SCTR's Pune Institute of Computer Technology Dhankawadi, Pune

A.Y. 2023-24

WADL MINI PROJECT REPORT ON

"Ecommerce Application"

Submitted By

Rupesh Kharche – 33336
Abhijit Khyade – 33337
Manasi Lavekar – 33340

Under the guidance of

Mrs. Deepali Salapurkar

DEPARTMENT OF INFORMATION TECHNOLOGY
ACADEMIC YEAR 2023-24

ABSTRACT

This project presents the development of an e-commerce application specifically tailored for the retail of watches, utilizing the MERN (MongoDB, Express.js, React.js, Node.js) stack. With the increasing popularity of online shopping, the demand for efficient and user-friendly e commerce platforms has surged. The focus on watches caters to a niche market segment while providing insights into the customization and scalability capabilities of the MERN stack.

The application encompasses key features essential for an optimal e-commerce experience, including user authentication, product browsing, cart management, secure payment processing, and order tracking. Leveraging React.js, the frontend interface offers a seamless and responsive user experience, facilitating smooth navigation and efficient interaction. Concurrently, the backend, powered by Node.js and Express.js, ensures robustness, scalability, and data integrity, with MongoDB serving as the reliable database solution.

Moreover, the project delves into advanced functionalities such as personalized recommendations, inventory management, and analytics integration to enhance user engagement and streamline business operations. By adhering to industry best practices and employing modern development methodologies, this e-commerce application serves as a comprehensive demonstration of MERN stack's capabilities in building sophisticated and scalable online platforms, particularly tailored for niche markets like watches

INTRODUCTION

In the contemporary digital landscape, e-commerce has revolutionized the way consumers shop, offering convenience, accessibility, and a diverse range of products at their fingertips. As the e-commerce industry continues to expand, there arises a growing demand for specialized platforms catering to niche markets. This project endeavors to address this need by developing a bespoke e-commerce application focused primarily on the retail of watches, leveraging the MERN (MongoDB, Express.js, React.js, Node.js) stack.

The decision to concentrate on watches stems from the enduring popularity and timeless appeal of these accessories. Watches serve not only as functional timekeeping devices but also as fashion statements, reflecting individual style and personality. By centering our e commerce application around watches, we aim to create a dedicated platform that resonates with enthusiasts, collectors, and consumers seeking premium timepieces.

The project's scope extends beyond mere retail; it encapsulates the intricacies of designing, implementing, and deploying a comprehensive e-commerce solution. From frontend development using React.js to backend management with Node.js and Express.js, and data storage in MongoDB, every aspect of the MERN stack will be explored and harnessed to create a robust, feature-rich, and scalable platform.

Through this project, we seek to demonstrate the capabilities of the MERN stack in crafting sophisticated e-commerce applications tailored to specific market segments. By combining cutting-edge technologies, industry best practices, and innovative design principles, our aim is to deliver an immersive and seamless shopping experience for watch enthusiasts while providing valuable insights into the development process and the potential applications of the MERN stack in the e-commerce domain.

In the dynamic landscape of digital commerce, the emergence of specialized e-commerce platforms has become pivotal in catering to the diverse preferences and interests of consumers worldwide. Within this paradigm, our project embarks on the development of a bespoke e-commerce application focused exclusively on the realm of watches, harnessing the power and versatility of the MERN (MongoDB, Express.js, React.js, Node.js) stack.
Watches hold a unique allure in the world of fashion and personal expression, transcending their utilitarian function to become emblematic symbols of style, sophistication, and individuality. With a rich history and a vibrant culture surrounding them, watches attract a dedicated community of enthusiasts, collectors, and aficionados. Our decision to centre the e-commerce platform on watches stems from a recognition of this passionate community and the inherent potential for delivering tailored experiences to meet their discerning tastes and preference.

LITERATURE SURVEY

1. **"React.js Essentials" by Artemij Fedosejev**:
   - This comprehensive guide explores both fundamental and advanced concepts of React.js, providing essential knowledge for frontend development.

2. **"Node.js Web Development" by David Herron**:
   - Offering a thorough exploration of Node.js backend development and deployment, this resource is invaluable for understanding the intricacies of server-side JavaScript programming.

3. **"Express.js Guide" by Azat Mardan**:
   - Delving deep into Express.js, this guide equips developers with the necessary skills to build RESTful APIs and web applications with ease and efficiency.

4. **"MongoDB: The Definitive Guide" by Shannon Bradshaw et al.**:
   - This authoritative guide covers the installation, data modeling, querying, and scaling aspects of MongoDB, providing a comprehensive understanding of this NoSQL database system.

5. **"React for E-commerce" by Gaurav Ramanan**:
   - Focusing specifically on leveraging React.js for ecommerce applications, this resource offers insights into creating product listings, shopping carts, and seamless checkout flows.

6. **"Web Application Security: A Beginner's Guide" by Bryan Sullivan**:
   - Providing an introductory overview of web application security principles, this guide addresses securing Node.js and MongoDB applications, ensuring robust protection against potential threats.

IMPLEMENTATION DETAILS

Web technologies used:

1. **MongoDB**: As the foundation of our data storage layer, MongoDB offers flexibility and scalability, allowing us to efficiently manage product information, user data, and transaction records. Leveraging its document-oriented architecture, we store and retrieve data seamlessly, ensuring robustness and reliability in our e-commerce application.
2. **Express.js**: Serving as the backend framework, Express.js provides a robust and minimalist framework for building web applications and APIs. With its middleware architecture, we handle routing, request processing, and response generation efficiently, ensuring optimal performance and maintainability in our application.
3. **React.js**: Powering the frontend interface, React.js enables us to create dynamic and responsive user interfaces with ease. Through its component-based architecture and virtual DOM, we design interactive UI components, manage state efficiently, and facilitate seamless navigation and user interactions, enhancing the overall user experience.
4. **Node.js**: As the runtime environment for our backend code, Node.js enables JavaScript to run server-side, offering a unified development environment across frontend and backend components. With its event-driven, non-blocking I/O model, we achieve high concurrency and scalability, handling multiple user requests simultaneously with minimal resource consumption.
5. **HTML5/CSS3**: Utilizing the latest standards in web markup and styling, we craft visually appealing and accessible user interfaces. HTML5 provides semantic structure to our content, while CSS3 enables precise control over layout, typography, and visual aesthetics, ensuring a consistent and engaging user experience across different devices and screen sizes.
6. **JavaScript (ES6+)**: Serving as the primary programming language for both frontend and backend development, JavaScript allows us to build interactive and feature-rich web applications. With support for modern language features such as arrow functions, async/await, and destructuring, we write clean, expressive code that enhances productivity and maintainability.
7. **JWT (JSON Web Tokens)**: For implementing secure authentication and authorization mechanisms, we utilize JWT, a compact and self-contained token format. By issuing JWT tokens upon successful authentication, we authenticate user requests and enforce access control policies, ensuring data security and integrity throughout the application.

Frontend development

1. **React.js**: The frontend of our e-commerce application is built entirely using React.js, a popular JavaScript library for building user interfaces. React's component-based architecture allows us to create modular, reusable UI components, enhancing code maintainability and scalability.
2. **Component Library**: We utilize existing component libraries such as Material-UI or Bootstrap to expedite the development process and ensure consistency in design across the application. These libraries provide a wide range of pre-designed components, including buttons, forms, cards, and navigation menus, which we customize and integrate seamlessly into our application.
3. **Responsive Design**: Employing CSS media queries and flexbox/grid layouts, we ensure that our application is fully responsive and adapts gracefully to various screen sizes and devices. This responsive design approach guarantees an optimal viewing experience for users accessing the application from desktops, laptops, tablets, and smartphones.
4. **State Management**: We implement state management using React's built-in state and props system, along with advanced state management libraries such as Redux or Context API for managing global application state. By centralizing and synchronizing state changes across components, we maintain data consistency and facilitate efficient communication between different parts of the application.
5. **Routing**: We utilize React Router, a declarative routing library for React, to handle client-side routing within our single-page application (SPA). With React Router, we define route configurations and navigate between different views/components based on URL changes, providing users with a seamless and intuitive browsing experience.
6. **API Integration**: We interact with backend APIs using asynchronous JavaScript techniques such as Axios or the built-in Fetch API. By sending HTTP requests to the backend server, we retrieve data (e.g., product listings, user information) and update the UI dynamically, ensuring real-time data synchronization and responsiveness.
7. **Form Handling and Validation**: We implement form handling and validation using libraries like Formik and Yup. These libraries streamline the process of building and validating complex forms, providing features such as form state management, input validation, error handling, and form submission, enhancing the usability and reliability of our application.
8. **Authentication and Authorization**: We integrate authentication and authorization mechanisms using JSON Web Tokens (JWT) or OAuth protocols. Upon successful authentication, users receive tokens that are stored securely in browser storage (e.g., localStorage or sessionStorage) and included in subsequent API requests to authenticate and authorize access to protected routes and resources.

Backend development

1. **Node.js**: Our backend is powered by Node.js, a runtime environment for executing JavaScript code server-side. Node.js enables us to build scalable, high-performance applications with its non-blocking, event-driven architecture, facilitating concurrent request handling and optimal resource utilization.
2. **Express.js**: We utilize Express.js, a minimalist web application framework for Node.js, to streamline the development of our backend APIs and middleware. Express.js provides a robust set of features for routing, request handling, middleware integration, and error management, enabling us to build RESTful APIs efficiently. 3. **RESTful API Design**: Following the principles of Representational State Transfer (REST), we design our backend APIs to be RESTful, with well-defined endpoints, HTTP methods, and resource representations. This standardized approach ensures interoperability, scalability, and ease of integration with frontend clients and third party services.
4. **Database Integration (MongoDB)**: We leverage MongoDB, a NoSQL database, for storing and managing application data. MongoDB's flexible document-oriented data model allows us to store data in JSON-like documents, accommodating evolving schema requirements and facilitating seamless integration with JavaScript-based applications.
5. **Mongoose ODM**: We use Mongoose, an Object-Document Mapping (ODM) library for MongoDB, to simplify database interactions and schema validation. Mongoose provides a higher-level abstraction over MongoDB, allowing us to define schemas, perform CRUD operations, and establish relationships between data entities effortlessly.
6. **Authentication Middleware**: We implement authentication middleware using libraries like Passport.js or JSON Web Tokens (JWT). This middleware authenticates incoming requests based on credentials provided by users (e.g., username/password or JWT token), verifying their identity and granting access to protected resources.

Integration

1. **API Endpoints**: We define clear and consistent API endpoints for communication between the frontend and backend components of our e-commerce application. Each endpoint corresponds to a specific resource or functionality (e.g., products, users, orders) and supports standard HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.
2. **RESTful Communication**: Following RESTful principles, we ensure that API endpoints adhere to standard conventions for resource naming, URL structure, and response formats. This standardized approach promotes interoperability, simplifies integration, and enhances scalability and maintainability.
3. **Data Transfer**: We transfer data between the frontend and backend using JSON (JavaScript Object Notation), a lightweight and widely supported data interchange format. JSON provides a simple and human-readable format for representing structured data, making it ideal for transmitting complex objects between client and server.
4. **AJAX Requests**: We utilize Asynchronous JavaScript and XML (AJAX) to facilitate client server communication without requiring page reloads. Using modern Fetch API, we send asynchronous requests to backend APIs, retrieve data, and update the UI dynamically, providing a seamless and responsive user experience.
5. **Error Handling**: We implement robust error handling mechanisms to gracefully manage errors and exceptions that may occur during API communication. In case of errors (e.g., network issues, server errors), we provide meaningful error messages and appropriate HTTP status codes to the client, enabling effective troubleshooting and error resolution.
6. **Authentication and Authorization**: We integrate authentication and authorization mechanisms into API endpoints to ensure secure access to protected resources. Depending on the authentication method (e.g., token-based authentication using JWT), we verify user credentials, generate and validate tokens, and enforce access control policies to protect sensitive data.
7. **Cross-Origin Resource Sharing (CORS)**: To enable secure cross-origin communication between the frontend and backend hosted on different domains, we configure CORS policies on the server-side. By specifying allowed origins, headers, and methods, we mitigate potential security risks associated with cross-origin requests and ensure secure data transmission.
8. **Middleware Integration**: We leverage middleware functions in Express.js to implement cross-cutting concerns such as authentication, request parsing, error handling, and response formatting. Middleware functions intercept incoming requests, perform specific tasks, and pass control to the next middleware in the chain, enabling modular and reusable request processing pipelines

FRONTEND :

1. Category Form :

```
 import React from "react";

const CategoryForm = ({ handleSubmit, value, setValue, inputName }) => {
  return (
    <>
      <form onSubmit={handleSubmit}>
        <div className="mb-3">
          <input
            type="text"
            className="form-control custom-focus-outline"
            placeholder="Enter new category"
            value={value}
            onChange={(e) => {
              setValue(e.target.value);
            }}
          />
        </div>

        <div className="text-right">
          <button type="submit" className="btn btn-outline-primary ">
            {inputName}
          </button>
        </div>
      </form>
    </>
  );
};

export default CategoryForm;
```

2. Search Functionality:

```
import React from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";

import { useSearch } from "../../context/search";
import { BASE_URL } from "../../api";

const SearchInput = () => {
  const [values, setValues] = useSearch();
  const navigate = useNavigate();

  //handle search
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.get(
        `${BASE_URL}/product/search/${values.keyword}`
      );
      setValues({ ...values, results: data });
      navigate("/search");
    } catch (error) {
      console.log(error);
    }
  };

  return (
    <div className="d-flex align-items-center">
      <form className="d-flex" role="search" onSubmit={handleSubmit}>
        <input
          className="form-control border-success custom-input-search"
          type="search"
          placeholder="Search"
          aria-label="Search"
          value={values.keyword}
          onChange={(e) => setValues({ ...values, keyword: e.target.value })}
        />
        <button className="btn btn-success custom-search-btn" type="submit">
          Search
        </button>
      </form>
    </div>
  );
};

export default SearchInput;
```

BACKEND:

1. Controllers :

```
  const slugify = require('slugify');

const categoryModel = require('../models/CategoryModel');

//create category
const createCategoryController = async (req, res) => {
  try {
    const { name } = req.body;
    if (!name) {
      return res.status(401).send({ message: 'Category name is required' });
    }

    const existingCategory = await categoryModel.findOne({ name });
    if (existingCategory) {
      return res.status(200).send({ message: 'Category already Exists...' });
    }
    const newCategory = await new categoryModel({
      name, slug: slugify(name)
    }).save();

    res.status(201).send({
      success: true,
      message: 'New Category created',
      newCategory
    })

  } catch (error) {
    console.log(error);
    res.status(500).send({
      success: false,
      error,
      message: 'Error in  category'
    })
  }
};

//update category
const updateCategoryController = async (req, res) => {
  try {
    const { name } = req.body;
    const { id } = req.params;
    const category = await categoryModel.findByIdAndUpdate(id,
            { name, slug: slugify(name) },
            { new: true });

        res.status(200).send({
            success: true,
            message: 'Category Updated Successfully',
            category
        });

    } catch (error) {
        console.log(error);
        res.status(500).send({
            success: false,
            message: 'Error while updating category',
            error
        })
    }
};

//get all categories
const getCategoriesController = async (req, res) => {
    try {
        const category = await categoryModel.find({});
        res.status(200).send({
            success: true,
            message: 'All Categories List',
            category
        })
    } catch (error) {
        console.log(error);
        res.status(500).send({
            success: false,
            message: "Error while getting all categories",
            error
        })
    }
};

//get single category
const getSingleCategoriesController = async (req, res) => {
    try {
        const category = await categoryModel.findOne({ slug: req.params.slug });
        res.status(200).send({
            success: true,
            message: 'Single Category fetched',
            category
        })
    } catch (error) {
        console.log(error);
        res.status(500).send({
            success: false,
            message: "Error while getting single category",
            error
        })
    }
};

//delete category
const deleteCategoryController = async (req, res) => {
    try {
        const { id } = req.params;

        const category = await categoryModel.findByIdAndDelete(id);

        res.status(200).send({
            success: true,
            message: 'Category deleted Successfully',
            category
        });

    } catch (error) {
        console.log(error);
        res.status(500).send({
            success: false,
            message: 'Error while deleting category',
            error
        })
    }
};

module.exports = { createCategoryController, updateCategoryController,
getCategoriesController, deleteCategoryController, getSingleCategoriesController };
```

2. Routes :

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

const { createCategoryController, updateCategoryController, getCategoriesController,
deleteCategoryController, getSingleCategoriesController } =
require('../controllers/categoryControllers');
const { requireSignIn, isAdmin } = require('../middlewares/authMiddleware');

//router object
```

```
const router = express.Router();

//CREATE CATEGORY || METHOD POST
router.post('/create-category', requireSignIn, isAdmin, createCategoryController);

//UPDATE CATEGORY || METHOD PUT
router.put('/update-category/:id', requireSignIn, isAdmin, updateCategoryController);

//GET ALL CATEGORIES || METHOD GET
router.get('/get-categories', getCategoriesController);

//GET SINGLE CATEGORY || METHOD GET
router.get('/single-category/:slug', getSingleCategoriesController);

//DELETE CATEGORY || METHOD DELETE
router.delete('/delete-category/:id',requireSignIn,isAdmin, deleteCategoryController);

module.exports = router;
```

3. <u>Database Connection</u> :

```
const express = require('express');
const colors = require('colors');
const dotenv = require('dotenv');
const morgan = require('morgan');
const cors = require('cors');

const mongoDB = require('./config/db');
const authRoutes = require('./routes/authRoutes');
const categoryRoutes = require('./routes/categoryRoutes');
const productRoutes = require('./routes/productRoutes');

//config dotenv
dotenv.config();

//database config
mongoDB();

//rest object
const app = express();

//middleware
app.use(express.json());
app.use(cors());
app.use(morgan('dev'));
```

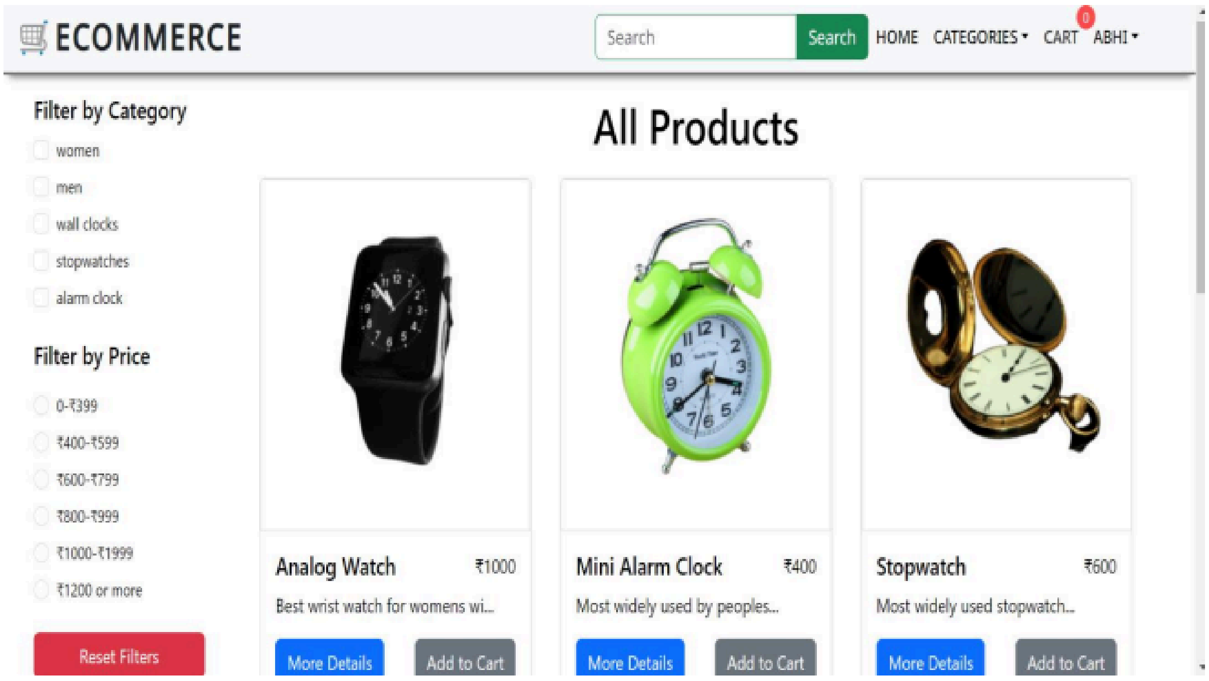
```
//routes
app.use("/auth", authRoutes);
app.use("/category", categoryRoutes);
app.use("/product", productRoutes);


//rest api
app.get('/', (req, res) => {
  res.send('<h1>Welcome to the Ecommerce Website using MERN.</h1>');
});

//PORT
const PORT = process.env.PORT || 8080;

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

OUTPUT



Filter by Category

women
men
wall clocks
stopwatches
alarm clock

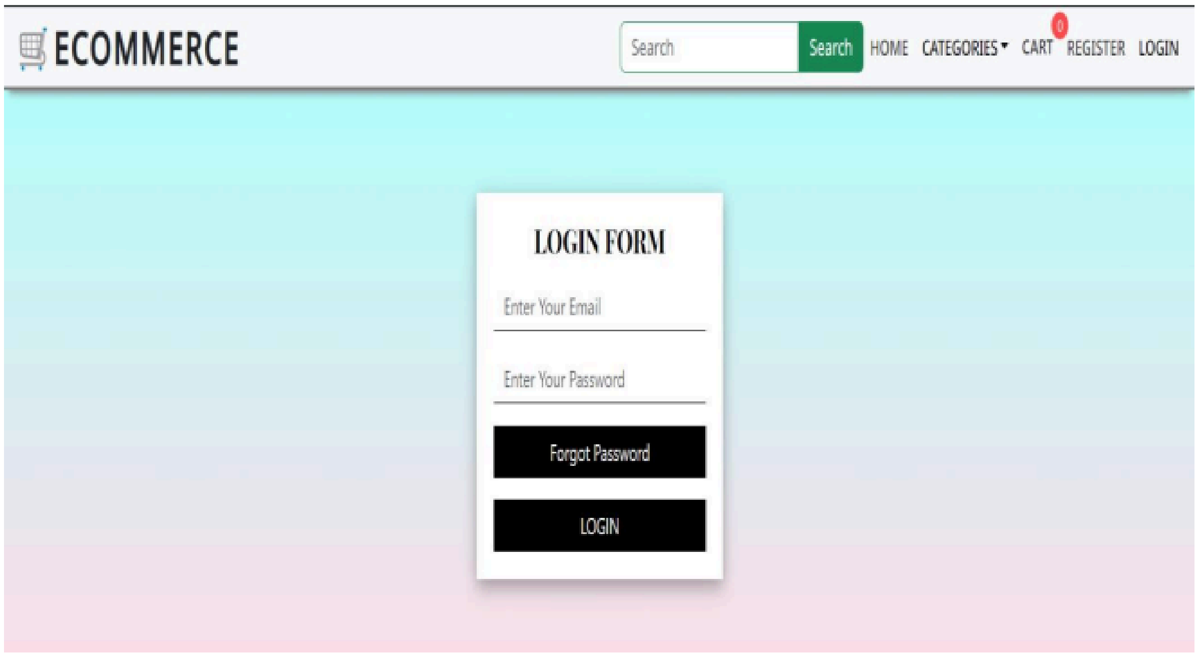
Filter by Price

0-₹399
₹400-₹599
₹600-₹799
₹800-₹999
₹1000-₹1999
₹1200 or more

Reset Filters

All Products

Analog Watch ₹1000	Mini Alarm Clock ₹400	Stopwatch ₹600
Best wrist watch for womens wi...	Most widely used by peoples...	Most widely used stopwatch...
More Details / Add to Cart	More Details / Add to Cart	More Details / Add to Cart



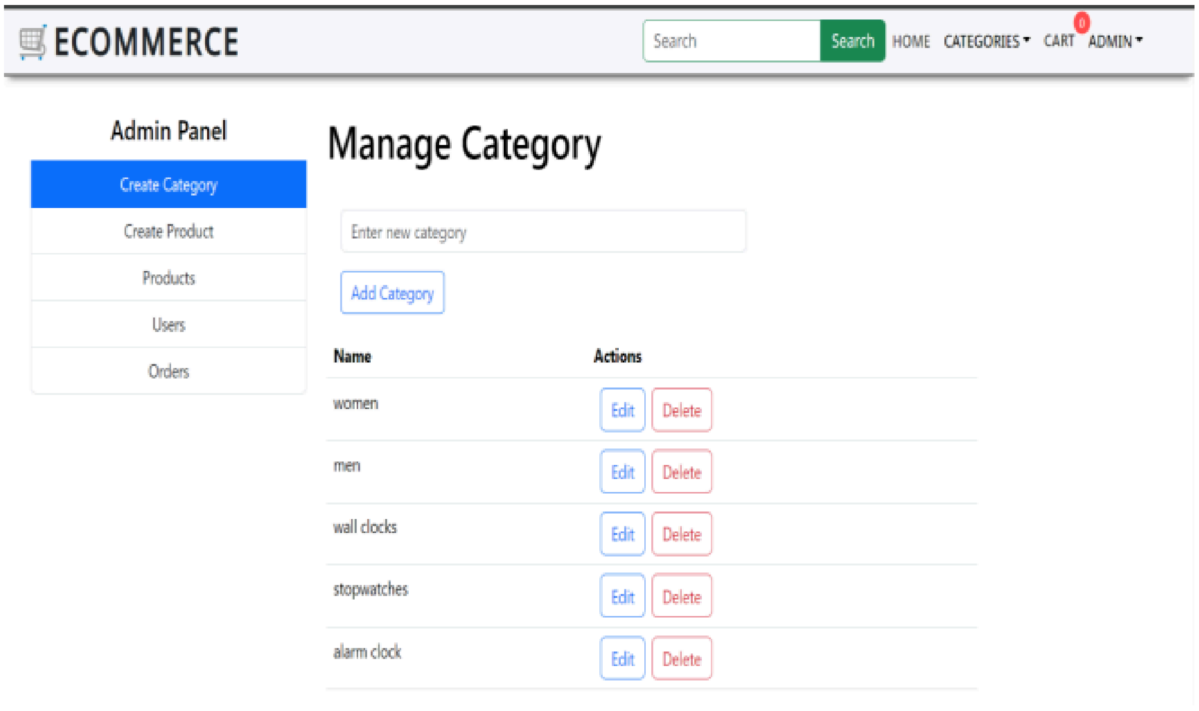
LOGIN FORM

Enter Your Email

Enter Your Password

Forgot Password

LOGIN



ECOMMERCE

Search Search HOME CATEGORIES CART 0 ADMIN

Admin Panel

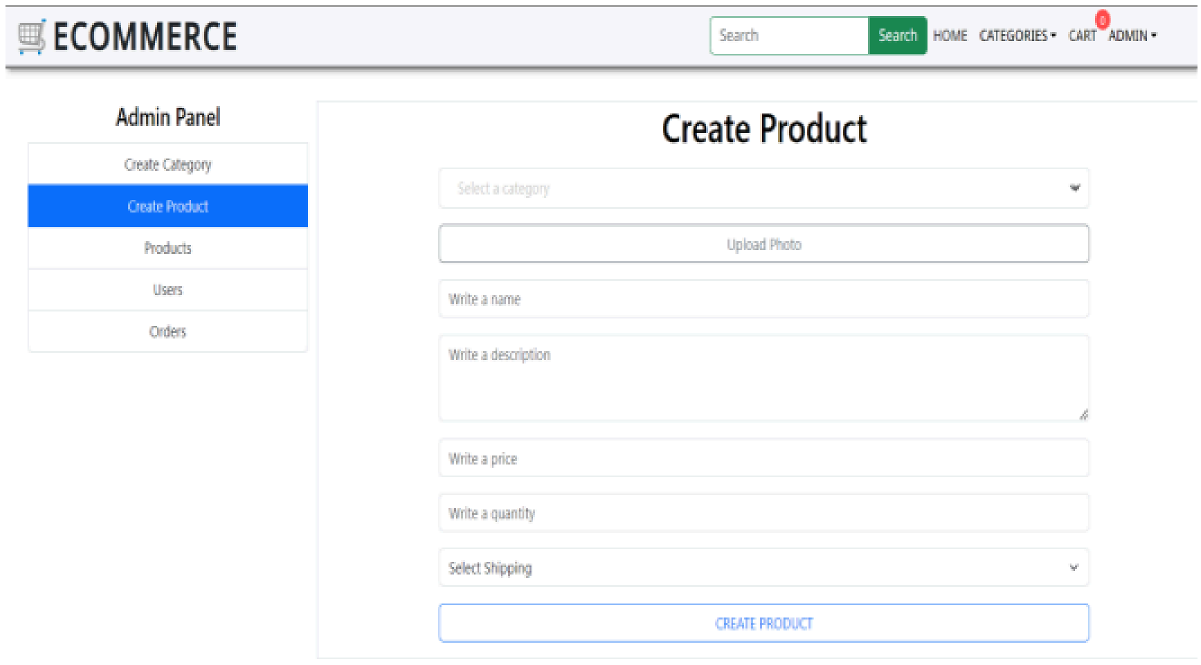
- Create Category
- Create Product
- Products
- Users
- Orders

Manage Category

Enter new category

Add Category

Name	Actions
women	Edit Delete
men	Edit Delete
wall clocks	Edit Delete
stopwatches	Edit Delete
alarm clock	Edit Delete



ECOMMERCE

Search Search HOME CATEGORIES CART 0 ADMIN

Admin Panel

- Create Category
- Create Product
- Products
- Users
- Orders

Create Product

Select a category

Upload Photo

Write a name

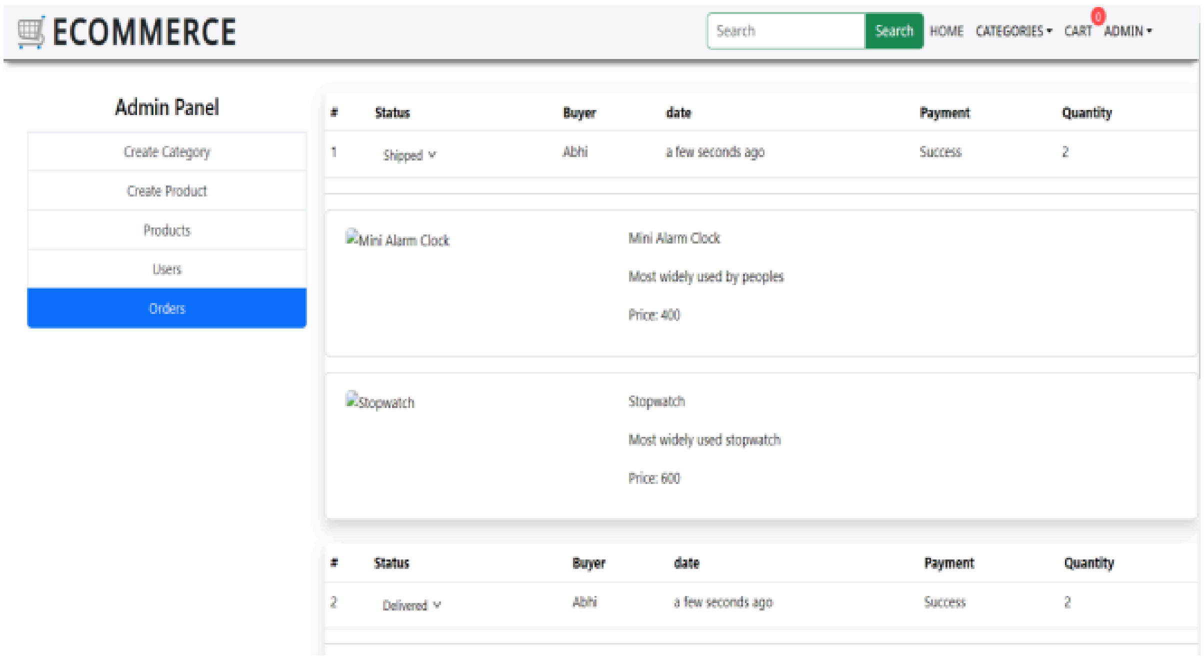
Write a description

Write a price

Write a quantity

Select Shipping

CREATE PRODUCT



ECOMMERCE

Search | HOME | CATEGORIES | CART 0 | ADMIN

Admin Panel

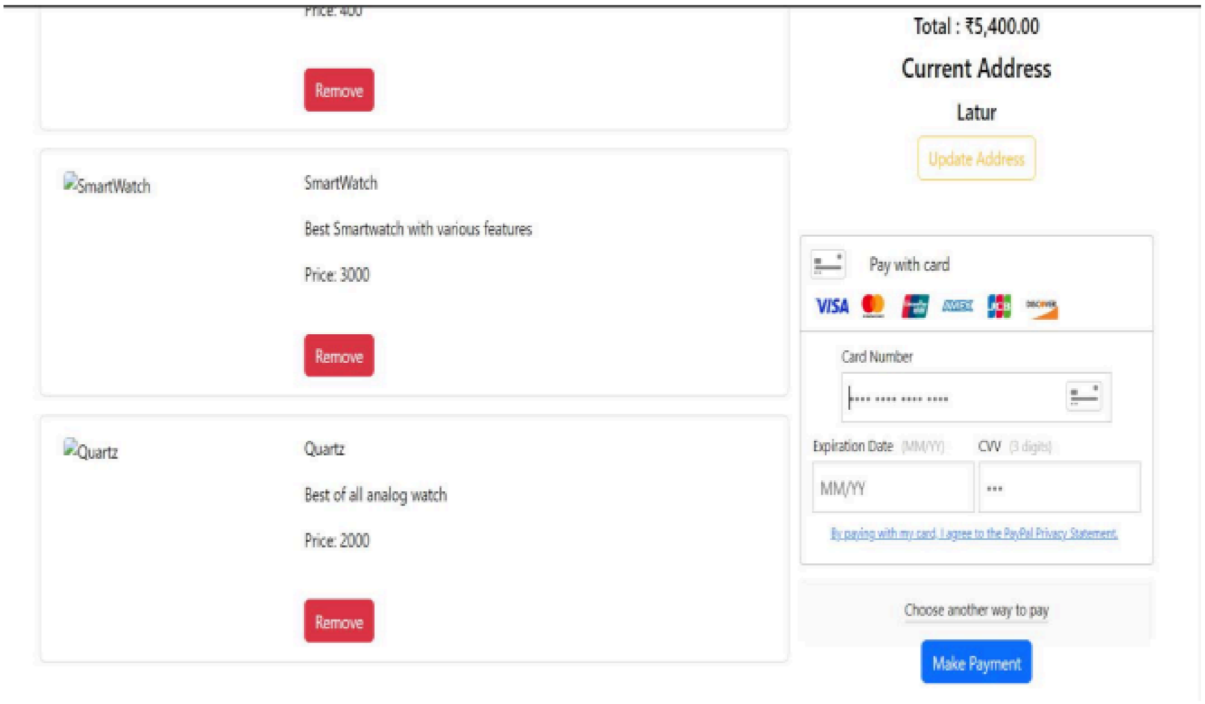
- Create Category
- Create Product
- Products
- Users
- Orders

#	Status	Buyer	date	Payment	Quantity
1	Shipped	Abhi	a few seconds ago	Success	2

Mini Alarm Clock
Mini Alarm Clock
Most widely used by peoples
Price: 400

Stopwatch
Stopwatch
Most widely used stopwatch
Price: 600

#	Status	Buyer	date	Payment	Quantity
2	Delivered	Abhi	a few seconds ago	Success	2



Price: 400

Remove

SmartWatch
SmartWatch
Best Smartwatch with various features
Price: 3000

Remove

Quartz
Quartz
Best of all analog watch
Price: 2000

Remove

Total : ₹5,400.00

Current Address

Latur

Update Address

Pay with card

Card Number

Expiration Date (MM/YY) CVV (3 digits)

MM/YY

By paying with my card, I agree to the PayPal Privacy Statement.

Choose another way to pay

Make Payment

CONCLUSION

In conclusion, our e-commerce application focused on watches, built with the MERN stack, represents a culmination of innovation and dedication. By leveraging modern web technologies, we've created a seamless and intuitive platform for watch enthusiasts. With dynamic frontend interfaces powered by React.js and a robust backend infrastructure using Node.js and Express.js, we've ensured scalability, performance, and security. Through integration of authentication, payment processing, and API communication, we've delivered a secure and reliable shopping experience. As we conclude this project, we acknowledge the ongoing nature of software development and remain committed to continuous improvement. Our e-commerce application stands as a testament to the power of collaboration and innovation in delivering transformative digital experiences in the realm of e-commerce.