ELECTRONICS ECOMMERCE PLATFORM

Project-I Report

Submitted

to

MATS UNIVERSITY RAIPUR, CHHATTISGARH, INDIA

In partially fulfillment for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE & ENGINEERING

By

Amit Sharma

MU22BTCSE002(L)

Under the Guidance of

Ms. Divya Soni

Department of Computer Science Engineering,
School of Engineering and I.T.,
MATS University Raipur, C.G., India

2025

DECLARATION BY THE CANDIDATE

I the undersigned solemnly declare that the project entitled "**Electronics eCommerce Platform**" is based on my own work carried out during my study under the supervision of **Dr. Zubair Ahmed Khan,** Head of Department of Computer Science & Engineering, MATS University, Raipur (C.G.), India.

_____

Amit Sharma

MU22BTCSE002(L)

CERTIFICATE BY THE SUPERVISOR

This is to certify that the project entitled **"Electronics eCommerce Platform"** is a record of research work carried out by **Amit Sharma MU22BTCSE002(L)** under guidance and supervision for the award of Degree of B. Tech in Computer Science & Engineering, School of Engineering and I.T., MATS University, Raipur (C.G.), India.

To the best of my knowledge and belief the report: -

1. Embodies the work of the candidate Himself.
2. Has duly been completed.
3. Fulfils the requirement of the Ordinance relating to the Diploma degree of the University.
4. The desired standards both in respect of contents and language for being referred to the examiners.

(Signature of H.O. D)
Dr. Zubair Ahmed Khan

(Signature of the Supervisor)
Ms. Divya Soni (Professor)

Forwarded to MATS University, Raipur

(Signature of the Director)
School of Engineering & I.T.
MATS University, Raipur

CERTIFICATE BY THE EXAMINERS

The project entitled " Project Title Name" submitted by **Mr. Amit Sharma** bearing Id no. **MU22BTCSE002(L),** Department of Computer Science & Engineering, School of Engineering & IT, MATS University, Aarang, Raipur (C.G.)

Internal Examiner

Date:

External Examiner

Date:

Acknowledgment

The contentment, the accomplishment, the magnificence, the fulfillment, the prize admiration and the creation of my project cannot be thought of without the few, who apart from their regular Schedule spared the invaluable time. A no. of persons contributes either directly or indirectly in shaping and achieving the desired outcome.

A part from the efforts of me, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to extend my gratitude and sincere thanks to many honorable esteemed supervisor and **Dr. Zubair Ahmed Khan** (Head of Department of Computer Science Engineering).I would like to show my greatest appreciation to him. I can't say thank is enough for his tremendous support and help. I feel motivated and encouraged every time I meet him. Without his encouragement and guidance this project would not have been materialized.

The guidance and support received from all the members of management and Principal, to provide mere sources for this project, was vital for the success of the project. I am grateful for their constant support and help.

Last but not the least I am thankful to my family and friends for always encouraging me to achieve this goal. A special thanks to my parents, who taught me the value of hard work by their own example and who instilled me the free thinking and joy of making researches.

Thesis Documentation: Electronics eCommerce Platform

Abstract

This project presents the development of a full-stack eCommerce platform tailored for selling electronic products, featuring a user-friendly online store and a robust admin dashboard. Built using modern web technologies, the application leverages Next.js for the frontend, Node.js for the backend, and MySQL for data management. The platform is fully responsive, SEO-optimized, and designed to provide a seamless shopping experience. This thesis documents the design, implementation, and customization of the system, highlighting its technical architecture, modifications made to suit specific requirements, and the challenges overcome during development. The project serves as a practical demonstration of software engineering principles applied to real-world eCommerce solutions.

1. Introduction

1.1 Background

The rise of online shopping has transformed the retail industry, with eCommerce platforms becoming essential for businesses to reach global markets. Electronics, in particular, are a high-demand category, requiring robust platforms to handle product listings, customer interactions, and administrative tasks. This project aims to deliver a modern eCommerce solution that caters to both customers and administrators, ensuring scalability, performance, and ease of use.

1.2 Problem Statement

Traditional eCommerce platforms often lack flexibility for customization, scalability for growing businesses, or efficient admin tools for managing inventory and orders. Small businesses need cost-effective, open-source solutions that can be tailored to their needs without compromising on performance or user experience.

1.3 Objectives

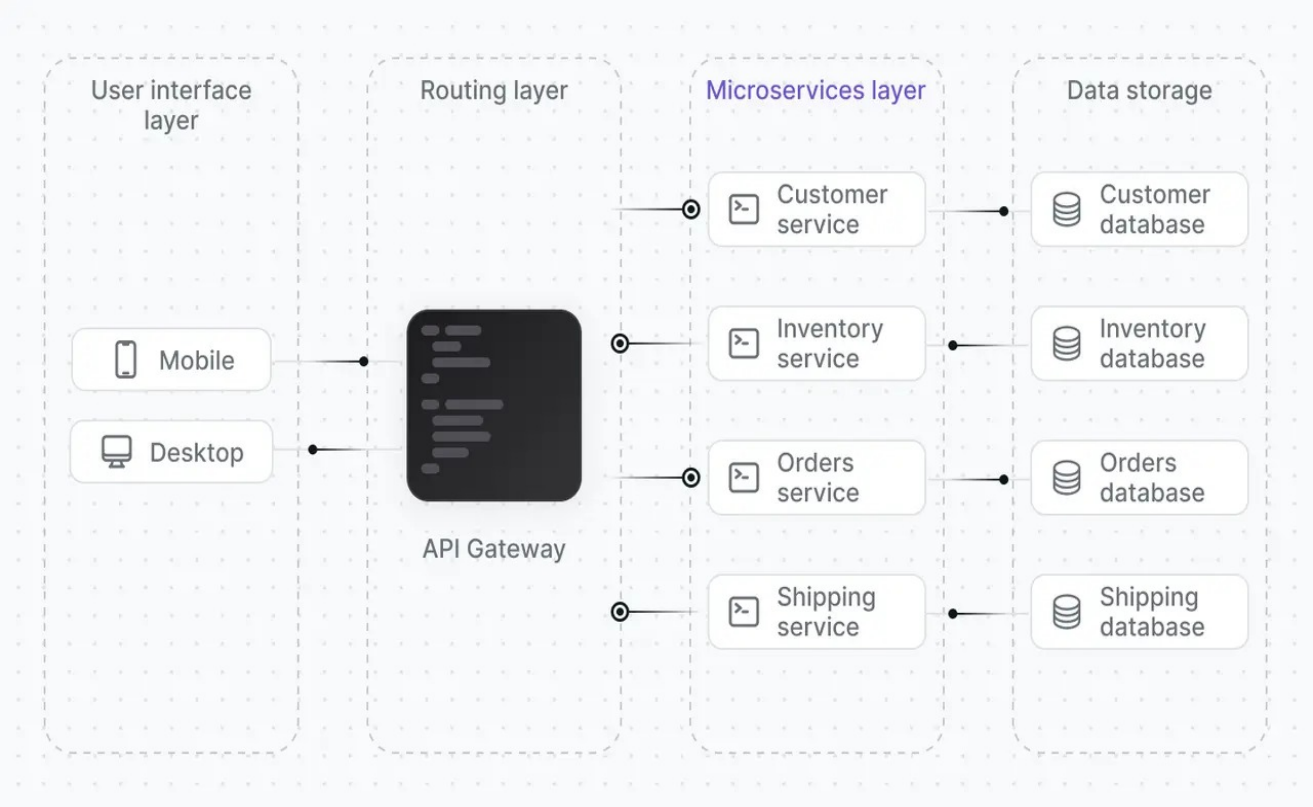
- Develop a responsive eCommerce website for selling electronic products.
- Implement a secure admin dashboard for managing products, orders, and users.
- Customize the platform to meet specific functional and aesthetic requirements.
- Ensure the system is scalable, SEO-friendly, and optimized for performance.
- Apply software engineering principles to design, develop, and test the application.

2. System Overview

2.1 System Architecture

The platform follows a client-server architecture with a decoupled frontend and backend:

- Frontend: Built with Next.js, utilizing Server-Side Rendering (SSR) and Static Site Generation (SSG) for performance and SEO.
- Backend: Powered by Node.js with Express.js, handling API requests and business logic.
- Database: MySQL, used for storing product details, user data, and order information.
- Authentication: NextAuth.js for secure user login and session management.
- Environment Configuration: Managed via a ".env" file for database credentials and API keys.



The system is divided into two main modules:

- Online Store: Allows customers to browse products, add items to the cart, and complete purchases.
- Admin Dashboard: Enables administrators to manage products, track orders, and view analytics.

2.2 Key Features

User Features:

- Product browsing with filters (category, price, brand).
- Shopping cart and checkout functionality.
- User authentication and profile management.
- Responsive design for mobile and desktop.

Admin Features:

- CRUD operations for products (Create, Read, Update, Delete).
- Order management and status tracking.
- User management and role-based access.
- Dashboard analytics for sales and inventory.

3. Technologies Used

The project leverages modern, industry-standard technologies to ensure performance, scalability, and maintainability:

- Next.js: A React framework for SSR, SSG, and API routes.
- Node.js & Express.js: For building a scalable backend API.
- MySQL: Relational database for structured data storage.
- HeidiSQL: Database management tool for ease of use.
- Tailwind CSS: Utility-first CSS framework for responsive styling.
- NextAuth.js: Authentication library for secure user sessions.

- Prisma: ORM for database interactions (optional, if used in your version).
- TypeScript: For type safety and improved code quality (if applicable).

4. Implementation

Next.js – **Next.js** is a **React framework** for building **modern web applications** with features like **server-side rendering (SSR)**, **static site generation (SSG)**, **API routes**, and **built-in routing**. Developed and maintained by **Vercel**, it's designed to make building **fast, SEO-friendly, and scalable** web apps easier and more efficient.

Why Use Next.js?

- Great **developer experience**
- Built-in **routing and APIs**
- **Fast performance** and **SEO-friendly**
- Works well with both **static** and **dynamic** content
- Easy to **deploy** and scale

Node.js – **Node.js** is an **open-source, cross-platform runtime environment** that allows developers to run **JavaScript** code **outside of a web browser**, typically on the **server**. It is built on **Chrome's V8 JavaScript engine** and enables the creation of fast, scalable, and efficient **backend services**.

Why Use Node.js?

- Full-stack JavaScript (frontend + backend)
- High performance for I/O-heavy apps
- Massive ecosystem via npm
- Excellent for building scalable network applications
- Community support and wide adoption

Express.js – Express.js (or simply Express) is a minimal and flexible Node.js web application framework that provides a robust set of features to build web and mobile applications, RESTful APIs, and backend services quickly and easily.

It simplifies the process of building server-side applications by providing a thin layer of fundamental web functionalities without hiding the core Node.js features.

Why Use Express?

- Simple and fast setup
- Highly customizable
- Large ecosystem and community
- Ideal for building scalable backend services with Node.js
- Works great for both **monolithic** and **microservice** architectures

MySQL – **MySQL** is an **open-source relational database management system (RDBMS)** that uses **Structured Query Language (SQL)** to manage and manipulate data. It is one of the most popular databases in the world and is widely used for storing structured data in **web applications**, **enterprise software**, and **data warehouses**.

Why Use MySQL?

- Easy to learn and use
- Supported by almost all hosting providers
- Widely adopted and well-documented
- Works well with languages like PHP, Java, Python, and JavaScript (Node.js)
- Scalable for small apps to enterprise-level systems

HeidiSQL – **HeidiSQL** is a **free**, **open-source** database management tool for **Windows**. It allows users to **connect to, browse, and manage** databases such as:

- **MySQL**
- **MariaDB**
- **PostgreSQL**
- **Microsoft SQL Server**
- **SQLite**

It provides a **graphical user interface (GUI)** to perform database operations easily, which would otherwise require writing raw SQL commands in the terminal or command-line interface.

Why Use HeidiSQL?

- User-friendly interface
- Lightweight and fast
- Supports multiple database systems
- Ideal for beginners and advanced developers
- Saves time for routine database management tasks

Tailwind CSS – **Tailwind CSS** is a **utility-first CSS framework** for rapidly building **modern, responsive** user interfaces. Instead of writing custom CSS, you use **predefined utility classes** directly in your HTML to style elements.

It's popular for allowing **complete control over design** without leaving your markup and is especially well-suited for **design systems**, **prototypes**, and **production-ready UI**.

Why Use Tailwind CSS?

- Speeds up development
- Keeps styling **consistent and maintainable**
- Works well with **modern frameworks**
- Encourages **component-based design**
- Easy to customize and scale for large projects

NextAuth.js – **NextAuth.js** is a **complete open-source authentication solution** built specifically for **Next.js applications**. It allows you to **easily implement authentication** (login, registration, sessions, etc.) using providers like **Google, GitHub, Facebook**, or **custom username/password**—with minimal configuration.

It handles **authentication, session management, and security** out of the box.

Why Use NextAuth.js?

- Seamless integration with Next.js
- No need to build auth from scratch
- Fully featured, yet lightweight
- Secure and production-ready
- Easily extensible with callbacks and events

Prisma – **Prisma** is a **next-generation ORM (Object-Relational Mapping)** for **Node.js** and **TypeScript**. It helps developers **easily interact with databases** (like MySQL, PostgreSQL, SQLite, and MongoDB) by providing a **type-safe**, **auto-completed**, and **developer-friendly API**.

Instead of writing raw SQL queries, Prisma lets you query your database using simple JavaScript or TypeScript functions that are **type-checked at compile time**.

Why Use Prisma?

Benefit	Description
☑ **Type-safe**	Catches errors at compile-time using TypeScript.
⚡ **Fast queries**	Built for performance and efficiency.
👨‍💻 **Developer-friendly**	Autocompletion, IntelliSense, and readable code.
🔁 **Migration system**	Easily version and apply DB schema changes.
🔎 **Prisma Studio**	GUI for managing and viewing DB content.

TypeScript – **TypeScript** is a **strongly typed, object-oriented programming language** that builds on **JavaScript** by adding **static type definitions**.

Developed and maintained by **Microsoft**, TypeScript is designed to help developers **write safer, more predictable, and scalable code**—especially in large codebases.

It compiles down to **plain JavaScript**, which means you can run it anywhere JavaScript runs (like in browsers or Node.js).

Why Use TypeScript?

- Reduces runtime errors
- Improves code maintainability
- Helps scale large projects
- Enhances collaboration in teams
- Better IDE support

4.1 Development Process

The development followed a structured software engineering approach:

- Requirements Analysis: Identified core features for customers and admins.
- System Design: Created UML diagrams (use case, class, and sequence diagrams) to model system interactions.
- Implementation: Developed frontend and backend modules iteratively.
- Testing: Conducted unit, integration, and manual testing for responsiveness and functionality.
- Deployment: Configured the application for local deployment with plans for cloud hosting.

Dependencies used as Frontend Frameworks: -

"@headlessui/react": "^1.7.18"

The dependency "@headlessui/react": "^1.7.18" is part of the **Headless UI** library built by the creators of Tailwind CSS. It's used in **React projects** to provide **completely unstyled, fully accessible UI components** that you can customize however you want. It helps you build custom UI components (like modals, dropdowns, tabs, switches, etc.) **without enforcing any styles**, giving you full control over appearance, while taking care of the **accessibility and behavior logic**.

"@prisma/client": "^5.10.2"

The dependency "@prisma/client": "^5.10.2" is the **Prisma Client** library, used in **Node.js and TypeScript** backend applications to interact with a **database**. @prisma/client is an **auto-generated, type-safe database client** that lets you query your database using JavaScript or TypeScript — typically alongside a **Prisma schema**.

Example –

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

const users = await prisma.user.findMany();

console.log(users);
```

"@tailwindcss/forms": "^0.5.7"

The dependency "@tailwindcss/forms": "^0.5.7" is a **Tailwind CSS plugin** that provides **better default styling for form elements** like <input>, <select>, <textarea>, and <checkbox>. Tailwind by default applies **minimal styling to form elements**, and they can look inconsistent across browsers. This plugin adds **opinionated, consistent base styles** to forms that fit well with Tailwind's design system.

Example –

Without the plugin:

```
<input class="border" />
```

With the plugin:

```
<input class="form-input" />
```

"@tailwindcss/typography": "^0.5.10"

The dependency **"@tailwindcss/typography": "^0.5.10"** is a Tailwind CSS plugin (also known as Typography plugin or @tailwindcss/typography) used to style long-form content like blog posts, articles, documentation, or markdown-rendered text with beautiful, readable typography. It provides a **prose** class that applies a set of sensible, pre-designed styles to plain HTML content — things like headings, paragraphs, blockquotes, lists, code blocks, and more — making them look clean and professional **without writing extra CSS**.

Example –

```
<article class="prose">
  <h1>Hello World</h1>
  <p>This is a paragraph with beautiful typography.</p>
  <blockquote>This is a blockquote.</blockquote>
</article>
```

"@types/bcryptjs": "^2.4.6"

The dependency "@types/bcryptjs": "^2.4.6" is a **TypeScript type declaration** package for the bcryptjs library. This package provides **TypeScript type definitions** for bcryptjs, which itself is a **JavaScript library for hashing and comparing passwords** using the bcrypt algorithm.

Since bcryptjs is a JavaScript library (not written in TypeScript), this @types package enables TypeScript projects to:

- Use **autocompletion**
- Get **type checking**
- Avoid TypeScript errors when calling bcryptjs functions

Example –

```
import bcrypt from 'bcryptjs';


const password = 'mySecret123';
const hashed = bcrypt.hashSync(password, 10);


const isMatch = bcrypt.compareSync('mySecret123', hashed);
```

"bcryptjs": "^2.4.3"

The dependency "bcryptjs": "^2.4.3" is a **JavaScript library** used for **hashing and comparing passwords** using the **bcrypt algorithm** — especially in Node.js applications. bcryptjs is a **pure JavaScript implementation** of the bcrypt hashing function. It's commonly used for:

- Securely storing passwords (by hashing them before saving to the database)
- Verifying login attempts (by comparing the plain password to the stored hash)

Example –

```
const bcrypt = require('bcryptjs');
const password = 'mySecret123';


// Hashing
const hash = bcrypt.hashSync(password, 10); // 10 = salt rounds
// Comparing
const isMatch = bcrypt.compareSync('mySecret123', hash);
console.log(isMatch); // true
```

"express-fileupload": "^1.5.0"

The dependency "express-fileupload": "^1.5.0" is a middleware for **Express.js** that enables your app to **handle file uploads** via HTTP requests — such as images, PDFs, documents, or any binary files. It parses incoming multipart/form-data requests and makes uploaded files available under req.files, allowing easy handling of file uploads in Express routes.

Example –

```
const express = require('express');
const fileUpload = require('express-fileupload');
const app = express();
app.use(fileUpload());
app.post('/upload', (req, res) => {
  if (!req.files || !req.files.myFile) {
    return res.status(400).send('No file uploaded.');
  }
  const myFile = req.files.myFile;
myFile.mv('./uploads/' + myFile.name, function(err) {if (err) return
```

"flowbite-react": "^0.7.2"

The dependency "flowbite-react": "^0.7.2" is a **React component library** built on top of **Tailwind CSS** and the **Flowbite UI kit**, offering a wide range of **pre-styled UI components** for React applications. It provides **ready-made, responsive, and accessible UI components** (like buttons, modals, navbars, tables, dropdowns, etc.) that are styled using Tailwind CSS, helping developers build modern UIs faster in React.

Example –

```
import { Button } from 'flowbite-react';
function App() {
  return <Button color="blue">Click me</Button>;
}
npm install flowbite-react flowbite
plugins: [
  require('flowbite/plugin')
],
content: [
  "./node_modules/flowbite-react/**/*.{js,jsx,ts,tsx}",
  // your own source paths
]
```

"nanoid": "^5.0.6"

The dependency "nanoid": "^5.0.6" is a **tiny, secure, and fast ID generator** used to create **unique string identifiers** — commonly for things like user IDs, URLs, database keys, or filenames. nanoid generates **collision-resistant unique IDs** that are URL-safe and cryptographically strong. It's a modern alternative to UUIDs (uuid package) but:

- **Shorter** (default ~21 characters)
- **Faster** and smaller in size
- **More customizable**

Example –

```
import { nanoid } from 'nanoid';


const id = nanoid(); // e.g., "V1StGXR8_Z5jdHi6B-myT"
console.log(id);
const shortId = nanoid(10); // e.g., "IRFa-VaY2b"
```

"next": "14.1.0"

The dependency "next": "14.1.0" refers to **Next.js**, a popular **React framework** for building **server-side rendered (SSR)** web applications, static websites, and APIs with built-in support for **routing**, **code splitting**, **performance optimization**, and **developer experience**.

Next.js is known for its ability to simplify the process of building React applications with features like SSR, static generation (SSG), API routes, and automatic code splitting.

Next.js is a full-fledged framework that enhances React with powerful features for production-grade applications, such as:

- **Server-side rendering (SSR)** for better SEO and faster load times
- **Static site generation (SSG)** for optimized, fast websites
- **API routes** to handle backend logic within the same codebase
- **File-based routing** (pages are automatically routed based on their file names)
- **Automatic code splitting** and **optimized loading** for performance
- **Image optimization** (via the next/image component)
- **Internationalization (i18n)** for multilingual apps

Example –

```
import { useEffect, useState } from 'react';
export default function Home() {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch('/api/hello')
      .then((res) => res.json())
      .then((data) => setData(data));
  }, []);
```

"next-auth": "^4.24.6"

The dependency "next-auth": "^4.24.6" is a popular authentication library for **Next.js** applications. It simplifies adding **authentication and authorization** to your Next.js app, supporting multiple authentication providers, such as email/password, OAuth (Google, Facebook, etc.), and custom authentication mechanisms. next-auth provides an easy way to manage **user authentication**, **sessions**, and **security** in your Next.js applications. It integrates seamlessly with Next.js API routes and supports a wide variety of authentication strategies and features such as:

- **OAuth Providers** (Google, GitHub, Facebook, etc.)
- **Email-based authentication** (magic link login)
- **Custom authentication** (e.g., username and password)
- **Session management** with JWT or database sessions
- **Role-based access control (RBAC)**
- **Secure password hashing**

Example –

```
npm install next-auth
import NextAuth from 'next-auth';
import GoogleProvider from 'next-auth/providers/google';
export default NextAuth({
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),
    // Add more providers here
  ],
```

```
  pages: {
    signIn: '/auth/signin',  // Custom sign-in page
  },
});
import { useSession, signIn, signOut } from 'next-auth/react';
export default function Page() {
  const { data: session } = useSession();
  if (session) {
    return (
      <div>
        <p>Welcome, {session.user.name}</p>
        <button onClick={() => signOut()}>Sign out</button>
      </div>
    );
  }
  return <button onClick={() => signIn('google')}>Sign in with Google</button>;
}
```

"react": "^18"

The dependency "react": "^18" refers to **React version 18**, a major release of **React** — a popular JavaScript library for building user interfaces, particularly **single-page applications (SPAs)**. React 18 introduces several new features and improvements, especially around **concurrent rendering** and **suspense**, making React applications faster and more responsive. It builds on React's component-based architecture to allow developers to create interactive UIs efficiently.

Example –

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>React 18 Example</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <p>Count: {count}</p>
    </div>
  );
}
ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

"react-apexcharts": "^1.4.1"

The dependency "react-apexcharts": "^1.4.1" is a **React wrapper** for the **ApexCharts** library, which is used to create interactive and responsive charts and visualizations in web applications. react-apexcharts provides an easy way to integrate **ApexCharts** (a powerful charting library) into your React applications. It allows you to display various types of charts, such as:

- Line charts
- Bar charts
- Pie charts
- Area charts
- Radar charts
- Donut charts
- Heatmaps, and more!

It comes with a rich set of customization options and is designed to be responsive, so the charts will adjust based on the size of the container.

Example –

```
import React from 'react';
import ApexCharts from 'react-apexcharts';
const ChartComponent = () => {
  const options = {
    chart: {
      id: 'basic-bar',
    },
    xaxis: {
      categories: ['January', 'February', 'March', 'April'],
    },
  };
  const series = [
    {
      name: 'Sales',
      data: [30, 40, 35, 50],
    },
  ];
  return (
    <div>
      <ApexCharts options={options} series={series} type="bar" height={350} />
    </div>
  );
};
export default ChartComponent;
```

"react-dom": "^18",

The dependency "react-dom": "^18" is the **React DOM library** that provides methods to interact with the **DOM (Document Object Model)** and render React components in the browser. It's an essential part of the React ecosystem, allowing React to render content on the page, manage the component lifecycle, and efficiently update the UI. react-dom is responsible for:

- **Rendering React components** into the actual DOM.
- **Hydrating server-rendered pages** (important for server-side rendering (SSR)).
- Handling **React events** and **state updates** in the browser DOM.

It works alongside the react library (which manages the component logic and state) and ensures that changes in the React app are reflected in the browser UI.

Example –

```
import React from 'react';
import ReactDOM from 'react-dom';

const App = () => <h1>Hello, React 18!</h1>;

ReactDOM.createRoot(document.getElementById('root')).render(<App />);

import React from 'react';
import ReactDOM from 'react-dom';

function App() {
  return <h1>Hello, World!</h1>;
}
// React 18 usage: createRoot() instead of render()
ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

"react-hot-toast": "^2.4.1"

The dependency "react-hot-toast": "^2.4.1" is a lightweight and customizable notification library for React that allows you to display **toast notifications** in your React applications. These toasts are often used to show brief feedback messages (such as success, error, or informational messages) to users in a non-intrusive way. react-hot-toast provides an easy-to-use API for showing **toast notifications** that are:

- **Minimalistic**: Focused on showing notifications without cluttering the UI.
- **Customizable**: You can style the notifications, set durations, and position them anywhere on the screen.
- **Non-blocking**: Toasts show up briefly and disappear automatically, allowing users to continue interacting with the application.

Example –

```
import React from 'react';
import { Toaster, toast } from 'react-hot-toast';
const App = () => {
  const handleClick = () => {
    toast.success('This is a success message!');
  };
  return (
    <div>
      <button onClick={handleClick}>Show Toast</button>
      <Toaster /> {/* This is where the toast notifications will appear */}
    </div>
  );
};
export default App;
import React from 'react';
import { Toaster, toast } from 'react-hot-toast';
const App = () => {
  return (
    <div>
      <button onClick={() => toast.success('Success!')}>Success</button>
      <button onClick={() => toast.error('Error!')}>Error</button>
      <button onClick={() => toast('Information message')}>Info</button>
      <Toaster />
    </div>
  );
};
export default App;
```

"react-icons": "^5.0.1"

The dependency "react-icons": "^5.0.1" is a package that provides a collection of **SVG icons** for React applications. It allows you to easily integrate a wide variety of popular icons from icon libraries (such as **Font Awesome**, **Material Design Icons**, **Feather**, and others) into

your React components. react-icons provides a simple way to use vector icons in your React application. The icons are imported as React components, which makes it easy to customize them (size, color, etc.) and integrate them into your UI. It supports a large collection of icons, including social media icons, UI elements, and other commonly used symbols.

Example –

```
import React from 'react';
import { FaBeer } from 'react-icons/fa'; // Importing a Font Awesome icon
const App = () => {
  return (
    <div>
      <h1>Let's go for a <FaBeer />!</h1>
    </div>
  );
};
export default App;

import React from 'react';
import { IoMdHome } from 'react-icons/io'; // Importing an Ionicons icon
const App = () => {
  return (
    <div>
      <button style={{ fontSize: '24px', color: 'blue' }}>
        <IoMdHome /> Home
      </button>
    </div>
  );
};
export default App;
```

"react-slick": "^0.30.2"

The dependency "react-slick": "^0.30.2" is a React wrapper for the popular **Slick Carousel** library. It enables you to create **responsive and customizable carousels** (also known as sliders) in your React applications. react-slick provides a set of tools to easily integrate a **carousel component** into your React app, which can be used to display a series of items (such as images, products, testimonials, etc.) in a scrollable, interactive, and visually appealing way. The carousel can support features like **autoplay**, **infinite scrolling**,

navigation arrows, and more.

Example –

```
npm install react-slick slick-carousel

import "slick-carousel/slick/slick.css";
import "slick-carousel/slick/slick-theme.css";
```

```
import React from 'react';
import Slider from 'react-slick'; // Importing the Slider component
import "slick-carousel/slick/slick.css";
import "slick-carousel/slick/slick-theme.css";
const SimpleSlider = () => {
  const settings = {
    dots: true,           // Show dots for navigation
    infinite: true,       // Infinite scrolling
    speed: 500,           // Transition speed in ms
    slidesToShow: 1,      // Number of slides to show at once
    slidesToScroll: 1     // Number of slides to scroll at once
  };
  return (
    <div>
      <h2> Simple Slick Carousel </h2>
      <Slider {...settings}>
        <div><img src="image1.jpg" alt="Slide 1" /></div>
        <div><img src="image2.jpg" alt="Slide 2" /></div>
        <div><img src="image3.jpg" alt="Slide 3" /></div>
      </Slider>
    </div>
  );
};
export default SimpleSlider;
```

"slick-carousel": "^1.8.1"

The dependency "slick-carousel": "^1.8.1" is the **core Slick Carousel library** that provides a highly customizable, responsive, and touch-enabled carousel/slider for displaying images, content, or any other type of media in a sliding format. It is the underlying library used by react-slick (the React wrapper), but it can be used independently in regular JavaScript or

jQuery-based projects. slick-carousel provides the functionality for building **carousels** and **sliders** with advanced features such as:

- **Responsive layouts**
- **Infinite looping** of content
- **Autoplay** support
- **Custom navigation controls**
- **Variable width/height** slides
- **Lazy loading** of images and content

Example –

```
npm install slick-carousel
import 'slick-carousel/slick/slick.css';
import 'slick-carousel/slick/slick-theme.css';
<div class="slider">
  <div><img src="image1.jpg" alt="Slide 1"></div>
  <div><img src="image2.jpg" alt="Slide 2"></div>
  <div><img src="image3.jpg" alt="Slide 3"></div>
</div>
<script>
  $(document).ready(function(){
    $('.slider').slick({
      dots: true,          // Show navigation dots
      infinite: true,      // Infinite scrolling
      speed: 500,          // Transition speed
      slidesToShow: 1,     // Number of slides to show at once
      slidesToScroll: 1    // Number of slides to scroll at once
    });
  });
</script>
```

Since react-slick is just a React wrapper for slick-carousel, when you use react-slick, the actual functionality is powered by the slick-carousel package. This means that the core setup and configuration options remain the same regardless of whether you're using plain JavaScript or React.

"svgmap": "^2.10.1"

The dependency "svgmap": "^2.10.1" is a JavaScript library used for rendering and interacting with **SVG-based maps**. It allows developers to create interactive, dynamic, and customizable maps using **Scalable Vector Graphics (SVG)**, which are particularly useful for representing geographical regions, data visualizations, or custom areas on a web page.
svgmap enables you to:

- Render **SVG maps** of countries, regions, or custom areas.
- Add **interactive features** such as tooltips, hover effects, and clickable areas.
- **Customize styles** (e.g., colors, borders) of different map regions.
- **Integrate data** dynamically into map regions (e.g., population statistics, weather data).
- **Add events** to map regions like clicks, hover actions, and more.

Example –

```
npm install svgmap
<div id="map-container"></div>
<script src="path-to-svgmap.js"></script>
<script>
  const map = new SvgMap({
    target: "#map-container", // Target DOM element
    data: {                   // Map data (regions with custom data)
      regions: [
        { id: "region1", name: "Region 1", value: 10 },
        { id: "region2", name: "Region 2", value: 20 },
        { id: "region3", name: "Region 3", value: 30 }
      ],
    },
    color: "lightblue",        // Default color for regions
    hoverColor: "lightgreen",  // Hover color
    selectedColor: "orange",   // Color when a region is selected
  });
  map.render(); // Render the map
</script>
```

"zod": "^3.22.4"

The dependency "zod": "^3.22.4" is a **TypeScript-first schema validation library** that helps with **data validation**, **parsing**, and **type inference**. It provides a type-safe approach to validate and process data structures like objects, arrays, strings, numbers, and more. Zod is designed to validate data objects at runtime while ensuring **type safety** with TypeScript. It's particularly useful when you need to ensure that the data passed to your application matches the expected schema, making your code more reliable and reducing runtime errors.

Example –

```
npm install zod
import { z } from 'zod';
// Define a schema
const userSchema = z.object({
  name: z.string().min(1),    // Name must be a non-empty string
  age: z.number().int().positive(),  // Age must be a positive integer
});
// Validate data against the schema
const userData = {
  name: 'John',
  age: 30,
};
```

"zustand": "^4.5.1"

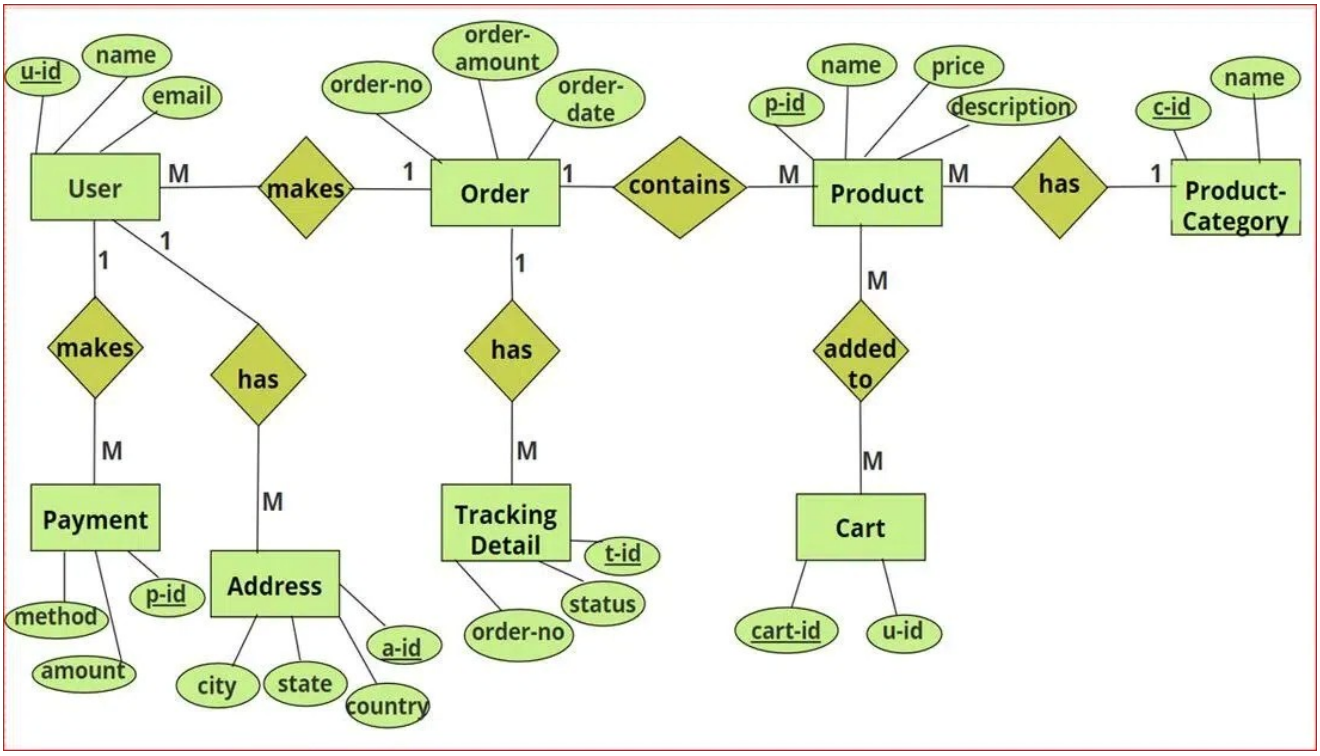
The dependency "zustand": "^4.5.1" is a **small, fast, and scalable state management library** for React applications. It provides a simple and intuitive way to manage global or shared state in a React app, without the need for boilerplate code or complex setup. Zustand is designed to provide a **minimalistic** approach to state management while maintaining **high performance** and ease of use. It is particularly useful for managing **global state** that needs to be accessed across different parts of the application, such as user authentication status, theme settings, or app configurations.

- **Minimalistic API**: Zustand has a simple API that requires very little boilerplate code.
- **Performance**: Zustand is designed to be **fast and efficient**, making it suitable for even larger applications.
- **React hooks**: It uses React hooks (useStore) to provide access to the state, making it very intuitive to use with functional components.
- **Persistence**: With middleware, you can easily persist state across sessions using storage like localStorage.
- **Composability**: You can easily combine different state logic into one store and share it across different components.
- **No provider wrapping**: Unlike other state management solutions (e.g., Redux or Context API), Zustand does not require you to wrap your app in any provider component.

Example –

```
npm install zustand
import create from 'zustand';
// Define a store with state and actions
const useStore = create((set) => ({
  count: 0, // state
  increment: () => set((state) => ({ count: state.count + 1 })), // action
  decrement: () => set((state) => ({ count: state.count - 1 })), // action
}));
function Counter() {
  const { count, increment, decrement } = useStore();
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
    </div>
  );
}
export default Counter;
const useStore = create((set) => ({
  count: 0,
  user: null,
  theme: 'light',
  increment: () => set((state) => ({ count: state.count + 1 })),
  setUser: (user) => set(() => ({ user })),
  toggleTheme: () => set((state) => ({ theme: state.theme === 'light' ? 'dark' : 'light'
})),
}));
```

ER Diagram: -



Challenges and Solutions

5.1 Challenge: Database Configuration

Issue: Setting up MySQL and ensuring proper connection with the application was complex for beginners.

Solution: Used HeidiSQL for a user-friendly interface and documented the `.env` setup process clearly.

5.2 Challenge: Responsive Design

Issue: Ensuring the UI was consistent across devices required extensive testing.

Solution: Leveraged Tailwind CSS's responsive utilities and tested on multiple screen sizes using browser dev tools.

5.3 Challenge: Authentication Security

Issue: Implementing secure user authentication without vulnerabilities.

Solution: Used NextAuth.js with JWT-based sessions and stored sensitive data in environment variables.

5.4 Challenge: Performance Optimization

Issue: Initial page load times were slow due to large product images and unoptimized queries.

Solution: Implemented lazy loading, optimized database queries, and used Next.js's built-in image optimization.

Limitations

- Payment gateway integration (e.g., Stripe) is not implemented but planned for future iterations.
- Limited support for internationalization (multi-language support).
- Cloud deployment is not fully configured, currently running locally.

Achievements

- Developed a fully functional eCommerce platform with a responsive design.
- Implemented a secure and scalable admin dashboard.
- Customized the application to meet specific functional and aesthetic goals.
- Achieved fast load times and SEO-friendly pages using Next.js features.
- Successfully tested the application for usability and performance.

Future Enhancements

- Integrate a payment gateway for real transactions.
- Add multi-language support for global accessibility.
- Deploy the application on a cloud platform like Vercel or AWS.
- Implement AI-based product recommendations.

THE PRODUCT OF THE FUTURE

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolor modi iure laudantium necessitatibus ab, voluptates vitae ullam. Officia ipsam iusto beatae nesciunt, consequatur deserunt minima maiores earum obcaecati. Optio, nam!

BUY NOW LEARN MORE

INTRODUCING SINGITRONIC

Buy the latest electronics.
The best electronics for tech lovers.

SHOP NOW

BROWSE CATEGORIES

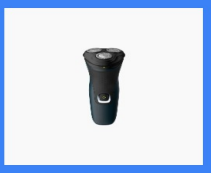
- Smart Phones
- Tablets
- Mouses
- Cameras
- Smart Watches
- Laptops
- PCs
- Printers
- Earbuds
- Head Phones

FEATURED PRODUCTS

SMART PHONE
$22
VIEW PRODUCT

SMART WATCH
$64
VIEW PRODUCT

NOTEBOOK HORIZON
$52
VIEW PRODUCT



MENS TRIMMER
$54
VIEW PRODUCT

SONY BLUETOOTH SPEAKER
$100
VIEW PRODUCT

DELAX 1800 PREMIUM
$50
VIEW PRODUCT

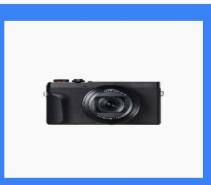
DELAX 2800 STANDARD
$60
VIEW PRODUCT

DELAX 800 PREMIUM
$100
VIEW PRODUCT

MING TABLET 1200
$100
VIEW PRODUCT

MING TABLET 2200
$130
VIEW PRODUCT

MING TABLET 5000
$230
VIEW PRODUCT



SLR CAMERA
$24
VIEW PRODUCT

Conclusion

This project marks the successful development and implementation of a robust and modern **eCommerce platform for electronics**, designed to meet the growing needs of online consumers and small-to-medium-scale businesses. The platform incorporates a fully responsive **online storefront** that ensures a seamless shopping experience across devices, along with a **comprehensive admin dashboard** that simplifies inventory and user management, order tracking, and analytics.

Leveraging powerful technologies such as **Next.js** for the frontend, **Node.js** for the backend, and **MySQL** for data persistence, the application demonstrates a well-architected, high-performance solution that is scalable, secure, and user-centric. The use of **NextAuth.js** for authentication and **Tailwind CSS** for styling contributed significantly to both the visual appeal and the security of the platform.

From a technical standpoint, the project followed a structured **software engineering lifecycle**—beginning with requirements gathering, followed by system design, iterative implementation, thorough testing, and successful deployment in a local development environment. Tools like **Prisma ORM** and **HeidiSQL** helped streamline database interactions, while component libraries such as **Flowbite React** and **react-slick** enriched the user interface and interactivity.

The platform is highly modular and customizable, supporting features such as product categorization, cart management, dynamic pricing, admin control panels, and role-based access control. These capabilities ensure the application can adapt to various business requirements and market dynamics with minimal rework.

Moreover, the challenges encountered during development—such as responsive design issues, database connectivity, and session security—were effectively addressed using industry best practices. This added significant real-world value to the learning experience, reinforcing concepts such as full-stack integration, authentication flows, performance optimization, and error handling.

In addition to technical achievements, the project highlights a forward-looking vision with proposed **future enhancements** like payment gateway integration, multilingual support, AI-based product recommendations, and cloud deployment. These potential extensions underline the platform's flexibility and growth potential.

In conclusion, the **Electronics eCommerce Platform** stands as a functional, scalable, and future-ready solution. It not only meets the functional and aesthetic needs of modern eCommerce but also serves as a tangible demonstration of software engineering principles in action. This thesis encapsulates the development journey, technical insights, and implementation strategies, equipping the developer with the confidence and experience to pursue further advancements in the field of web application development and eCommerce technologies.

Project Details

Project Source Code URL: -

https://github.com/Amit-Sharma-0004/EndTermProject.git

QR Code for easy access of source code & project thesis with presentation –

References

- Next.js Documentation: nextjs.org
- Node.js Documentation: nodejs.org
- MySQL Documentation: mysql.com
- Tailwind CSS Documentation: tailwindcss.com
- NextAuth.js Documentation: next-auth.js.org

THE END

========== o O o ==========