

# **RESPONSIVE E-COMMERCE WEBSITE**

## **A PROJECT REPORT**

*Submitted by*

Shashank Mudgal(211b287)

Shivam Swaraj(211b292)

Stuti Jain(211b319)

**Under the guidance of: Dr. Prateek Pandey**



July 2023-December 2023

*Submitted in partial fulfillment for the award of the degree*

*Of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE**

**Department of Computer Science & Engineering**

**JAYPEE UNIVERSITY OF ENGINEERING &**

**TECHNOLOGY, AB ROAD, RAGHOGARH, DT. GUNA-473226**

**MP, INDIA**

### **Declaration by the Student**

I hereby declare that the work reported in the B. Tech. project entitled as **RESPONSIVE E-COMMERCE WEBSITE**, in partial fulfillment for the award of degree of B.Tech submitted at Jaypee University of Engineering and Technology, Guna, as per best of my knowledge and belief there is no infringement of intellectual property right and copyright. In case of any violation I will solely be responsible.

Shashank Mudgal(211b287)

Shivam Swaraj(211b292)

Stuti Jain(211b319)

Department of Computer Science and  
Engineering  
Jaypee University of Engineering and  
Technology

Guna, M.P., India

Date: 28/11/2023



## JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY

Grade 'A+' Accredited with by NAAC & Approved U/S 2(f) of the UGC Act,  
1956A.B. Road, Raghogarh, Dist: Guna (M.P.) India, Pin-473226

Phone: 07544 267051, 267310-14, Fax: 07544 267011

Website: [www.juet.ac.in](http://www.juet.ac.in)

### CERTIFICATE

This is to certify that the work titled “**RESPONSIVE E-COMMERCE WEBSITE**” submitted by “**SHASHANK MUDGAL(211b287), SHIVAM SWARAJ(211b292) , STUTI JAIN(211b319)**” in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering of Jaypee University of Engineering & Technology, Guna has been carried out under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property right and copyright. Also, this work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma. In case of any violation concern student will solely be responsible.

Signature of Supervisor :

Name of Supervisor : Dr. Prateek Pandey

Designation:

Date:

## ACKNOWLEDGEMENT

Any endeavor cannot lead to success unless and until a proper platform is provided for the same. This is the reason why: we find ourselves fortunate to complete our work on a minor project under the supervision of Dr. Prateek Pandey. Our sincere gratitude to him for having faith in us and thus allowing us to carry out a project on technology completely new to us, for which we had to research and learn many new things, which will help us deal with advanced work in future. He helped immensely by guiding us throughout the project in any possible ways he could. Last but not the least, we would like to thank Dept. Of Computer Science & Engineering who created this opportunity

Name of Student .....  
Signature of the student .....  
Date .....

Name of Student .....  
Signature of the student .....  
Date .....

Name of Student .....  
Signature of the student .....  
Date .....

## **SUMMARY**

In today's generation, most people are using technology for leading their lives and fulfilling their daily needs. In this generation most of us using E-Commerce websites for shopping for clothes, groceries, and electronics. We have developed one E-commerce web application by using MERN stack technology as it contains MongoDB, Express.JS library, Node.JS platform. This application is fully functional with different views for user and admin and it also has integrated with payment gateway for checkout. By using this website we can buy different types of t-shirts and we can choose different products and can delete them also. We have developed administrative functions for the website such as create a product, create categories, Admin dashboard Manage products, Manage categories. For customers, they can quickly add their items to the cart. Based on the items in the cart the bill gets generate and the customer can pay by using stripe.

**KEYWORDS:** JavaScript, Software Stack, Framework, Library, Performance analysis, React. Js, mongoDB, Node.js, Express.js.

# Table of Contents

List of Abbreviations.

1. Introduction
  2. Application Infrastructure – The MERN stack
    - 2.1 MERN Details
      - 2.1.1. Node.js
      - 2.1.2. Express.js
      - 2.1.3. MongoDB
      - 2.1.4. React.js
    - 2.2. How Does MERN stack work?
    - 2.3 React.js Frontend
    - 2.4 Mongoddb databases
  3. Application Implementation
    - 3.1.Application Requirements
    - 3.2 Application Development
      - 3.2.1 Front-end development
        - 3.2.1.1 Basic setup and Routing
        - 3.2.1.2 User Authentication
        - 3.2.1.3 Dashboard Component
        - 3.2.1.4 Home Component
        - 3.2.1.5 Shop Component
        - 3.2.1.6 Cart Component and Payment Gateway
      - 3.2.2 Back-end development
        - 3.2.2.1 Basic Setup
        - 3.2.2.2Monguse Schema Creation
        - 3.3.2.3 Routing and API documentation
  - 4 .Conclusion
- Referenc*

## **List of Abbreviations**

1. ASP :Active Server Pages
2. JSON :JavaScript Object Notation
3. PHP: Hypertext Preprocessor
4. MERN: MongoDB, Express, React.js, Node.js J
5. HTTP: Hypertext Markup Language
6. CSS: Cascading Style Sheets
7. REST: Representational State Transfer
8. API :Application programming interface
9. URL :Uniform Resource Locator
10. NPM: Node Package Manager
11. NoSQL :Non-Structured Query Language
12. MVC: Model View Controller
13. UI: User Interface
14. DOM :Object Data Modeling

# Chapter 1: INTRODUCTION

we all know that technology has become an essential tool for online marketing these days. If we see all over the world most of the people are showing interest to buy things in online. However, we can see that there are many small shops and grocery stores are selling their things offline. With this type of selling most of us will face bad experience. For instance, in some shops seller had the product to sell in the offer but the buyer may not know about it, or the customer may need the product urgently then he will go the shop, but the product is out of stock, in the case, he will face bad experience. Moreover, in online shopping customers can select a wide range of products based upon their interests and their price also, one can compare prices also from one store to another by using online shopping.

By encountering the all problems and weaknesses of the offline shopping system, creating an E-commerce web application is necessary for searching and shopping in each shop. These days we have seen so many E-commerce websites are created like Flipkart, Amazon, Myntra one can easily buy their necessary products by using these websites. By using their products by staying in their home. Eventually, we can see the difference between the prices of products also as if we see the cost of the product will be slightly high in offline shopping when compared to online shopping.

For creating these types of E-commerce web applications MERN stack will be the best option that can help us for creating the most effective and powerful web applications.

The objectives of this project were to illustrate and understand the fundamental concepts and usage of each technology in the MERN stack, as well as their compatibilities and advantages as a complete stack in web application development. The project achieved that goal by utilizing these modern technologies and implementing a web application. The idea of this web application was toward a startup running by the author's parents as they decided to open a book retail store. By researching, ecommerce – an enormous platform is emerging at an extraordinary speed over the last decades all over the world and providing more advantages and conveniences as compared to physical stores. Ecommerce has changed permanently the way business and consumer interact, which allows users to connect with their favourite shops and brands whenever and wherever they want and also helps stores to

more actively approach consumers. It is believed that the growth of e-commerce for the next incoming years is increasing beyond measure rate with the release of modern technologies.





## Chapter 2: Application Infrastructure – The MERN stack

The MERN stack is basically a JavaScript-based stack which is created to facilitate the development process. MERN comprises of four open-source elements: MongoDB as the database, Express as server framework, React.js serves as client library and Node.js is an environment to run JavaScript on the server. These technologies introduce an end-to-end web stack for developers to utilize in web development. MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack.

- MongoDB — document database
- Express(.js) — Node.js web framework
- React(.js) — a client-side JavaScript framework
- Node(.js) — the premier JavaScript web server

Express and Node make up the middle (application) tier. Express.js is a server-side web framework, and Node.js is the popular and powerful JavaScript server platform. Regardless of which variant you choose, ME(RVA)N is the ideal approach to working with JavaScript and JSON, all the way through.

### Node.js

Node.js is JavaScript environment provider and the most essential core of the MERN stack. Node.js is now the most widely used free open source web server environment created by Ryan Dahl. It allowed to execute JavaScript code on the server. It is able to run on multiple platforms like windows, Linux and mac OS. Node.js is dependent on Google V8 engine which is the core of Chrome browser. C and C++ run both Node and V8 which is better in performance speed and memory consumption.

How node handles the request:

- Sends the task to the computer's file system .
- Ready to handle the next request.

- When the file system is open and read the file, the server returns the request to the client.

## Express.js

Express is a micro and flexible prebuilt framework based on Node that can provide faster and smarter solution in creating server-side web applications. Express is made of Node so it inherits all Node's features like simplicity, flexibility, scalability and performance as well. In brief, what Express does to Node is the same as Bootstrap does to HTML/CSS and responsive design. It makes programming in Node a piece of cake and provides developers some additional tools and features to improve their server-side coding. Express is literally the most famous Node framework so that whenever people mention about Node, they usually imply Node combined with Express. TJ Halewyck released Express the first time in 2010 and currently it is maintained by the Node foundation and developers who contribute to the open source code. Despite the fact that Express itself is completely minimalist, developers have programmed many compatible middleware packages to solve almost all issues in web development. Express offers a quick and straightforward way to create a robust API with a set of helpful HTTP methods and middleware.

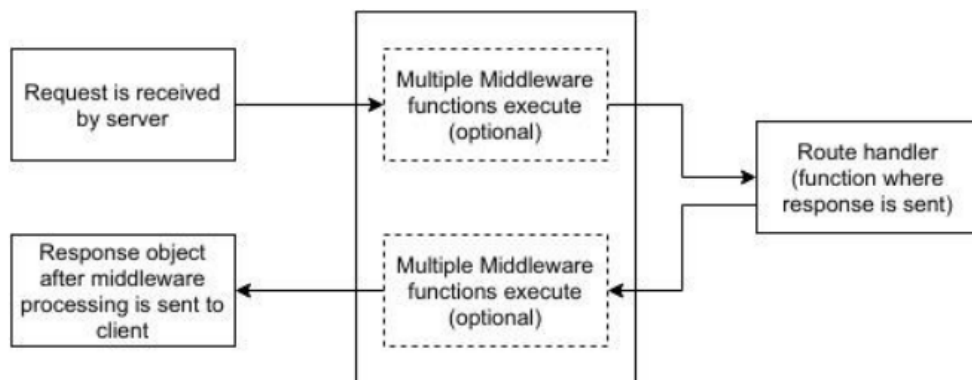


Diagram of middleware execution order

## MongoDB

MongoDB is a document-based NoSQL database used mainly for scalable high-volume data applications and data involved jobs which does not work well in a relational model. It is among the most popular non-relational database which emerged in the mid-2000s. MongoDB

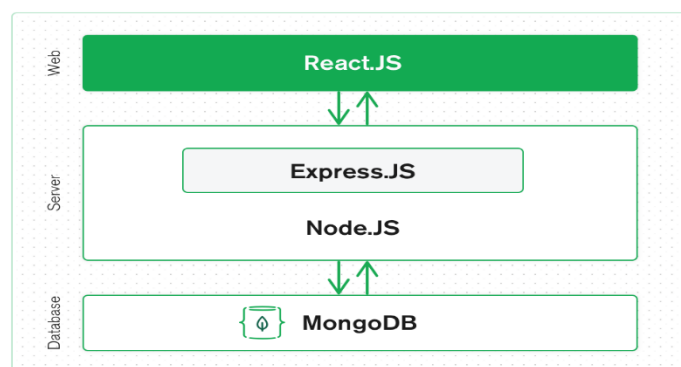
architecture comprises of collections and documents replacing the use of tables and rows from traditional relational databases point of view. One of the most essential functionalities of MongoDB is its ability to store dynamic data in flexible BSON documents, which means documents sharing the same collection can have different fields and key-value pairs, while data structure can be changed without any restrictions at any time

## React.js

React is an open-source front-end JavaScript-based library that specializes in building user interfaces. It was originally created by a Facebook software engineer and first implemented in newsfeed feature of Facebook in 2011 and followed by Instagram a year later. Due to the minimal understanding requirement of HTML and JavaScript, React is easy to learn and thanks to the support by Facebook and strong community behind it, React expands its popularity and becomes one of the most used JavaScript library

## How does the MERN stack work?

The MERN architecture allows you to easily construct a three-tier architecture (front end, back end, database) entirely using JavaScript and JSON.



## React.js front end

The top tier of the MERN stack is React.js, the declarative JavaScript framework for creating dynamic client-side applications in HTML. React lets you build up complex interfaces through simple components, connect them to data on your back-end server, and render them as HTML.

React's strong suit is handling stateful, data-driven interfaces with minimal code and minimal pain, and it has all the bells and whistles you'd expect from a modern web framework: great support for forms, error handling, events, lists, and more.

### **MongoDB database tier**

If your application stores any data (user profiles, content, comments, uploads, events, etc.), then you're going to want a database that's just as easy to work with as React, Express, and Node.

That's where MongoDB comes in: JSON documents created in your React.js front end can be sent to the Express.js server, where they can be processed and (assuming they're valid) stored directly in MongoDB for later retrieval.

## **Chapter 3: Application Implementation**

A prototype version of the e-commerce application was created to apply the study of MERN stack as well as to understand how they contribute to the whole entity in web development

### **3.1 Application requirements**

Typically, a (business to customer) B2C e-commerce web application has two types of users, which are admin and user. Admins are responsible for some specific management tasks such as creating, updating and removing the products from the database as well as managing user orders. A user can browse and read product's information displayed in the application. He can also add the product to the shopping cart and perform the payment for that product. While some resources and web routes are public, the others can only be accessed by an admin or a signed in user.

### **3.2 Application development**

This section is dedicated to demonstrate the functionalities development process from back-end to front-end of the e-commerce application. The project structure is divided into 2 folder, ecommerce and ecommerce-front which contain the source code of back-end and frontend respectively.

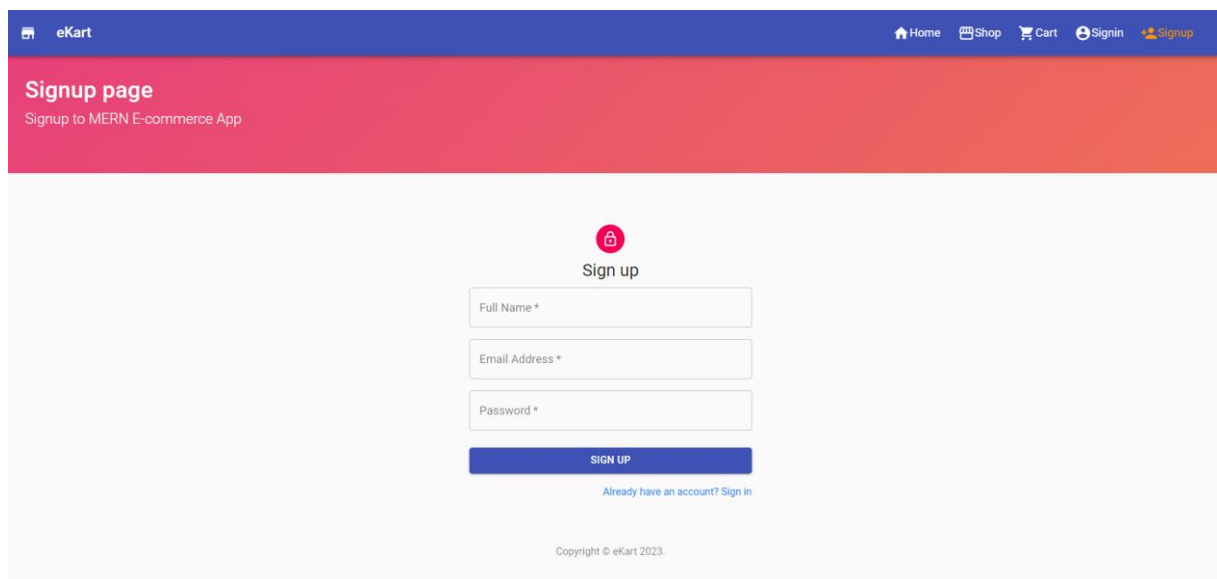
#### **3.2.1 Front-end development**

##### **3.2.1.1 Basic setup and Routing**

Unlike back-end is performed with multiple technologies, front-end development is handled by only React.js. Facebook has developed Create React App node module for quickly scaffolding a React template, which saves developers from all those complicated configurations. Create React App not only generates a boilerplate with core structure for a React application but also takes care of some build script and development server. After running command 'npx create-react-app' to generate a template, bootstrap – the most popular CSS framework is added to minimize styling tasks from scratch since the author want to focus on the development part using MERN stack. Then Routes.js file is created with the sole purpose of including all the route paths with their corresponding components by using a package called react-router-dom. Route component is then rendered by React Dom as the root component of the application.

### 3.2.1.2 User Authentication Sign up

Component Although some main pages of this application like Home page, Shop page do not require the user to sign up in order to enhance user experience, if users want to make a purchase, they have to sign up and log in to their account first. Below is a simple user interface to sign up new users.

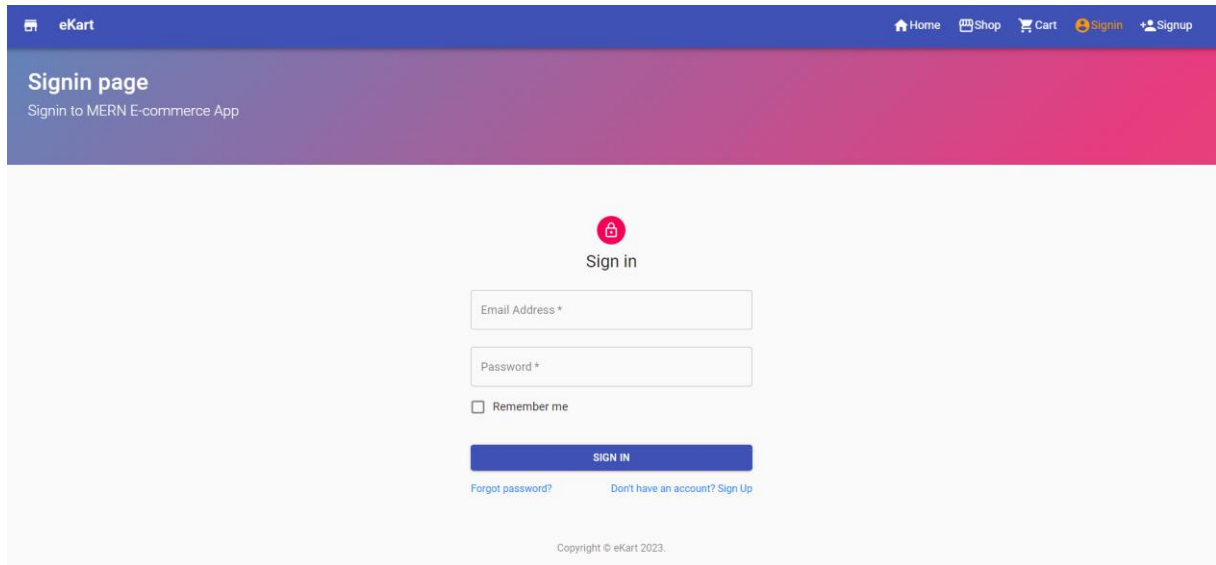


The image shows a web application interface for signing up. At the top, there is a blue navigation bar with the 'eKart' logo and links for Home, Shop, Cart, Signin, and Signup. Below this is a red banner with the text 'Signup page' and 'Signup to MERN E-commerce App'. The main content area is white and features a 'Sign up' form. The form consists of three input fields: 'Full Name \*', 'Email Address \*', and 'Password \*'. Below these fields is a blue button labeled 'SIGN UP'. At the bottom of the form, there is a link that says 'Already have an account? Sign in'. The footer of the page indicates 'Copyright © eKart 2023'.

Figure displays a web page with shared navigation bar and layout. Right below is the signup form which includes some inputs for name, email and password with a submit button. Bootstrap class 'form-group' is applied to facilitate these styling tasks. In this component, beside the state of each input field there are error and success state which determine if some alerts should be shown up when user fail or succeed to sign up. Given the user put in correct data and press submit, fetch method is used to send a POST request with user data under json format in the body to the back-end API 'api/signup'. After validation process, a new instance of user model is created based on req.body and is saved into the database. The server then returns a json file containing user data without hashed password back to the client. Finally, a success alert is shown up and is redirecting the user to the sign in page.

## Sign in Component

The sign in component is quite similar to the sign up one except the data flow when the user press submit button. After the server handled the POST request from the client, a token is signed and sent back to the client along with user information.



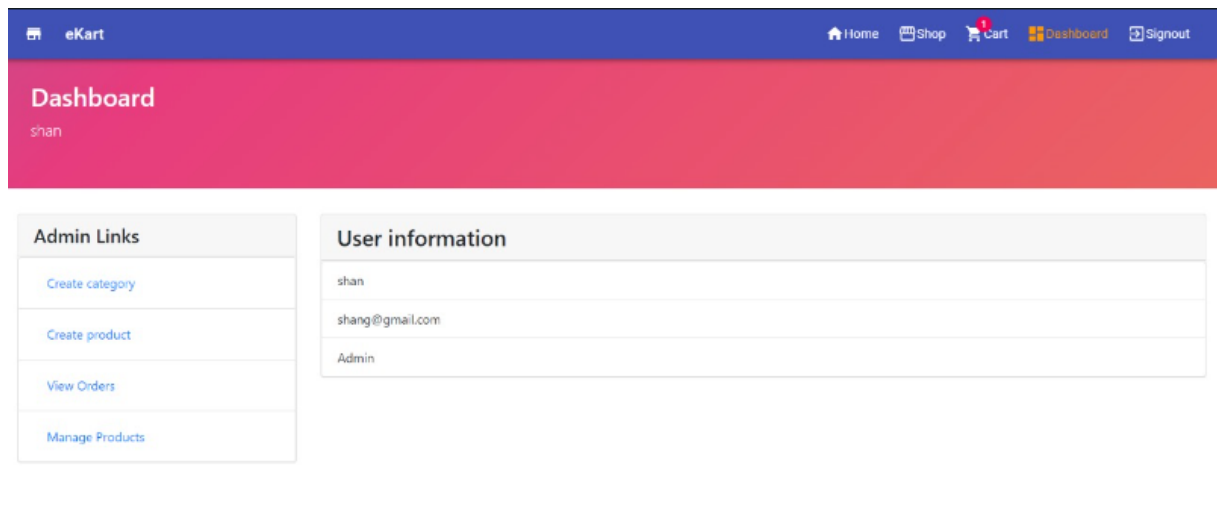
The screenshot shows the 'Signin page' of the eKart application. The page has a blue header with the eKart logo and navigation links for Home, Shop, Cart, Signin, and Signup. Below the header, there's a pink banner with the text 'Signin page' and 'Signin to MERN E-commerce App'. The main content area is white and contains a 'Sign in' form. The form includes a red lock icon, the text 'Sign in', two input fields for 'Email Address \*' and 'Password \*', a 'Remember me' checkbox, a blue 'SIGN IN' button, and two links: 'Forgot password?' and 'Don't have an account? Sign Up'. At the bottom, there's a copyright notice: 'Copyright © eKart 2023.'

### 3.2.2.3 Dashboard Component

#### Admin Dashboard

Admin dashboard is accomplished with this component as can be seen in figure below. It consists of 2 section elements which both apply Bootstrap class Card. All admin information is shown on the right side while some admin actions like creating category and product, viewing and managing order are on the left. By clicking on tasks link, admin will be navigated to the matching component where he/she can create a category or a product. The admin also has the ability to view orders and manage them, for example setting the order status from processing to shipped or delivered. Moreover, ManageProduct component enables the admin user to modify products or even delete them from the database.





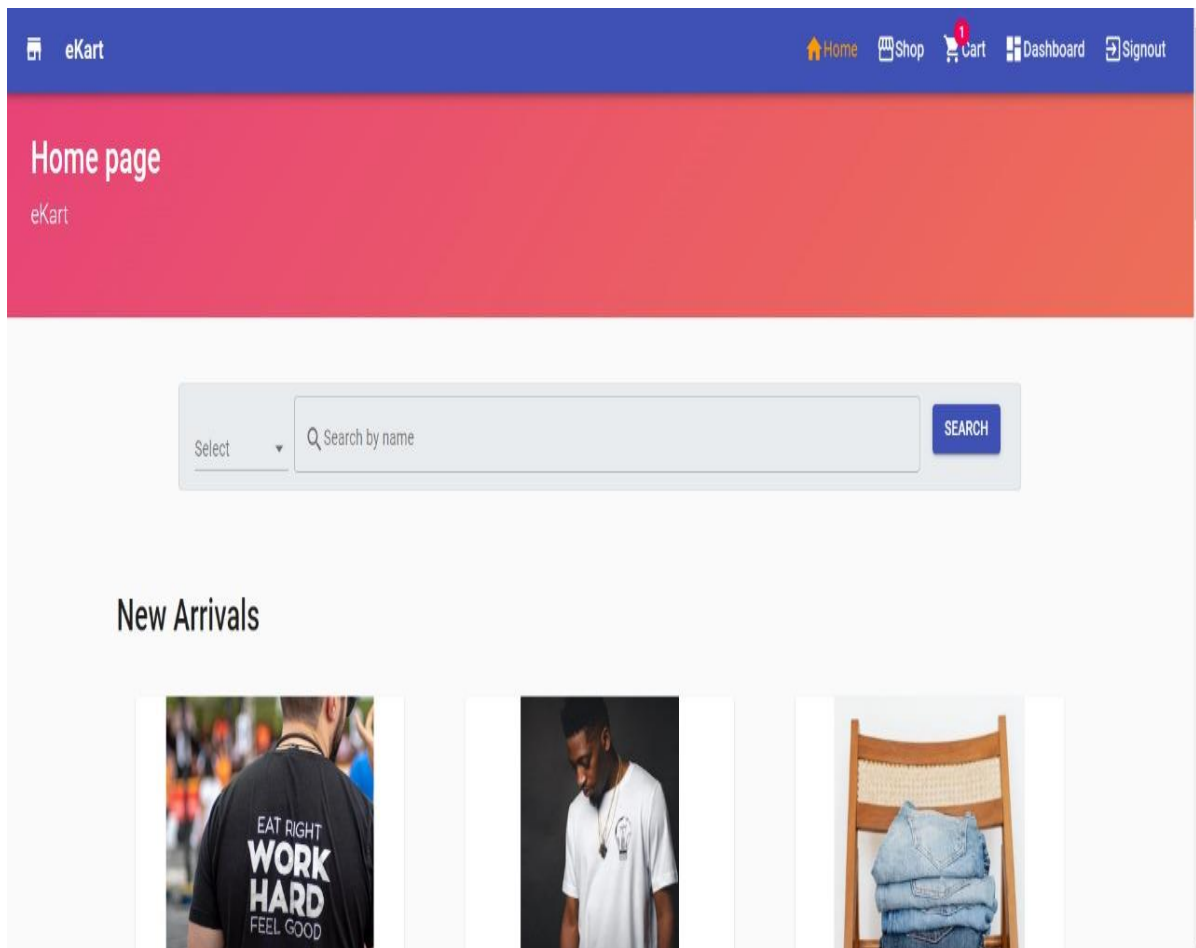
## User Dashboard

Generally, the user dashboard component is quite similar to the admin one. Since the user have less power than an admin, there is fewer actions for a user to perform. Checking out in the Cart page or updating their profiles are the two only user's tasks, as visualized in figure. User interface of User Dashboard Unlike admin dashboard, user's purchase history is presented under his/her personal information. This list provides a precise look of order history, such as product name, price and the time it was performed.

### 3.2.2.4 Home Component UI Description

Home page, which is the main page of this application is demonstrated by Home component. This component is accessible through a public empty URL. It plays an important role in an e-commerce application.

An overview of the Home page is described in figure . Some of the most fundamental functionalities of an e-commerce application are shown in this component. There is also a search engine rendered by the Search component on top of the page right below the shared



layout. Here user can search product based on category and by text in name.

## Logic and Data Flow

```
import React, { useState, useEffect } from 'react';
import Layout from './Layout';
import { getProducts } from './apiCore';
import Card from './Card';
import Search from './Search';
import 'fontsource-roboto';
import Copyright from './Copyright';

const Home = () => {
  const [productsBySell, setProductsBySell] = useState([]);
  const [productsByArrival, setProductsByArrival] = useState([]);
  const [error, setError] = useState('');

  const loadProductsBySell = () => {
    getProducts('sold').then((data) => {
      if (data.error) {
        setError(data.error);
      }
    });
  }
}
```

```

    } else {
      setProductsBySell(data);
    }
  });
};

const loadProductsByArrival = () => {
  getProducts('createdAt').then((data) => {
    if (data.error) {
      setError(data.error);
    } else {
      setProductsByArrival(data);
    }
  });
};

useEffect(() => {
  loadProductsByArrival();
  loadProductsBySell();
}, []);

return (
  <Layout
    title='Home page'
    description='MERN E-commerce App'
    className='container-fluid'
  >
    <Search />
    <div className='row'>
      <div className='col-md-1'></div>
      <div className='col-md-10'>
        <h2 className='mb-2'>New Arrivals</h2>
        <div className='row'>
          {productsByArrival.map((product, i) => (
            <div key={i} className='col-xl-4 col-lg-6 col-md-6 col-sm-12'>
              <Card product={product} />
            </div>
          ))}
        </div>

        <h2 className='mb-2 mt-4'>Best Sellers</h2>
        <div className='row'>
          {productsBySell.map((product, i) => (
            <div key={i} className='col-xl-4 col-lg-6 col-md-6 col-sm-12'>
              <Card product={product} />
            </div>
          ))}
        </div>
      </div>
    </div>
  )
);

```

```

    <Copyright />
  </Layout>
);
};

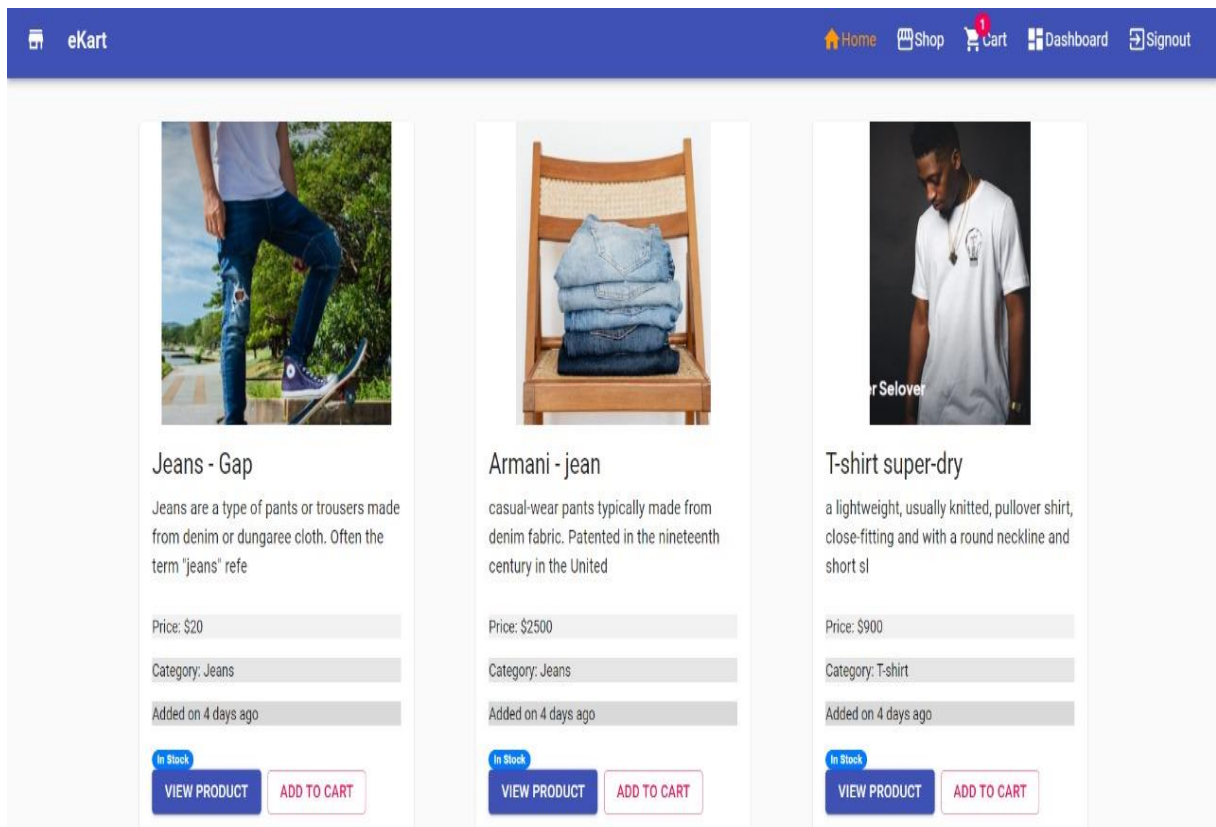
export default Home;

```

The provided React component represents a home page for an e-commerce app. It employs functional components and React hooks such as **useState** and **useEffect** for state management and data fetching. The component fetches and displays new arrivals and best-selling products by calling the **getProducts** function. It also integrates a search functionality through a **Search** component. The layout is structured using Bootstrap classes, ensuring responsiveness. The page includes two sections for new arrivals and best sellers, each populated with product cards rendered using the **Card** component. Additionally, a copyright notice is displayed at the bottom of the page. Overall, the component combines stateful logic, data fetching, and modular components to create a dynamic and visually appealing e-commerce home page.

### 3.2.2.5 Shop Component

A shop page enables user to



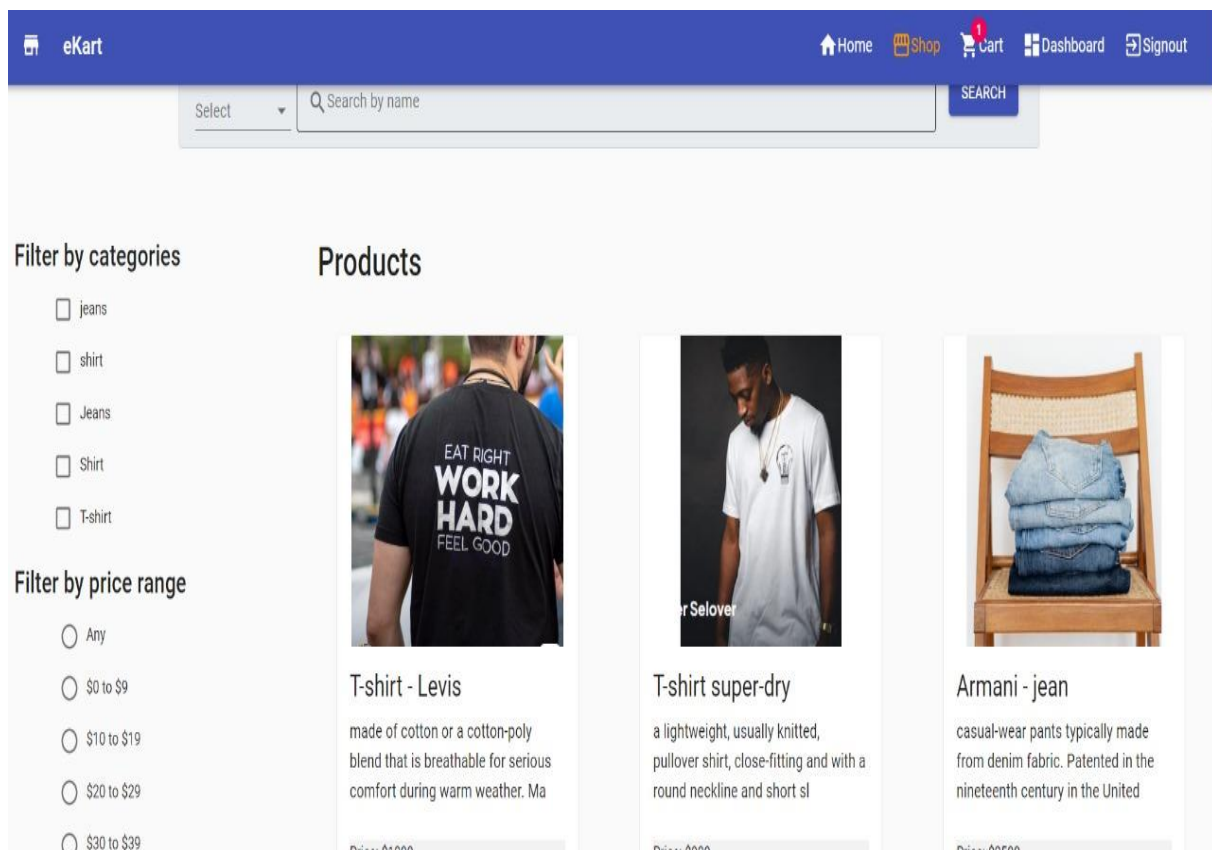


Figure above illustrates the Shop component with shared layout and a navigation bar like Home,Shop ,Cart,Dashboard,signout, followed by some filter features on the left side and filtered results on the right side of the page. Category filter is handled by checkbox component while price filter and categories filter is presented by radio-box component.

There are 5 components in the price filter that is any, \$0-\$9 , \$10-\$19, \$19-\$29 , \$30-\$39

So as per your choice you can fill in the radio boxes to see the further collections.

And in the categories filter there are again five choices like jeans,tshirt,shirt etc.

## UI DESCRIPTION

The home page enables the user to sort the products based on price and categories.

## Logic and Data Flow

The **Shop** component in this React application serves as a dynamic product filtering and display page. It utilizes Material-UI components for a polished interface. Users can filter products by category and price range through checkboxes and radio buttons. The component fetches categories on mount, displays filter options, and dynamically loads and displays filtered products. The "Load more" button allows users to retrieve additional products. The component maintains state for filters, categories, and product results. Overall, it creates a user-friendly shopping experience with interactive filtering options and an aesthetically pleasing design.

```
import React, { useState, useEffect } from 'react';
import Layout from './Layout';
import Typography from '@material-ui/core/Typography';
import Link from '@material-ui/core/Link';
import Box from '@material-ui/core/Box';
import Button from '@material-ui/core/Button';
import Card from './Card';
import { getCategories, getFilteredProducts } from './apiCore';
import Checkbox from './Checkbox';
import RadioBox from './RadioBox';
import { makeStyles } from '@material-ui/core/styles';

import Search from './Search';
import { prices } from './fixedPrices';
import Copyright from './Copyright';

const Shop = () => {
  const [myFilters, setMyFilters] = useState({
    filters: { category: [], price: [] },
  });

  const [categories, setCategories] = useState([]);
  const [error, setError] = useState(false);
  const [limit, setLimit] = useState(6);
  const [skip, setSkip] = useState(0);
  const [size, setSize] = useState(0);
  const [filteredResults, setFilteredResults] = useState([]);

  const init = () => {
    getCategories().then((data) => {
      if (data.error) {
        setError(data.error);
      } else {
```

```

    setCategories(data);
  }
});
};

const loadFilteredResults = (newFilters) => {
  // console.log(newFilters);
  getFilteredProducts(skip, limit, newFilters).then((data) => {
    if (data.error) {
      setError(data.error);
    } else {
      setFilteredResults(data.data);
      setSize(data.size);
      setSkip(0);
    }
  });
};

const loadMore = () => {
  let toSkip = skip + limit;
  // console.log(newFilters);
  getFilteredProducts(toSkip, limit, myFilters.filters).then((data) => {
    if (data.error) {
      setError(data.error);
    } else {
      setFilteredResults([...filteredResults, ...data.data]);
      setSize(data.size);
      setSkip(toSkip);
    }
  });
};

const useStyles = makeStyles((theme) => ({
  btn: {
    background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
    borderRadius: 3,
    border: 0,
    color: 'white',
    height: 48,
    padding: '0 20px',
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
  },
})));

const classes = useStyles();

const loadMoreButton = () => {
  return (
    size > 0 &&
    size >= limit && (
      // <button onClick={loadMore} className='btn btn-warning mb-5'>

```

```

    // Load more
    // </button>
    <Button onClick={loadMore} variant='contained' className={classes.btn}>
      Load more
    </Button>
  )
);
};

useEffect(() => {
  init();
  loadFilteredResults(skip, limit, myFilters.filters);
}, []);

const handleFilters = (filters, filterBy) => {
  // console.log("SHOP", filters, filterBy);
  const newFilters = { ...myFilters };
  newFilters.filters[filterBy] = filters;

  if (filterBy === 'price') {
    let priceValues = handlePrice(filters);
    newFilters.filters[filterBy] = priceValues;
  }
  loadFilteredResults(myFilters.filters);
  setMyFilters(newFilters);
};

const handlePrice = (value) => {
  const data = prices;
  let array = [];

  for (let key in data) {
    if (data[key]._id === parseInt(value)) {
      array = data[key].array;
    }
  }
  return array;
};

return (
  <Layout
    title='Shop page'
    description='Search and find books'
    className='container-fluid'
  >
    <Search />
    <div className='row'>
      <div className='col-md-3'>
        <h4>Filter by categories</h4>
        <ul>
          <Checkbox

```



```

      categories={categories}
      handleFilters={(filters) => handleFilters(filters, 'category')}
    />
  </ul>

  <h4>Filter by price range</h4>
  <div>
    <RadioBox
      prices={prices}
      handleFilters={(filters) => handleFilters(filters, 'price')}
    />
  </div>
</div>

<div className='col-md-9'>
  <h2 className='mb-2'>Products</h2>
  <div className='row'>
    {filteredResults.map((product, i) => (
      <div key={i} className='col-xl-4 col-lg-6 col-md-12 col-sm-12'>
        <Card product={product} />
      </div>
    ))}
  </div>
  <hr />
  {loadMoreButton()}
</div>
</div>
<Copyright />
</Layout>
);
};

export default Shop;

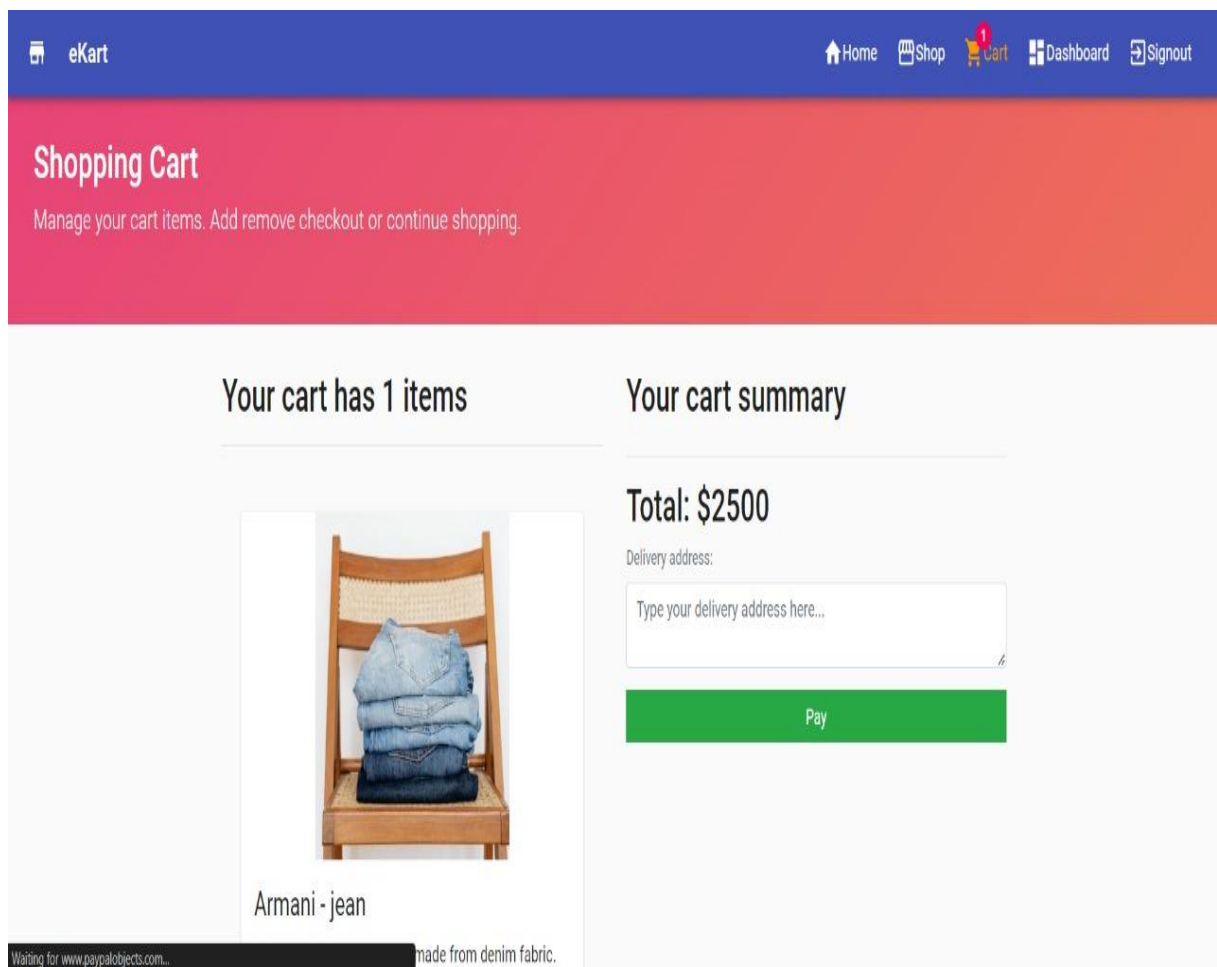
```

### 3.2.1.6 Cart Component and Payment Gateway

#### UI Description

Cart page is the last page users have to go through to get the purchase done. Beside some common shared components, there are a Card component which renders the product that users want to buy and a check-out component that demonstrates the total amount, delivery address and a form to fill in with payment information. This Card component inside Cart is slightly different than the one in Home or Shop component. It has a remove button to delete the product

from the cart and an input field where users can increase or decrease the quantity of product in the end. Cart component is visualized clearly in figure below.



## Logic and Data Flow

When a user clicks Add to cart button on a specific product, that product item will be added to the local storage. Then the user is redirected to this Cart page. Local storage is utilized to store cart data so that the data will not be lost every time the user refresh the page. Cart data stays in local storage until it is removed or after the purchase is handled successfully. To implement and test payment gateway, a test environment which is almost equivalent to production environment – Braintree sandbox is installed. Braintree supports credit card and PayPal, two of the most

popular and emerging payment methods. Firstly, Braintree module is installed in the back-end server so that it can generate a token which is necessary to perform Braintree in the client.

## Braintree

Braintree provides businesses with the ability to accept online and in-app payments. Braintree provides client libraries and integration examples in [Ruby](#), [Python](#), [PHP](#), [Java](#), [.NET](#), and [Node.js](#); mobile libraries for [iOS](#) and [Android](#); and Braintree.js for in-browser card encryption.<sup>[18]</sup>

Braintree also works with most e-commerce and billing platforms, including [BigCommerce](#), [WooCommerce](#), and [Magento](#).

Braintree initiated the credit card data portability standard in 2010, which was accepted as an official action group of the Data Portability project. Credit card data portability is supported by an opt-in community of electronic payment processing providers that agree to provide credit card data and associated customer information to an existing merchant upon request in a [payment card industry](#) (PCI) compliant manner.

```
router.get('/braintree/getToken/:userId', requireSignin, isAuth,
generateToken);
router.post(
  '/braintree/payment/:userId',
  requireSignin,
  isAuth,
  processPayment
);
router.param('userId', userById);
```

As soon as Checkout component is mounted, Effect Hook initiate a function which send an API request to tell the server to create a token for the client. When the token is available with at least 1 product item in the cart, a payment method is shown based on Drop-in, which is imported from the package 'braintree-web-drop-in-react'. After all input are filled with valid data, the user clicks pay button to complete the transaction and create an order in the database, followed by a removal of cart in the local storage and an alert informing successful payment.

## 3.2.2 Back-end development

### 3.2.2.1 Basic setup

The very first thing to do is to set up an Express application by creating an app.js file, which is the entrance file of the project, where some of the necessary middlewares and node packages are stored. Express instance is defined under app variable.

A screenshot of a code editor showing the contents of a file named 'app.js'. The code is written in JavaScript and includes several 'require' statements for various packages: 'express', 'mongoose', 'morgan', 'body-parser', 'cookie-parser', 'cors', and 'express-validator'. It also shows a call to 'require('dotenv').config()' and a comment '// app'. Finally, it defines a constant 'app' as 'express()'. The lines are numbered from 1 to 10.

```
JS app.js
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const morgan = require('morgan');
4  const bodyParser = require('body-parser');
5  const cookieParser = require('cookie-parser');
6  const cors = require('cors');
7  const expressValidator = require('express-validator');
8  require('dotenv').config();
9  // app
10 const app = express();
```

*A screenshot of app.js file from the e-commerce application*

The purpose of those middlewares in figure is given below:

- Mongoose is discussed already
- Morgan logs the request details
- bodyParser unzips the whole body part of an incoming request and makes it available on req.body object .
- cookieParser deconstructs cookie header and populates all the cookie in req object under property named cookies.
- dotenv let developers use environment variable by accessing .env file.

The following step is setting up database connection. MongoDB atlas is used due to various advantages it brings in for the project as chapter 2.3.3 summarizes above. Since Atlas is an online database service, no installation is required. By accessing MongoDB Atlas official site and following instruction there, a cluster is created with the author's personal choice of cloud provider and region followed by a connection string which is saved as MONGO\_URI variable in the environment file. It is then used by mongoose to connect to the database. When the

connection is successful, 'DB Connected' is displayed in the terminal. Otherwise, a message telling the connection error is printed out as Figure below explains.

```
20 // db
21 mongoose.connect(
22   process.env.MONGO_URI,
23   {
24     useNewUrlParser: true,
25     useCreateIndex: true,
26     useUnifiedTopology: true
27   }
28 )
29 .then(() => console.log('DB Connected'));
30
31 mongoose.connection.on('error', err => {
32   console.log(`DB connection error: ${err.message}`)
33 });
```

*A code snippet of how to connect to the database.*

The back-end folder is structured by 3 main sub-folders: models, routes and controllers. While models contain all the mongoose schemas that are based on application requirements, routes define all the API routes and controllers comprise of the logic that needed to be execute after each incoming request match with the corresponding route.

### 3.2.2.2 Mongoose Schema Creation

Mongoose Schema Creation There is a total of 4 schemas in this application: category, order, product, and user. Each mongoose schema represents for a document structure User schema is written first because user model is needed to handle authentication and authorization.

User schema as can be seen from figure below, a user schema has 3 required properties called name, email and password. Role property is set to 0 by default as it means this is a normal user. By manually changing the role value to other the user becomes an admin. The user model only accepts defined properties in the schema to be added to the database. While history shows an array of purchases that user used to make, timestamps display the time of creation and modification of a user object.

```
const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      trim: true,
      required: true,
```

```

    maxlength: 32,
  },
  email: {
    type: String,
    trim: true,
    required: true,
    unique: true,
  },
  hashed_password: {
    type: String,
    required: true,
  },
  about: {
    type: String,
    trim: true,
  },
  salt: String,
  role: {
    type: Number,
    default: 0,
  },
  history: {
    type: Array,
    default: [],
  },
},
{ timestamps: true }
);

```

For security reasons, real password is not saved in the database but an encrypted one, therefore the author used Crypto, which is a Node module to hash the password while combined with a tool called mongoose virtual. A Mongoose Virtual is a property that is not stored in MongoDB. Mongoose Virtuals, which has getter and setter methods are great option for computing properties.

```

// virtual field
userSchema
  .virtual('password')
  .set(function (password) {
    this._password = password;
    this.salt = uuidv1();
    this.hashed_password = this.encryptPassword(password);
  })
  .get(function () {
    return this._password;
  });
userSchema.methods = {

```

```

authenticate: function(plainText) {
  return this.encryptPassword(plainText) === this.hashd_password;
},

encryptPassword: function (password) {
  if (!password) return "";
  try {
    return crypto
      .createHmac('sha1', this.salt)
      .update(password)
      .digest('hex');
  } catch (err) {
    return "";
  }
},
};

module.exports = mongoose.model('User', userSchema);

```

Figure above demonstrates how 'hashed\_password' is created based on Mongoose Virtual. Firstly, Virtual creates a property named password for user object. When a new user is signed up, the value of 'password' property from the rebody is set to the virtual property and is passed to setter function as an argument. Then 'password' is encrypted and set to 'hashed password' property. A function is also assigned to user Schema method object to create an authenticate method which compare the encrypted input password with the hashed one to verify the user.

## Product schema

A product schema is structured as below. It has all the properties that any ecommerce product may need, like name, description, price, quantity, category, etc. Sold field is set to 0 as default value and is incremented after each user purchase. One different with the user schema above is that category field has a type of ObjectId while the 'ref' option indicates which model to refer to during population (in this case category model). This illustrates the relationship between product model and category model and by using populate method, category data is no longer its original \_id but replaced with mongoose document retrieved from the database.

```

const productSchema = new mongoose.Schema(
{
  name: {

```

```

    type: String,
    trim: true,
    required: true,
    maxlength: 32,
  },
  description: {
    type: String,
    required: true,
    maxlength: 2000,
  },
  price: {
    type: Number,
    trim: true,
    required: true,
    maxlength: 32,
  },
  category: {
    type: ObjectId,
    ref: 'Category',
    required: true,
  },
  quantity: {
    type: Number,
  },
  sold: {
    type: Number,
    default: 0,
  },
  photo: {
    data: Buffer,
    contentType: String,
  },
  shipping: {
    required: false,
    type: Boolean,
  },
},
{ timestamps: true }
);

```

### 3.2.2.3 Routing and API Documentations

Data distribution and sharing between two or more systems has always been an essential aspect of software development. Taking into account a case when a user search for some books of a specific category, an API is called and the request is sent to the server. The server analyzes that request, performs necessary actions and finally, user gets a list of books as a response from the server. This is the way how REST API works. An API stands for Application Programming



Interface, which is a set of rules that allows two applications or programs to interact with each other. Then Routes.js file is created with the sole purpose of including all the route paths with their corresponding components by using a package called react-router-dom. Route component is then rendered by ReactDOM as the root component of the application.

```
const Routes = () => {  
  return (  
    <BrowserRouter>  
      <Switch>  
        <Route path="/" component={Home} exact />  
        <Route path="/shop" component={Shop} exact />  
        <Route path="/signin" component={Signin} exact />  
        <Route path="/signup" component={Signup} exact />  
        <PrivateRoute path="/user/dashboard" component={Dashboard} exact />  
        <AdminRoute path="/admin/dashboard" component={AdminDashboard} exact />  
        <AdminRoute path="/create/category" component={AddCategory} exact />  
        <AdminRoute path="/create/product" component={AddProduct} exact />  
        <Route path="/product/:productId" component={Product} exact />  
        <Route path="/cart" component={Cart} exact />  
        <AdminRoute path="/admin/orders" component={Orders} exact />  
        <PrivateRoute path="/profile/:userId" component={Profile} exact />  
        <AdminRoute path="/admin/products" component={ManageProducts} exact />  
        <AdminRoute  
          path="/admin/product/update/:productId"  
          component={UpdateProduct}  
          exact  
        />  
        <Route component={NotFound} />  
      </Switch>  
    </BrowserRouter>  
  );  
};
```

## **Chapter 4: Conclusion**

In conclusion, the development of an e-commerce website represents a multifaceted endeavor blending design, technology, and user experience. Throughout this report, the focus has been on the creation of a responsive platform capable of catering to the dynamic needs of modern users across various devices.

The project's success hinges on meticulous design considerations that prioritize an intuitive interface and a seamless user journey. By employing responsive design principles, the website ensures accessibility and functionality regardless of the device used, fostering a positive user experience.

The integration of front-end and back-end technologies stands as the backbone of this initiative. The collaboration of HTML, CSS, JavaScript, and server-side technologies like PHP or Python empowers the website with interactive features, secure payment systems, efficient database management, and user authentication.

The website's key features, such as comprehensive product catalogs, user account management, secure payment gateways, and intuitive search functionalities, are pivotal in enhancing user engagement and facilitating hassle-free transactions.

Moreover, rigorous testing protocols ensure the website's robustness, security, and performance across diverse environments before deployment. Post-launch, continuous maintenance, updates, and attentive customer support remain imperative in upholding the website's functionality, security, and user satisfaction.

In essence, this e-commerce website endeavor embodies a commitment to providing a user-centric, secure, and accessible platform for seamless online transactions. Its success lies not only in its technical prowess but also in its ability to meet and exceed user expectations while fostering trust and convenience in the digital marketplace.

## **REFERENCES**

<https://legacy.reactjs.org/tutorial/tutorial.html>

<https://blog.hyperiondev.com/index.php/2018/09/10/everything-need-know-mern-stack/>

<https://nodejs.dev/learn/the-v8-javascript-engine>

<https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/>

[https://developer.mozilla.org/enUS/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/enUS/docs/Learn/Server-side/Express_Nodejs/Introduction)

<https://docs.mongodb.com/manual/core/data-model-design/>

<https://reacttraining.com/react-router/web/guides/primary-components>

## **STUDENT DETAILS**



**Shashank Mudgal**

**Er.no-211b287**

**CSE**



**Stuti Jain**

**Er.no-211b319**

**CSE**



**Shivam Swaraj**

**Er.no-211b292**

**CSE**

