voucher mechanism

Introduce OTP mechanism after order and delivery.

References:

https://www.w3schools.com
https://www.mongoodb.com/
https://react.dev/
https://www.javatpoint.com/
https://nodejs.org/en/docs