AITRONICS E-COMMERCE WEBSITE

A PROJECT REPORT

Submitted by

ANJALI SAH

Under the Supervision of

MR. SHAMEEM AHMAD ANSARI

in partial fulfillment for the award of the degree

of

Bachelor of Computer Application

DEPARTMENT OF COMPUTER APPLICATION

INTEGRAL UNIVERSITY, LUCKNOW

JUNE 2023

ACKNOWLEDGEMENT

I would like to express my sincere gratitude and appreciation to all those who have contributed to the successful completion of the Aitronics E-Commerce Website. Your dedication to teaching and your unwavering commitment to excellence have left an indelible mark on my intellectual and personal development.

First and foremost, I extend my heartfelt thanks to **Professor Md. Faisal** (HOD), for his unwavering guidance and expertise. His insightful feedback, constructive criticism, and continuous motivation played a crucial role in shaping the direction of my major project. I am truly grateful for his patience, dedication, and commitment to my academic growth.

I would immensely grateful to express my gratitude to **Dr. Mohd Faizan** for his valuable insights and recommendations. His vast knowledge and expertise in the field provided me with a supportive environment and valuable discussions which helped me navigate through the complexities of my research. These experiences have not only enriched my knowledge but have also allowed me to develop essential skills such as problem-solving, data analysis, and effective communication.

I would like to express my heartfelt appreciation to **Mr. Shameem Ahmad Ansari**. His willingness to share their perspectives and exchange ideas greatly enriched my project. I am immensely grateful for the countless hours he has devoted for providing constructive feedback on my project. His insightful comments and suggestions have significantly improved my critical thinking skills and have challenged me to push the boundaries of my own abilities.

I would like to express my gratitude to everyone who has contributed to my major project, I extend my sincerest appreciation. Your support and encouragement have been instrumental in the successful completion of this endeavor. I am truly grateful for the opportunities and learning experiences that this project has provided me.

Thank you all for your guidance, patience, and unwavering support. It has been an honor and a privilege to be your student.

With heartfelt appreciation,

Anjali Sah

CERTIFICATE

Certified that this project report **"AITRONICS E-COMMERCE WEBSITE"** is the bonafide work of "**ANJALI SAH"** who carried out the project work under my supervision.

Mr. Shameem Ahmad Ansari
Assistant Professor
Department of Computer Application
Integral University, Lucknow

CERTIFICATE

Certified that this project report **"AITRONICS E-COMMERCE WEBSITE"** is the bonafide work of "**ANJALI SAH"** who have successfully carried out the major project.

Dr. Mohd Faizan
Project Coordinator
Deptt. of Computer Application
Integral University, Lucknow

Prof. Md. Faisal
Head
Deptt. of Computer Application
Integral University, Lucknow

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Date: JUNE 2023

ANJALI SAH

TABLE OF CONTENTS

3. Application Implementation

4. Discussion

LIST OF ABBREVIATIONS

API: Application Programming Interfaces, make software development and innovation simpler by making it easy and safe for programs to share data and functions.

ASP: Active Server Pages is a development framework for building web pages.

DOM: Document Object Model is a programming interface for HTML and XML documents. It shows what the page looks like so that programs can change the document's structure, style, and content. It shows the document with nodes and objects.

ODM: In Object Oriented Data Model, data and their relationships are contained in a single structure which is referred as object in this data model.

JSON: JavaScript Object Notation format is used in data storing and transmitting.

BSON: BSON is a computer data interchange format. The name "BSON" is based on the term JSON and stands for "Binary JSON". It is a binary form for representing simple or complex data structures including associative arrays.

JSX: JavaScript XML is an extended syntax that allows programmers to write HTML in React easily.

NPM: Node package manager is a Node.js tool to construct and administer JavaScript programming libraries.

REST: Representational State Transfers is an architectural style applied to networked applications. It exists as a series of constraints applied to implementations of network elements, allowing for unified interface semantics, rather than application-specific syntax and implementations.

JWT: JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties.

UI: User Interface is everything a user interacts with when using a digital product or service.

1. Introduction

Nowadays, technology is growing incredibly fast. The rapid innovation of hardware devices makes software technologies to advance as well, automatically take place of old technologies. Because of the significant expanding in the number of electronic devices that use Internet and real-time feature, performance is key. By tradition, web development has been carried out by technologies such as JAVA servlets, ASP.NET or PHP. While those technologies are quite widespread and have good features with many years of development and are supported by a large community, they still have some limitations concerning about today's need which is performance. The MERN stack (MongoDB, Express, React and Node) with their simplicity and uniformity, has been recently developed to become a better solution for this performance issue.

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 thesis 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, the author realized how e-commerce – 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. Understanding this need, the author's solution was to create an e-commerce web application as an online bookstore in order for the startup to develop its business strategy.

This document structure was organized as follow. The first section brought in the goal of the thesis and technologies used. Next, essential concepts and theoretical background of each technology in the stack was introduced along with example, followed by the third section which demonstrated carefully and thoroughly the application development process, from back-end to front-end. In the end, this paper provided discussion of the project with further improvements and gave conclusion about the final product.

1.1 E-commerce Overview

Ecommerce, often known as electronic commerce or online commerce, refers to the buying and selling of things and services through the internet. The term "e-commerce" was initially used in the 1960s. With the rising popularity of mobile devices, social media has become a powerful affirmation of the strength and growth of the web page after years of development. Commercial launchers help to expedite the growth of trade (E-commerce).

1.2 MERN - Stack Objective and Scope

The objective of the MERN stack is to provide developers with a unified and efficient way to build modern web applications. It aims to leverage JavaScript, a widely used programming language, across the entire application stack, from the server-side to the client-side. The MERN stack covers both the frontend and backend aspects of web development, allowing developers to create scalable and performant applications. Here's a brief overview of each component's role in the stack:

1. Full-Stack Development: The MERN stack enables developers to build both the frontend and backend components of a web application using JavaScript. It provides a unified development environment for creating end-to-end solutions.

2. Single Language: JavaScript is the primary language used throughout the MERN stack, allowing developers to write code for the entire application using a single language. This promotes code reuse, consistency, and simplifies the development process.

3. Frontend Development: React, one of the key components of the MERN stack, is a powerful library for building user interfaces. It provides a component-based architecture, virtual DOM, and a rich ecosystem of libraries and tools for frontend development.

4. Backend Development: Node.js and Express.js form the backend part of the MERN stack. They provide a server-side environment for handling requests, creating APIs, and performing server-side operations. Node.js offers event-driven, non-blocking I/O, making it efficient for handling concurrent requests.

5. Database: MongoDB is a NoSQL database used in the MERN stack. It provides a flexible schema and allows developers to store data in a JSON-like format, making it suitable for handling complex and changing data structures

6. Scalability: The MERN stack is designed to handle scalability challenges. It offers features like server-side rendering, caching, and load balancing to ensure the application can handle increasing traffic and data volumes.

7. Community and Ecosystem: The MERN stack has a vibrant and active community that contributes to the development of libraries, frameworks, and tools. This provides developers with a wide range of resources and support when building applications using the MERN stack.

8. Deployment: The MERN stack supports various deployment options, including cloud platforms, containerization, and serverless architectures. This flexibility allows developers to deploy their applications in a manner that best suits their needs.

In summary, the scope of the MERN stack includes frontend and backend development, database management, scalability, deployment options, and a supportive community. It offers a comprehensive set of tools and technologies for building robust, scalable, and modern web applications.

1.3 Advantages of MERN Stack Development

The MERN stack offers numerous advantages for building e-commerce applications, businesses can create robust and feature-rich e-commerce platforms that deliver exceptional user experiences and drive business growth. Some common advantages are:

1. Single Language: The MERN stack allows developers to use JavaScript throughout the entire application, both on the frontend and backend. This enables seamless communication and code sharing between different components of the e-commerce application, making development more efficient and streamlined.

2. Efficient Development: The MERN stack provides a comprehensive set of tools and libraries that are specifically designed for building web applications. React simplifies the creation of interactive and responsive user interfaces, while Express.js and Node.js offer a robust server-side framework for handling requests and building APIs. This combination results in faster development cycles and improved productivity.

3. Scalability and Performance: The MERN stack is well-suited for handling high volumes of traffic and data. MongoDB, a NoSQL database, offers scalability and flexibility, allowing e-commerce businesses to store and manage large amounts of product data, customer information, and transaction records. Node.js, with its non-blocking I/O model, supports handling multiple concurrent requests efficiently, ensuring high-performance delivery of content and seamless user experiences.

4. Real-time Updates: Using the MERN stack, developers can easily implement real-time features in e-commerce applications. For example, using React and Node.js together with technologies like WebSocket or the Socket.IO library, you can enable real-time notifications for order updates, inventory changes, or chat functionality, enhancing the overall user experience.

5. Extensive Ecosystem: The MERN stack has a thriving and supportive community. It offers a vast ecosystem of libraries, frameworks, and open-source projects that can accelerate development and provide solutions for common e-commerce challenges. The availability of community-contributed packages ensures developers have access to a wide range of tools and resources for building feature-rich e-commerce applications.

6. Reusability and Code Maintainability: React component-based architecture promotes code reusability, allowing developers to create modular UI components that can be reused across different parts of the e-commerce application. This not only saves development time but also enhances code maintainability and simplifies future updates or modifications.

7. SEO-Friendly: React, when used with server-side rendering (SSR) techniques, enables search engine optimization (SEO) for e-commerce applications. SSR ensures that search engines can crawl and index the content effectively, improving the application's visibility and search engine rankings.

8. Cross-Platform Compatibility: The MERN stack supports building applications that are compatible with various platforms and devices. With React ability to create responsive and mobile-friendly interfaces that can provide consistent user experiences across different devices, including desktops, tablets, and smartphones.

1.4 Limitations

E-commerce, or electronic commerce, refers to the buying and selling of goods and services over the internet. While e-commerce offers numerous benefits and opportunities, there are also some limitations and challenges that businesses in this sector may face. Here are some common limitations for e-commerce:

1. Technical Challenges: E-commerce relies heavily on technology infrastructure, including websites, servers, payment gateways, and security measures. Technical issues such as website downtime, slow loading times, and security vulnerabilities can negatively impact the user experience and erode customer trust.

2. Security Concerns: E-commerce involves the transmission of sensitive customer information, such as credit card details and personal data. Ensuring robust security measures to protect against data breaches, hacking attempts, and fraudulent activities is crucial. Failure to address security concerns can lead to financial losses and damage to the business's reputation.

3. Lack of Personal Interaction: Unlike physical stores, e-commerce lacks the face-to-face interaction between customers and sales staff. This absence of personal touch can make it challenging to provide personalized recommendations, address customer queries in real-time, and build customer loyalty.

4. Customer Trust: Establishing trust with customers is vital in e-commerce. Potential customers may have concerns regarding the authenticity and quality of products, as well as the reliability of the seller. Building trust through customer reviews, secure payment options, clear return policies, and communication is essential for e-commerce success.

5. Logistics and Shipping: Efficient logistics and timely delivery are crucial in e-commerce. Businesses need to ensure reliable shipping partners, manage inventory effectively, and handle order fulfilment accurately. Challenges such as shipping delays, inventory management issues, and product damage during transit can affect customer satisfaction and loyalty.

6. Returns and Customer Service: E-commerce customers often have the expectation of hassle-free returns and responsive customer service. Handling returns, exchanges, and customer inquiries promptly and effectively can be demanding, requiring efficient systems and processes to ensure customer satisfaction.

7. Digital Marketing and Competition: E-commerce operates in a highly competitive digital landscape. Businesses need to invest in effective digital marketing strategies to reach and engage their target audience. However, standing out among competitors and driving traffic to the e-commerce platform can be challenging and may require significant marketing efforts and investments.

8. Infrastructure and Scaling: E-commerce businesses need to scale their operations to handle increasing website traffic, order volumes, and customer demands. Ensuring a robust and scalable infrastructure, including servers, databases, and payment gateways, is essential to avoid performance issues during peak periods.

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.

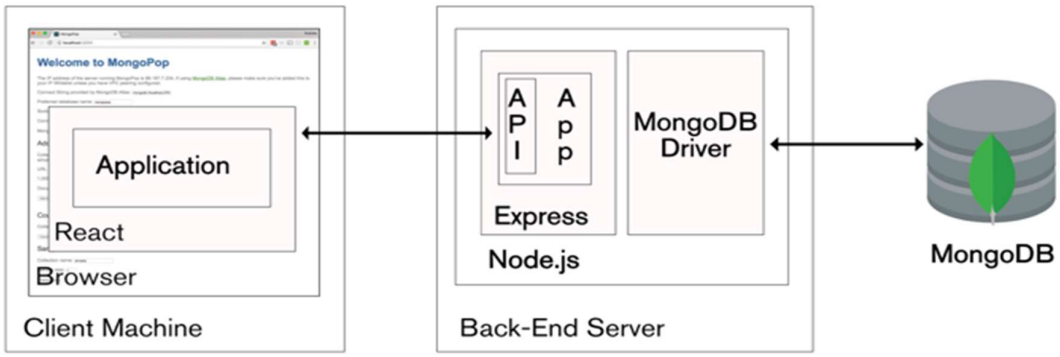


Figure 1. MERN stack architecture

Figure 1 explains the architecture of MERN stack. Firstly, Express with Node.js create API which is used for logic and to interact with MongoDB. When React client sends a HTTP request to the back-end server. The server analyses the request, retrieves data from the database and responses with that data. React client is updated based on that returned data.

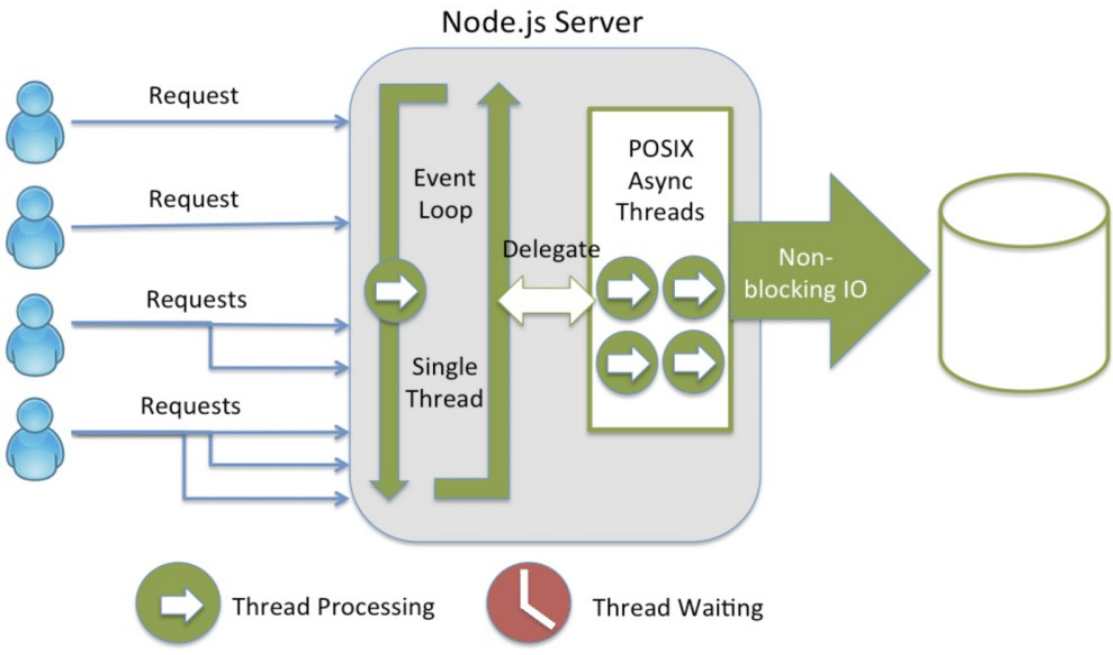
The MERN stack's application infrastructure encompasses client-side and server-side components, along with a NoSQL database. It enables seamless communication and integration, and provides various deployment options for hosting web applications. The stack supports communication and integration between these components, and offers various deployment options including cloud platforms, containerization, and serverless architectures. This infrastructure forms the backbone of MERN stack applications, providing a reliable and efficient environment for building web applications.

2.1 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.

A Node app runs in an individual process with event looping, without the need of a new thread for each operation. This is opposed to traditional servers which make use of limited thread to handle requests. Node is a non-blocking, asynchronous and event-driven engine aimed for scalable application development. Generally, Node libraries are created using non-blocking pattern. Application calls a request and then move on to work on next task rather than stalling while waiting for a response. When the request is completed, a call back function informs the application about the results. This allows multiple connections or requests to a server to be executed simultaneously which is important when scaling applications. MongoDB is also invented to be used asynchronously; therefore, it is compatible with Node.js applications.

Figure 2. A popular web server task (reading a file on server and send the content as response to the client) handled Node.js



Node terminate the waiting time between handling requests and move on to the next request as Figure 2. illustrates.

2.2.1 Node Modules

Node module is similar to JavaScript libraries which consists of a package of functions to be included in application when needed. Node has a variety of modules that offer fundamental features for web applications implementation. While developers can customize their own modules for personal project, Node has many built in modules that can be used instantly without installation. One of the most popular built-in modules is http module, which can be used to create an HTTP client for a server.

```
var http = require('http');

//create a server object:

http.createServer(function (req, res) {
        res.write('This is shown in client');          //write a response to the client
        res.end();                                     //end the response
 }).listen(8080);                                      //the server object listens on port 8080
```

Listing 1. A simple code in Node.js to create a server.

Listing 1 demonstrates the usage of http module and 'createServer' function to initiate a server running on port 8080. A text string is shown in the client as a response.

2.2.2 Node Package Manager

In 2017, statistic showed that more than 350 000 packages are found in the npm registry, make it the largest software repository on Earth. Due to being an open source and having a basic structure, the Node ecosystem has prospered dramatically, and currently there are over 1 million open-source free package, which facilitates development process a lot. Beginning with being a method to download and manage Node dependencies, npm has become a powerful means in front-end JavaScript.

There are two types of Node packages installation, globally or locally by the npm install command. The package then is downloaded from the NPM registry and presented in a folder named node_modules. The package is also added to the package.json file under property dependencies. By using require function follow with the package name, it is usable in the project as the example below in listing 2:

```
$ npm install <Module Name>

var name = require('Module Name');
```

Listing 2. Installation and usage of Node package.

Package.json is an obligatory file in all JavaScript/Node project. It is a JSON format file which displays pair value of names and versions of all the packages that required to run the project as well as other commands. Sometimes when a package got outdated, it will show up by running the command 'npm outdated'. If those updates are main releases, common 'npm outdated' do not work because a core release usually comes up with crucial changes which can cause the project trouble. In this case, Node package 'npm check-updates' need to be installed globally and then running command 'ncu-u' will upgrade all the version noted in package.json. Now 'npm update' is ready to handle all the updates.

Figure 3. A screenshot of package.json file in E-commerce application.

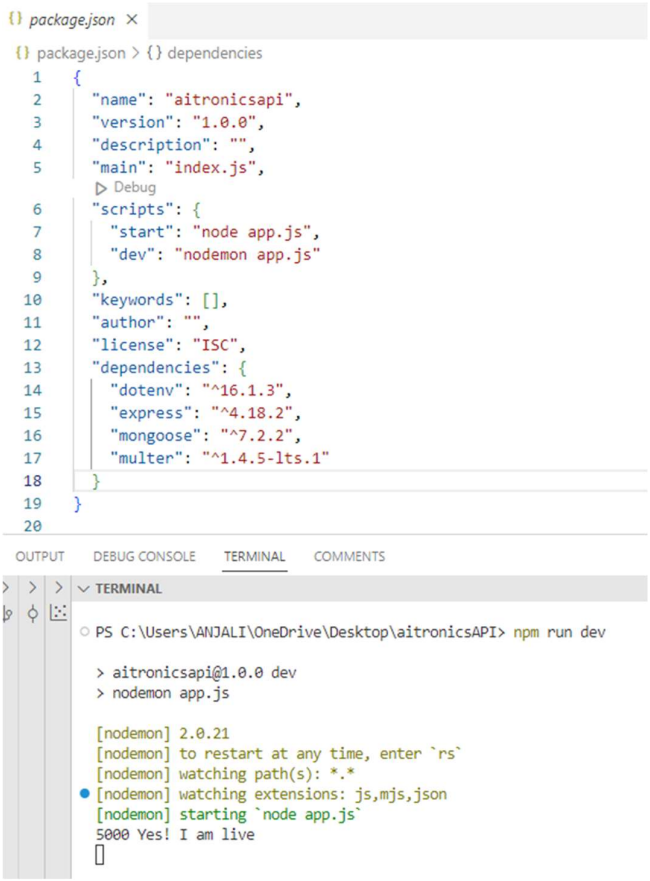


Figure 3 illustrates some Node package used in the application under dependencies property such as express, multer, mongoose, dotenv, etc.

2.2.3 Node Event Loop

Event Loop is one of the most essential concepts to master Node.js. Node.js takes advantage of events heavily and it is literally one of the reasons why Node has pretty highspeed compared to other technologies. As this paper mentions above, Node.js code runs on a single thread, there is only one thing occurring at a time. This seems like a drawback but turns out to be helpful because developers do not have to worry about concurrency problems but only have to care mainly if anything could block the thread such as infinite loops or synchronous requests. The execution flow of JavaScript code will be blocked if there is a piece of code that takes too long to return back to the event loop. The event loop constantly looks for all the function calls in the call stack and execute one by one respectively.

Listing 3. Example about function call in event loop.

```
const hello = () => console.log('hello')
const hey = () => console.log('hey')
const run = () => {
    console.log('run') ;
    setTimeout(hello, 0) ;
    hey()
}
run() ;
```

Listing 3 presents the surprising output. When this code executes, first run() is called. Inside run(), setTimeout() is called first with hello as an argument. It is instructed to run instantly as 0 is passed as timer and finally hey() is called.

This is happening because the callback function of setTimeout() which is hey() is put in the Message Queue after the timer expire (immediately). Message queue is simply where functions are queued before code can reach them. The loop process prioritizes everything in the call stack and when there is nothing left, functions in the message queue are selected to run.

2.2 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 Holowaychuk 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.

2.2.1 Basic Routing

Routing determines the way how an application's endpoints (URIs) interact with client requests. The basic syntax consists of an object as instance of Express and the correspondent HTTP request method such as app.get(), app.post() to handle GET and POST request respectively.

Routing methods take a callback function as parameter. The specified function is called when the HTTP requests match the defined routes and methods.

Listing 4. Example of Routing functions

```
var express = require('express');
var app = express();   // create instance of express
//respond with 'hello' when a GET request is made to the route :/hello
app.get('/hello', function(req, res) {
        res.send('hello');    //response method
})
var callback1 = function(req, res, next) {
        next();
}
```

```
var callback2 = function(req, res, next) {
        next();
}
app.get('/multiple', [callback1, callback2]);
```

As can be seen from listing 4, routing method in line 17 has more than 1 callback function as parameters. In this case, it is crucial to add 'next' to each callback function arguments and call it later in the body to switch control to the next callback. 'next' function will be discussed more in the next section.

2.2.1 Writing and using Middleware

Middleware functions have the ability to approach request object, response object and the 'next' function in the application's request-response phase. Middleware execution return an output which could be either the final outcome or could be passed as an argument to the next middleware until the end of the cycle. The 'next' function belongs to the Express router which starts the following middleware right after the current middleware finished. If a middleware does not finish the req-res cycle, next() must be called to pass control to the next one. Otherwise, the request will be suspended.

Figure 4. Middleware function components.

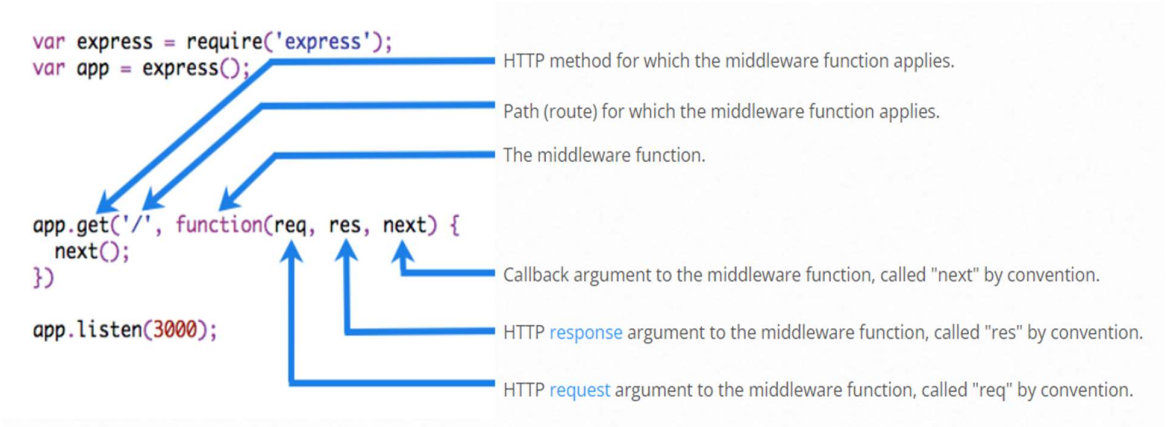


Figure 4 illustrates middleware concepts in details.

An Express application is basically a sequence of middleware calls. The call orders are based on the order which they are declared as:

```
var express = require('express');
var app = express();

//First middleware before response is sent
app.use(function(req, res, next){
  console.log("Start");
  next();
});

//Route handler
app.get('/', function(req, res, next){
  res.send("Middle");
  next();
});

app.use('/', function(req, res){
  console.log('End');
});

app.listen(3000);
```

Figure 5. Diagram of middleware execution order.

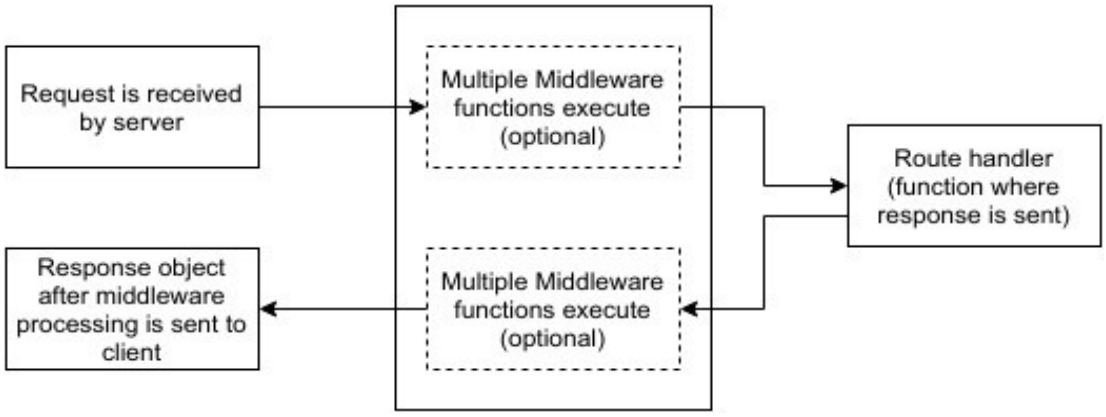


Figure 5. explains that Express treats all middleware in the same way so that the order in which your route handlers and other middleware functions are written is exactly the execution order.

2.3 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. Therefore, this removes the necessity of storing strictly structured data which is obligatory in other relational databases and improves database operation speed significantly. Indexing and aggregation offer robust method to access and work with data easily. Whatever field in a document can be indexed, leading to a much better search performance. MongoDB also provides numerous operations on the documents such as inserting, querying, updating as well as deleting. With the diversity of field value and strong query languages, MongoDB is great for many use cases and can horizontally scale-out to provide storage for larger data volumes, make it stably be the most popular NoSQL database globally.

2.3.1 NoSQL Database

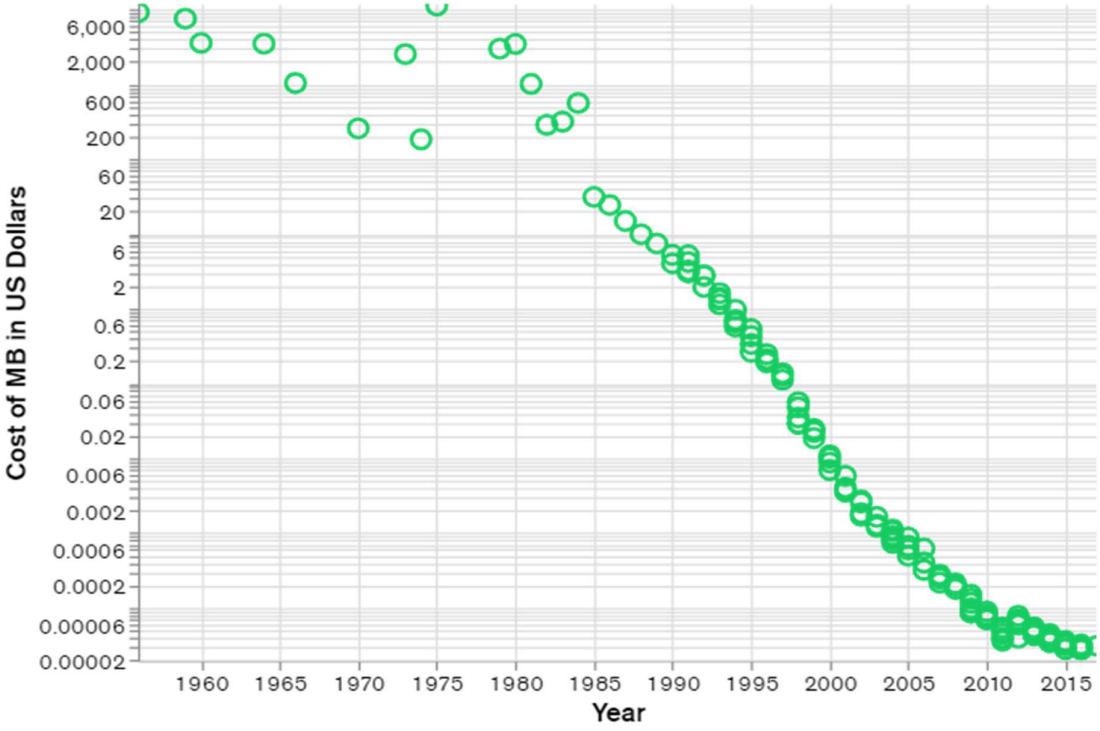
There are big differences between the mechanism NoSQL databases store data and relational databases do. NoSQL databases are differentiated based on their data model. There are 4 main types:

- Document databases
- Key-value databases
- Wide-column databases
- Graph databases

They share one common thing which is dynamic schema and can scale easily with large data loads and high user volumes. A popular misunderstanding is that NoSQL databases do not handle relationship data well. While in fact, many developers believed that data modelling relationship is easier for NoSQL databases due to the nested individual data structure.

During the late 2000s, as a result of the considerable decrement in storage price, there is no need to create a complicated data model just to avoid data duplication. NoSQL databases took a big step forward and became a good option for developers and large data volume applications.

Figure 6. Cost per MB of data overtime.



As price of storage critically decreased as figure 6 demonstrates, the number of applications along with the data needed to store increased incredibly. Defining the schema beforehand for all those data is almost impossible. Therefore, NoSQL databases provide developers flexibility to save a large amount of unstructured data.

2.3.2 Data Models

The key challenge in data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns. When designing data models, always consider the application usage of the data (i.e., queries, updates, and processing of the data) as well as the inherent structure of the data itself. Data model design is determined by the document's structure and relationships between data. Related data can be embedded into a single document.

1. Embedded Data Model

Embedded documents present the connection between data by attaching related data in a single structure. Related data then can be fetched and manipulated in a single operation.

Figure 7. A picture of an embedded document.

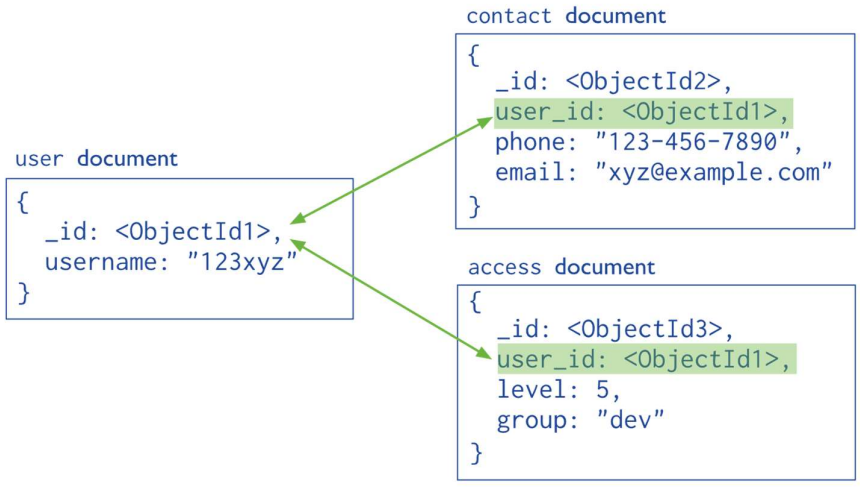
Figure 7. illustrates how related information is stored in the same database record. As a consequence, fewer queries are needed to handle operations leading to a better performance, especially with read operation.

In conclusion, embedded data models should be used when there are relationships between entities or when one to many relationships exists, as long as the document size is smaller than 16mb which is BSON largest size allowed.

2. Normalized Data Model

As its own name shows, this kind of data model is normal and common. Applications can resolve these references to access the related data. References are added from one document and linked to other to present the relationship between data. Related data then can be accessed by resolving references.

Figure 8. References between related data.



As can be seen from figure 8, both contact document and access document have reference linked to the user document.
Normalized data models should be used in these following cases:

- data duplication happens when embedding but performance advantages are not enough to outperform the duplication.
- complicated many-to-many relationships exist.
- big hierarchical data exist.

2.3.3 MongoDB Atlas

MongoDB Atlas, which announced in 2016 by MongoDB creator team, is the cloud service for storing data globally. It offers everything MongoDB provide without further requirements when building applications, allowing developers to fully focus on what they do best. With Atlas, developers do not have to worry about paying for unnecessary things as they follow a pay-as-you-go standard.

MongoDB Atlas introduces a simple interface to get started and running. Basically, by choosing instance size, region and any customized features needed, Atlas brings out the suitable instance and includes it with the following:

- Built-in Replication: Atlas ensures constant data availability by providing multiple servers, even when the primary server is down.
- Backups and Point-in-time Recovery: Atlas puts a lot of efforts to prevent data corruption.
- Automated Update and One-Click Upgrade: users take advantage of the latest and best features as soon as they are released due to Atlas automated patching.
- Various options for additional tools: users can freely decide on which regions, cloud providers and billing options they prefer to use, making them feel like customizing their own instances.
- Fine-Grained Monitoring: users are kept up with a diversity of organized information, to let them know the right time to advance things to the next level.

MongoDB Atlas is practical and reliable thanks to its integrated automation mechanisms. With Atlas, it is unnecessary to concern about operational tasks as following:

- Supply and Configuration: Atlas gives clear instruction step by step to go through the setup process, so developers do not have to think about what aspect to choose even if they are beginners.
- Patching and Upgrades: Atlas is integrated with automatic upgrade and patching, ensures user can reach latest features when they are released as patching process takes just some minutes with no downtime.
- Monitoring and Alerts: database and hardware metrics are always visible so users can foresee and prepare for any performance and user experience problems.

2.3.4. Mongoose

Mongoose is an Object Data Modelling (ODM) JavaScript-based library used to connect MongoDB with Node.js. Mongoose provides a straight-forward, schema-based solution to create your application data template. It handles data relationships, provides methods to validate schema, and is used to render and represent between objects in MongoDB.

Mongoose schema is covered inside a mongoose model, and a specific schema creates model. While schema and model are slightly equivalent, there is a primary difference: schema determines the document formation, default values, validator, etc. while model is responsible for document-related operation like creating, querying, updating and deleting. A schema definition is simple, and its structure is usually based on application requirements. Schemas are reusable and can include multiple child-schemas. Schema applies a standard structure to all the documents in a collection. Moreover, Mongoose provides additional features like query helper functions and business logic in the data. For example, Mongoose can help connect database with the server and perform typical database operations for reading, writing, updating and deleting data. Mongoose removes the need of writing complicated data validations by providing schema validation rules to allow only acceptable data to be saved in MongoDB.

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection as:

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const blogSchema = new Schema({
  title: String, // String is shorthand for {type: String}
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

2.4 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 libraries.

Figure 9. Number of package downloads with NPM per year of React, Solid, Svelte, Angular and Vue.

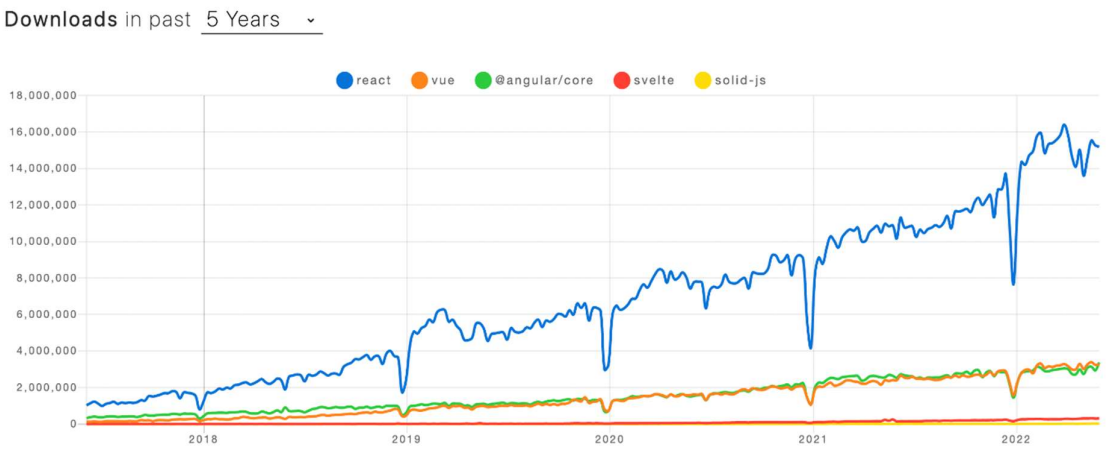


Figure 9. compares the number of package downloads with NPM among React and other popular frameworks – Solid, Svelte, Angular and Vue. As can be seen, the number of React has increased dramatically in popularity in the past few years and definitely has proven its preference. React is the favourite framework of 40.14% of developers, Angular with 22.96%, and Vue with 18.97% of developers.

React works only on user interfaces of web and mobile applications. This equivalent to the view layer of MVC template. ReactJS allows developers to create reusable UI components as well as to create scalable web applications that can change data without reloading the page. In addition to providing reusable component code, which means reducing development time and lessening the risk of bugs and errors, react released some key features that add to its appeal for developers.

- JSX (JavaScript Syntax Extension)
- Virtual DOM
- Components

2.4.1 JSX

The beginning foundation of any basic website is made by HTML files. Web browsers read and display them as web pages. During this process, browsers create a Document Object Model which is a representational tree of the page structure. Dynamic content can be added by modifying the DOM with JavaScript. JSX is XML/HTML-like syntax used to make it easier to modify the DOM by using simple HTML-like code. Literally, JSX makes it possible to write brief HTML-like structures in the same file with JavaScript code, then transpolar such as Babel will convert these expressions to standard JavaScript code. Contrary to the past when JavaScript is put into HTML, now JSX allows HTML to be mixed in JavaScript.

React takes advantage of JSX to handle templating and writing React element. It is not compulsory to use JSX, but following are some benefits that should be considered:

- Time efficiency due to performing optimization while compiling code to JavaScript.
- Type-safe due to the ability to find out most of the errors during compilation.
- Faster in templating, if developers are familiar with HTML.

Figure 10. Some examples of JSX

HeroSection.js

src > components > HeroSection.js > HeroSection

```
1  import { NavLink } from "react-router-dom";
2  import styled from "styled-components";
3  import { Button } from "../styles/Button";
4
5  const HeroSection = ({ myData, myValue }) => {
6    const { name } = myData;
7    const { value } = myValue;
8
9    return (
10     <Wrapper>
11       <div className="container">
12         <div className="grid grid-two-column">
13           <div className="hero-section-data">
14             <p className="intro-data">Welcome to </p>
15             <h1> {name} </h1>
16             <p> {value} </p>
17             <NavLink>
18               <Button>shop now</Button>
19             </NavLink>
20           </div>
```

As can be seen in figure 10 line 6, variable ‘name’ is called inside JSX by wrapping it in curly braces. In general, all valid JavaScript expression need to be inside curly braces in order to be used in JSX. Line 15 and 16 indicate that in case of an attribute, either quotes or curly braces should be used properly depend on specific situations

2.4.2 Virtual DOM

Considered as the next important breakthrough in web development after AJAX, the virtual DOM (Document Object Model) is the main reason why React is able to create fast and scalable web application.

Normally without react, the web application uses HTML to update its DOM. This works well for uncomplicated and static sites, but it can be a huge performance issue for dynamic websites especially the one that involve high user interaction and view update because the whole DOM needs to reload every time the user triggers a page refresh. Despite of JavaScript engines nowadays which are fast enough to carry out such complicated applications, DOM manipulations are still not that advanced.

Another factor is that DOM is tree-structured so that even a change in a layer can cause extreme changes. Updating DOM is always the shortcoming when it comes to performance. React optimized this by introducing Virtual DOM.

Virtual DOM is a representation of the DOM object, where it executes all updates required before rendering the final page to the browser. Whenever a user action happens or by any means, there is a change to the DOM, react will generate a copy of the Virtual DOM. At that point, a diff algorithm compares the previous and current states of the Virtual DOM and figures out the optimal way with the least amounts of updates needed to apply these changes. If there are any differences, react updates that single element only rather than update a ton of elements, while if there is no change, react make browser render nothing. As a result, this behaviour spends less computing power and less loading time, leading to better user experience and performance.

Figure 11. A picture of React Virtual DOM

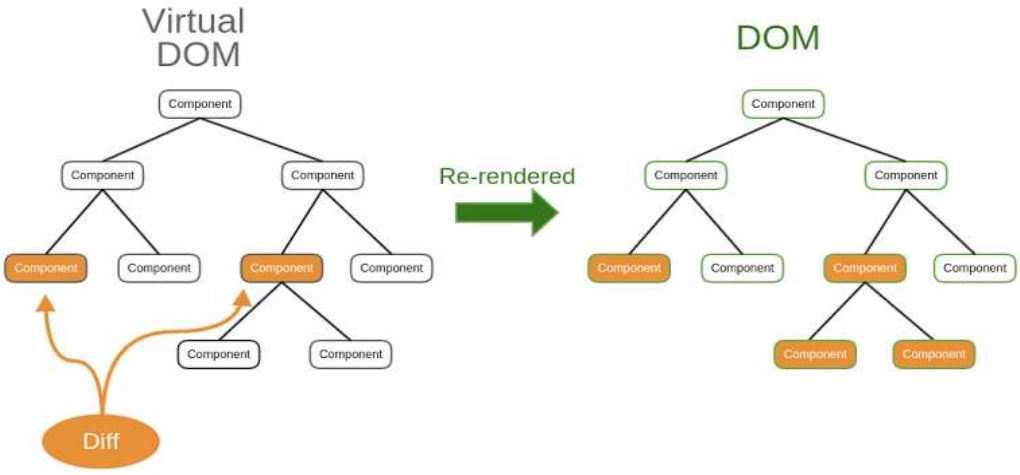


Figure 11. indicates how Virtual DOM works, only the specific changes are updated and re-rendered to the real DOM. This is the core reason behind React high performance.

2.4.3 Components

React is all about component, which reflects the primary unit of every application. React allow developers to divide the user interface into simple, independent and reusable components. A developer can start with common tiny components that can be used repeatedly on several pages like navigation bar, button, input form etc. This is what reusability means, which is a perfect way to reduce development time, so that developers can concentrate on more crucial features and business logic. Then comes the turn of wrapper components that include children components with internal logic. Each component determines its own way how it should be rendered. A component can also be combined with other components or nested inside higher order components. The development process keeps going on like that until it reaches the root component which is also the application. As a result, this approach guarantees the application consistency and uniformity which make it easier for maintenance as well as further codebase growth.

There are 2 types of components, functional component and class component:

1. Functional Components

Functional components are completely presentational and basically are written under the form of a JavaScript function that can receive some props (properties) as argument and return a React element. Thanks to their predictability and straightforwardness, functional components always make the same result given the same props. Therefore, developers tend to pick functional components whenever possible. Stateless component or dumb component is all functional component's name as it comes from the natural fact that functional components do not manage any state and their task is simply output user interface elements.

Listing 5. A functional component

```
Function Hello (props) {
Return <h1> Hello {props.name} </h1>;
}
```

Listing 5 illustrates an example of a functional component which takes 'name' as a props and return a JSX.

This function is a valid React component because it accepts a single "props" (which stands for properties) object argument with data and returns a React element. We call such components "function components" because they are literally JavaScript functions.

2. Class Components

ECMAScript6 class syntax are utilized to write class-based components, the one which not only can return JSX but also can dispatch action, fetch data, handle events and have local state. Class components offer more functionalities by inheriting from React.Component class. They can access all additional lifecycle methods from their parent class and manipulate code at particularly moment during the process.
While functional components are dumb and stateless, class components are said to be smart and stateful thanks to being a container of the local state, internal logic and other children components. Therefore, a class component should be considered in case if there is a need for local state management or lifecycle method as well as some logic. Otherwise, a functional component is more suitable.

Listing 6. A class component

```
class Welcome extends React.Component {
   render() {
     return <h1>Hello, {this.props.name}</h1>;
   }
}
```

As Listing 6. summarizes, all React Component subclasses have to define render function. A constructor function is implemented here because there is initial state and inside constructor, super(props) should be called first out of all statements. If not, this can lead to errors because of undefined props.

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 is able to 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.

By interviewing and researching some normal customers and business owners, the author figured out what different type of user want from the application, what features they think are necessary for an e-commerce web application. Based on that, a list of user stories is shown below to illustrate some of the required functionalities for this application:

- As a user, user want to create an account for himself
- As a user, user want to update his profile
- As a user, user want to surf through all the products
- As a user, user want to see product information such as category, price, name, review, picture, etc.
- As a user, user want to add many products to the shopping cart and is able to view the cart
- As a user, user want to delete products from the cart
- As a user, user want to modify the quantity of products inside the cart
- As a user, user want to pay for the products in the cart
- As an admin, admin want to add product to the database
- As an admin, admin want to remove product from the database
- As an admin, admin want to update product to the database
- As an admin, admin want to add category to the database
- As an admin, admin want to manage user orders

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. Due to the limited scope of this thesis, it is not able to mention all the files or describe every step in the project into details, but it aims to discuss precisely about all fundamental and important parts that needed to implement the MERN application. Basic concepts of any third-party libraries or module are also explained along the way. 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 Back-end Development

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 middleware and node packages are stored. Express instance is defined under app variable.

Figure 12.1 A screenshot of app.js file from the e-commerce application

Figure 12.2 A screenshot of Database connect.js file from the e-commerce application

The purpose of those middleware in figure 12.1 & 12.2 are explained as below:

- Mongoose is discussed in chapter 2.3.4
- 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 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.

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 13 below explains.

```
JS data.js
JS data.js > ...
1   require("dotenv").config();
2   const connectDB = require("./db/connect");
3   const Product = require("./model/product");
4
5   const ProductJson = require("./products.json");
6
7   const start = async () =>{
8       try{
9           await connectDB(process.env.MONGODB_URL);
10          //to refuse the re-insertion of file data
11          await Product.deleteMany();
12          await Product.create(ProductJson);
13          console.log("Products inserted successfully");
14      }
15      catch(error){
16          console.log(error);
17      }
18  };
19
20  start();
21
22  //run node data.js to insert mongodb data at atlas cloud
```

Figure 13. A code snippet of how to connect to the database

2. MongoDB 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.

Product schema

A product schema is structured as figure 16 below. It has all the properties that any e-commerce 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 Object Id 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.

product.js

model > product.js > productSchema > rating

```
const  mongoose  = require("mongoose");

const productSchema = new mongoose.Schema({

    id:{
        type:String,
        required:true,
    },
    name:{
        type:String,
        required:true,
    },
    image:{
        data:Buffer,
        contentType:String,
    },
    price:{
        type:Number,
        required:[true, "price must be provided"],
    },
    rating:{
        type:Number,
        default:4.5,
    },
    createdAt:{
        type:Date,
        default: Date.now(),
    },
    company:{
        type:String,
        enum:{
            values: ["apple", "samsung", "dell", "hp", "razer", "msi","rolex","nokia","asus","lenovo", "mi"],
            message: `{VALUE} is not a valid brand.`,
        }
    },
    description:{
        type:String,
        required:true,
    },
    catogory:{
        type:String,
        required:true,
    },
    featured:{
        type:Boolean,
        default:false,
    }
});

module.exports = mongoose.model("Product", productSchema);

```

Figure 14. Product Schema structure

While user schema and product schema look complex, category schema is pretty simple and straightforward with only name and timestamps field.

3. Authentication with JWT

JSON Web Token (JWT) is a standard that defines a compact and self-contained way to transmit data securely between client and server as a JSON object. Tokens can be transferred through an URL or a HTTP request header easily thanks to its compact size while self-contained feature let JWT comprise all the necessary user information. Data inside a JWT is verified and reliable because it is signed with a secret key or a public private key pair. Moreover, the signature also confirms that only the party holding the private key have signed it.

JWTs consist of three parts separated by dots (.), which are:

- **Header**
- **Payload**
- **Signature**

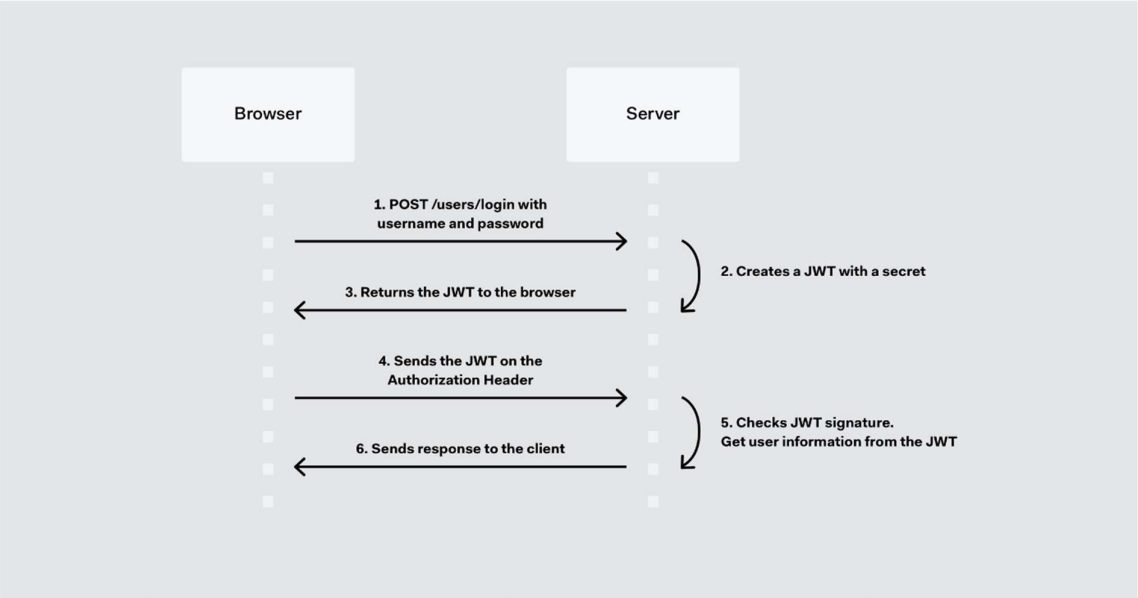


Figure 15. How JWT works in authorization

Authentication is the most popular use case of JWT. After a user sign in, the authorization is granted and a JWT is created by the application and sent back to the user. By that time, all following requests made by user will contain the token in the authorization header with Bearer schema. The server will check for a valid token, and if it is available, the user is allowed to access protected routes, services and resources that require that token. As JWT is self-contained, all needed data stay there, leading to less works toward the database.

4. 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 analyses 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. REST determines how the API looks like. It stands for "Representational State Transfer" an abstract definition of the most popular web service technology. REST is a way for two computer systems to communicate over HTTP in a similar way to web browsers and servers. It is a set of rules that developers follow when they create their API.

Table 1. Product related API

Method	Route path	Description
GET	api/product/:productId	Return a product with the given ID under json format
POST	api/product/create/:userId	Create a new product and save it to database, return a json object of that product. Required to be admin and be authenticated
DELETE	api/product/:productId/:userId	Remove a product with the given ID and return a json message object. Required to be admin and be authenticated
PUT	api/product/:productId/:userId	Update a product with the given ID and return updated product under json object. Required to be admin and be authenticated
GET	api/products	Return a list of products under json format
GET	api/products/search	Return a list of products based on search query under json format
GET	api/products/related/sort	Return all the products with the same category as the given ID product

Table 1 above illustrates all the product related API routes with corresponding method and description in brief. As can be seen, only user with admin role can perform operation like creating, deleting and updating to a product. Other public routes can be accessed wherever and by whatever kind of user even without a signed account.

Figure 16. Usage of $regex in MongoDB

Regular Expressions are frequently used in all languages to search for a pattern or word in any string. MongoDB also provides functionality of regular expression for string pattern matching using the **$regex** operator. MongoDB uses PCRE (Perl Compatible Regular Expression) as regular expression language. This is also applied to other routes which have userId, categoryId and orderId as well since it facilitates the logic implementation.

5. API Testing with Postman

At this moment of the development process, the front-end is not available yet so a third-party tool called Postman is used to handle API testing.

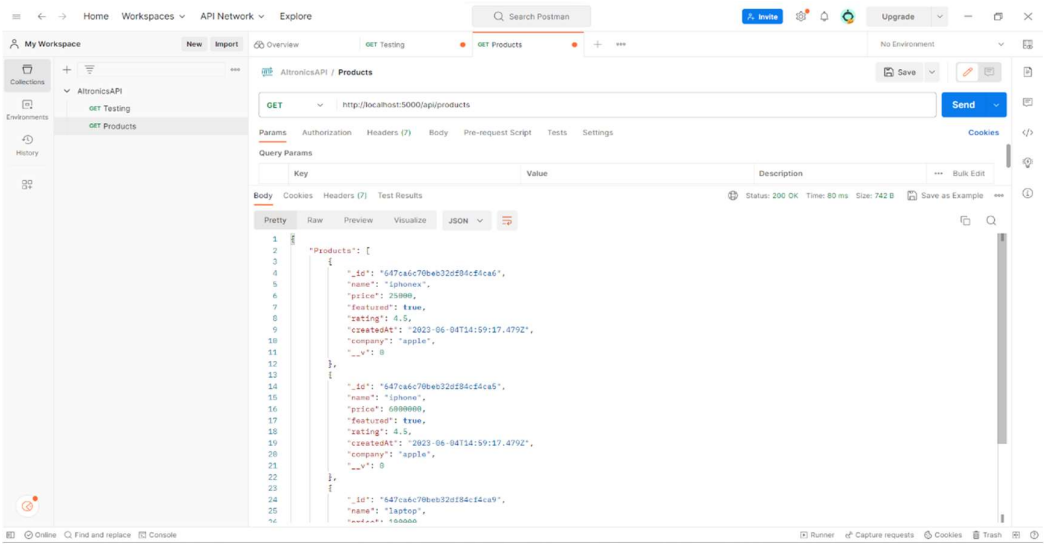


Figure 17. Testing with Postman.

As can be seen from figure 17, a GET request has been called to the route ‘api/products’ and been successfully handled. Since this is a public route and categories is not protected resource, no authorization token in the header is needed. As a result, an array of json object containing all the categories is returned. Each category object includes all the properties defined in Mongoose schema like name, timestamps and auto generated ID which can be used to perform searching as ‘api/products?name=phone’.

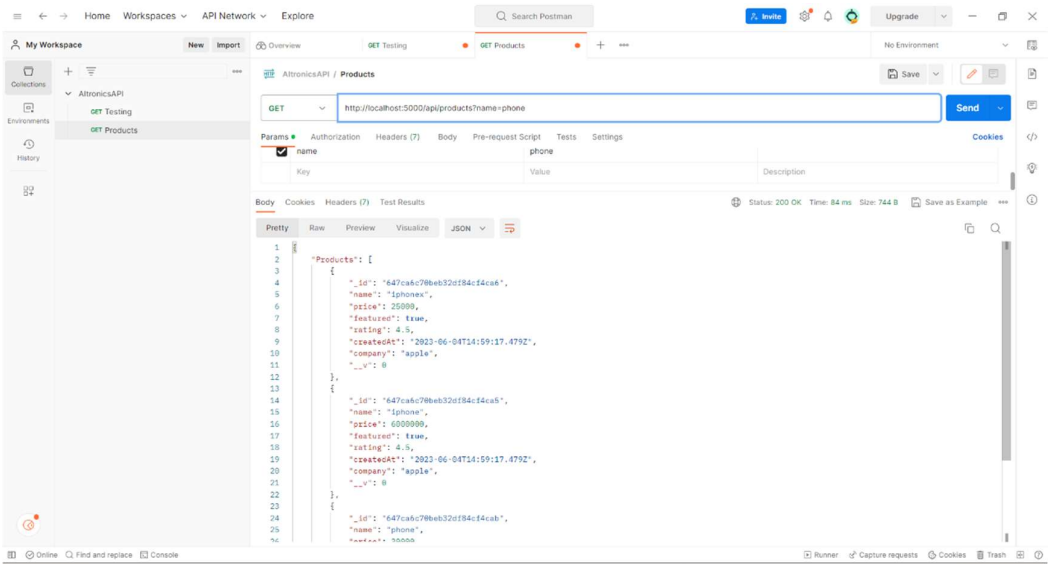


Figure 18. Searching using name ID in Postman.

3.2.2 Front-end Development

1. Basic Setup and Routing

Unlike back-end is performed with multiple technologies, front-end development is handled by only React.js. It is not impossible to set up a React application from scratch, but this process includes of many intimidating and time-consuming steps, such as setting up Babel to convert JSX to compatible JavaScript that old browser can understand and configuring Webpack to bundle all the modules. Fortunately, 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, styled components – 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 ReactDom as the root component of the application.

```
  return (
    <ThemeProvider theme={theme}>
      <Router>
        <GlobalStyle />
        <Header />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/products" element={<Products />} />
          <Route path="/contact" element={<Contact />} />
          <Route path="/singleproduct/:id" element={<SingleProduct />} />
          <Route path="/cart" element={<Cart />} />
          <Route path="*" element={<ErrorPage />} />
        </Routes>
        <Footer />
      </Router>
    </ThemeProvider>
  );
};

export default App;
```

Figure 19. A screenshot of App.js file, with Router wraps around Switch and Route

Figure19. summarizes all the route paths and the component to be render when that route is accessed. For example, any kind of user can go to public route like '/product' to access Product component, but only authenticated user can see Dashboard or Profile component as they are protected routes.

3. Structure of the project

3. Application Elements

3.1 index.html

index.html

public > index.html > ...

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>

```

Figure 20. Screenshot of index.html

Figure 20 represents the HTML file's core component. “.js files” include definitions for all the major functions. The external CSS sources used in this project are stored in this file. "root" is the id for the web app.

3.2 Styled Components in React

styled-components are the result of wondering how we could enhance CSS for styling React component systems. By focusing on a single use case, we managed to optimize the experience for developers as well as the output for end users. It removes the mapping between components and styles. This means that when you're defining your styles, you're actually creating a normal React component, that has your styles attached to it.

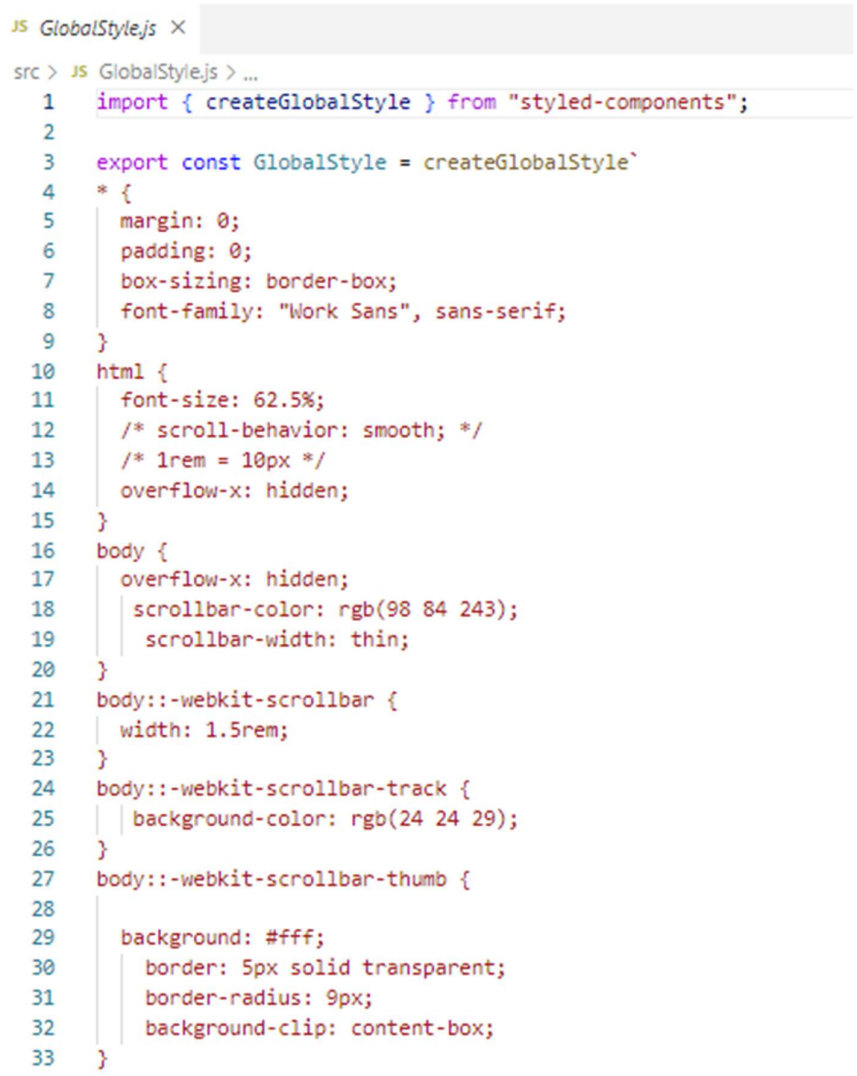


Figure 21. Screenshot of GlobalStyle.js file

Figure 21. represents the GlobalStyle file's core component. ".js files" include definitions for all the major functions. The CSS rules are automatically vendor prefixed; styled-components takes care of that for you! Styled components use stylis.js package under the hood for prefixing the css rules.

3.3 JavaScript and Components in React

App.js

src > App.js > default

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import About from "./About";
import Home from "./Home";
import Products from "./Products";
import Contact from "./Contact";
import Cart from "./Cart";
import SingleProduct from "./SingleProduct";
import ErrorPage from "./ErrorPage";
import { GlobalStyle } from "./GlobalStyle";
import { ThemeProvider } from "styled-components";
import Header from "./components/Header";
import Footer from "./components/Footer";

const App = () => {
  const theme = {
    colors: {
      heading: "rgb(24 24 29)",
      text: "rgba(29 ,29, 29, .8)",
      white: "#fff",
      black: " #212529",
      helper: "#8490ff",
    },
    media: {
      mobile: "768px",
      tab: "998px",
    },
  };

  return (
    <ThemeProvider theme={theme}>
      <Router>
        <GlobalStyle />
        <Header />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/products" element={<Products />} />
          <Route path="/contact" element={<Contact />} />
          <Route path="/singleproduct/:id" element={<SingleProduct />} />
          <Route path="/cart" element={<Cart />} />
          <Route path="*" element={<ErrorPage />} />
        </Routes>
        <Footer />
      </Router>
    </ThemeProvider>
  );
};

export default App;
```

Figure 22. Screenshot of App.js file

Figure 22. displays the subsequent list of additional components. Regarding the website's layout, there are three major components: Footer, Header, and Layout. The page's header is shown by using Header component. Accordingly, the page's footer can be displayed by using Footer component. Layout component helps combining the Header and Footer, also content that is provided in the Layout. Loader component is basically the animated spinner that is shown when loading assets. CartPage component provides the list of ready-to-checkout products. Using Homepage component, customers can see a list of things with accompanying photographs, as well as filter products by name and category. OrdersPage is the page where users can view their placed orders. By instance, the ProductPage views zoomed pictures and detailed of the product, including its name, category, and price.

index.js

src > index.js > ...

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { AppProvider } from "./context/productcontex";
import { FilterContextProvider } from "./context/filter_context";
import { CartProvider } from "./context/cart_context";
import { Auth0Provider } from "@auth0/auth0-react";

const root = ReactDOM.createRoot(document.getElementById("root"));

const domain = process.env.REACT_APP_AUTH_DOMAIN;
const clientId = process.env.REACT_APP_CLIENT_ID;

root.render(
  <Auth0Provider
    domain={domain}
    clientId={clientId}
    redirect_uri={window.location.origin}>
    <AppProvider>
      <FilterContextProvider>
        <CartProvider>
          <App />
        </CartProvider>
      </FilterContextProvider>
    </AppProvider>
  </Auth0Provider>
);

reportWebVitals();
```

Figure 23. Screenshot of index.js file

Figure 23. Displays the subsequent list of Login page of the website, requiring a valid email and password. Users can redirect to Register page to create a new account by providing an email valid address, password, and re-entered password.

3.4 Redux

For JavaScript applications, Redux is a reliable state management solution. It aids in the development of programs that are reliable, run in a variety of settings (client, server, and native), and are simple to test. The Elm language and Facebook's Flux architecture were the inspiration for Redux. As a result, Redux is frequently used in collaboration with React.

The working way of Redux is very simple. There is a central "store" that holds the entire state of the application. Each component can access the stored state without having to send it from one component to another. There are three things need to be constructed: actions, store, and reducers.

In this project, Redux is mainly used to manage the loading, adding and deleting state of the products in store and cart. Here are some illustrations of the way Redux was used in the project.

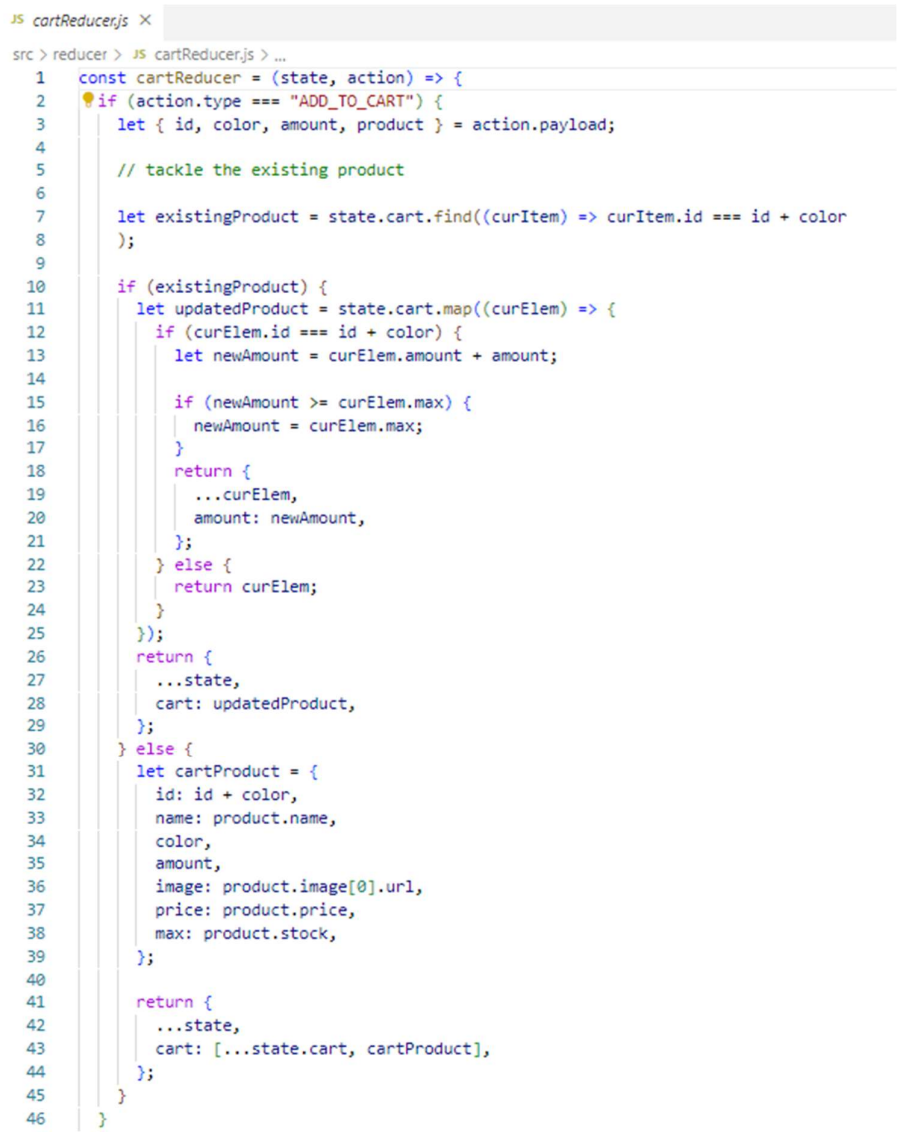


Figure 24. Screenshot of cartReducer.js redux file

filterReducer.js

src > reducer > filterReducer.js > filterReducer

```
const filterReducer = (state, action) => {
  switch (action.type) {
    case "LOAD_FILTER_PRODUCTS":
      let priceArr = action.payload.map((curElem) => curElem.price);
      console.log(
        "🚀 ~ file: filterReducer.js ~ line 5 ~ filterReducer ~ priceArr",
        priceArr
      );

      let maxPrice = Math.max(...priceArr);
      console.log(
        "🚀 ~ file: filterReducer.js ~ line 23 ~ filterReducer ~ maxPrice",
        maxPrice
      );

      return {
        ...state,
        filter_products: [...action.payload],
        all_products: [...action.payload],
        filters: { ...state.filters, maxPrice, price: maxPrice },
      };

    case "SET_GRID_VIEW":
      return {
        ...state,
        grid_view: true,
      };

    case "SET_LIST_VIEW":
      return {
        ...state,
        grid_view: false,
      };
```

Figure 25. Screenshot of filterReducer.js redux file

productReducer.js

src > reducer > productReducer.js > ...

```
const ProductReducer = (state, action) => {

  switch (action.type) {
    case "SET_LOADING":
      return {
        ...state,
        isLoading: true,
      };

    case "SET_API_DATA":
      const featureData = action.payload.filter((curElem) => {
        return curElem.featured === true;
      });

      return {
        ...state,
        isLoading: false,
        products: action.payload,
        featureProducts: featureData,
      };

    case "API_ERROR":
      return {
        ...state,
        isLoading: false,
        isError: true,
      };
```

Figure 26. Screenshot of productReducer.js redux file

3.5 Context Hook

Context provides a way to pass data through the component tree without having to pass props down manually at every level. In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g., locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

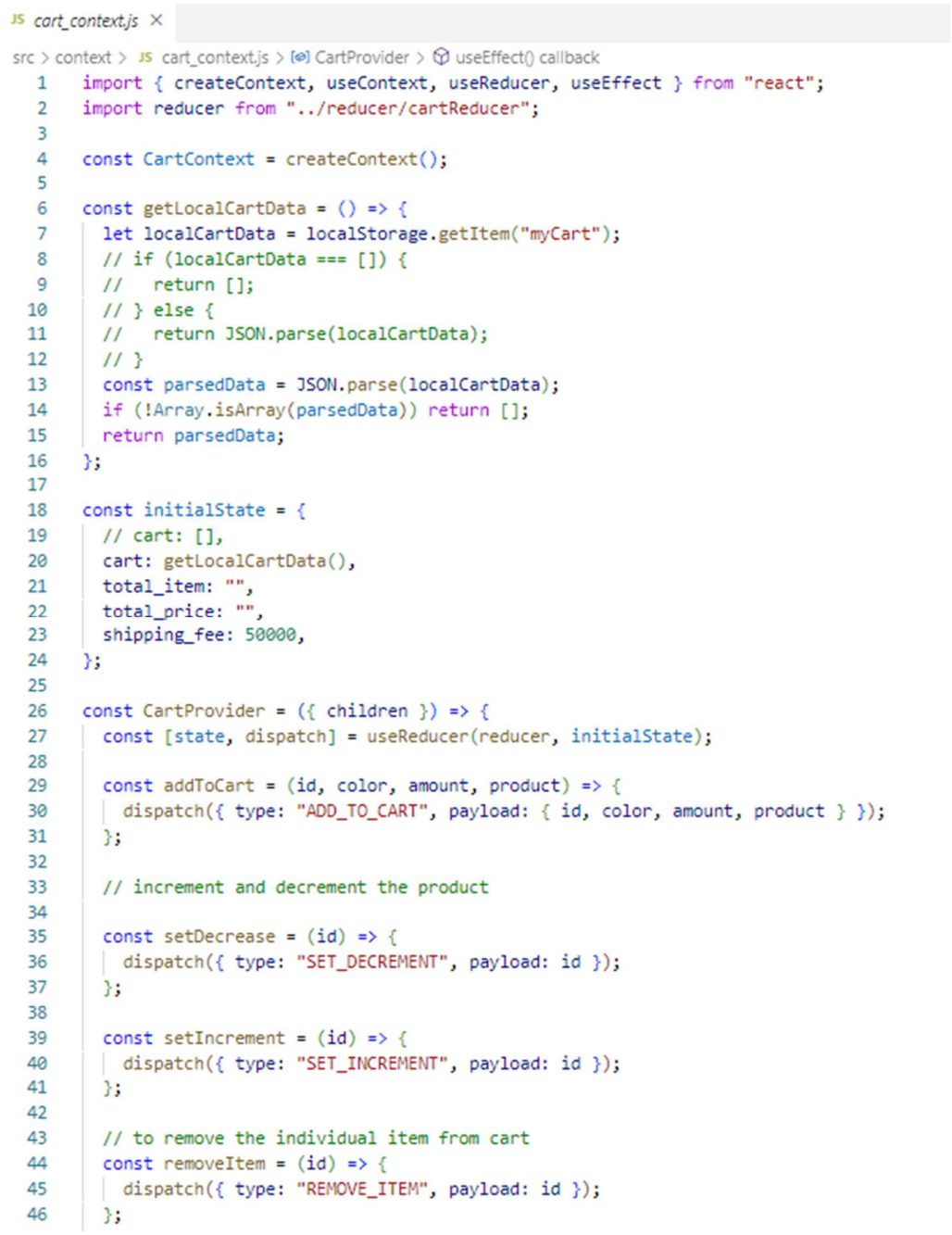


Figure 27. Screenshot of cartContext.js context file.

filter_context.js

src > context > filter_context.js > initialState

```
import { createContext, useContext, useReducer, useEffect } from "react";
import { useProductContext } from "./productcontex";
import reducer from "../reducer/filterReducer";

const FilterContext = createContext();

const initialState = {
  filter_products: [],
  all_products: [],
  grid_view: true,
  sorting_value: "lowest",
  filters: {
    text: "",
    category: "all",
    company: "all",
    color: "all",
    maxPrice: 0,
    price: 0,
    minPrice: 0,
  },
};

export const FilterContextProvider = ({ children }) => {
  const { products } = useProductContext();

  const [state, dispatch] = useReducer(reducer, initialState);

  // to set the grid view
  const setGridView = () => {
    return dispatch({ type: "SET_GRID_VIEW" });
  };

  // to set the list view
  const setListView = () => {
    return dispatch({ type: "SET_LIST_VIEW" });
  };

  // sorting function
  const sorting = (event) => {
    let userValue = event.target.value;
    dispatch({ type: "GET_SORT_VALUE", payload: userValue });
  };
```

Figure 28. Screenshot of filterContext.js context file.

Figure 28. Displays the Context hook is primarily used when some data needs to be accessible by many components at different nesting levels. Apply it sparingly because it makes component reuse more difficult.

productcontex.js

src > context > productcontex.js > AppProvider

```
import { createContext, useContext, useEffect, useReducer } from "react";
import axios from "axios";
import reducer from "../reducer/productReducer";

const AppContext = createContext();

const API = "https://localhost:5000/api/products";

const initialState = {
  isLoading: false,
  isError: false,
  products: [],
  featureProducts: [],
  isSingleLoading: false,
  singleProduct: {},
};

const AppProvider = ({ children }) => {
  const [state, dispatch] = useReducer(reducer, initialState);

  const getProducts = async (url) => {
    dispatch({ type: "SET_LOADING" });
    try {
      const res = await axios.get(url);
      const products = await res.data;
      dispatch({ type: "SET_API_DATA", payload: products });
    } catch (error) {
      dispatch({ type: "API_ERROR" });
    }
  };

  // my 2nd api call for single product

  const getSingleProduct = async (url) => {
    dispatch({ type: "SET_SINGLE_LOADING" });
    try {
      const res = await axios.get(url);
      const singleProduct = await res.data;
      dispatch({ type: "SET_SINGLE_PRODUCT", payload: singleProduct });
    } catch (error) {
      dispatch({ type: "SET_SINGLE_ERROR" });
    }
  };
```

Figure 29. Screenshot of productContext.js context file

Figure 29. If you only want to avoid passing some props through many levels, component composition is often a simpler solution than context. Creates a Context object. When react renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above it in the tree.

3.6 Home Component

Home.js

src > Home.js > Home > data > name

```
1   import FeatureProduct from "./components/FeatureProduct";
2   import HeroSection from "./components/HeroSection";
3   import Services from "./components/Services";
4   import Trusted from "./components/Trusted";
5
6   const Home = () => {
7     const data = {
8       name: "AItronics Store.",
9     };
10
11    const value ={
12      value: "Discover a smarter way to shop for electronics - with AItronics.",
13    }
14
15    return (
16      <>
17        <HeroSection myData={data} myValue={value} />
18        <FeatureProduct />
19        <Services />
20        <Trusted />
21      </>
22    );
23  };
24
25  export default Home;
```

Figure 30. Screenshot of Home.js file

Figure 30. Displays the elements on home page of the website. These elements include a navigation menu for easy browsing, a visually appealing hero banner to showcase featured products or promotions, product listings with images, prices, and discounts, a search bar for quick product searches, featured categories to help users find what they're looking for, testimonials or reviews to build trust, call-to-action buttons to guide users towards desired actions, and a footer section with important links and information. It's crucial to prioritize clean design, optimize page loading speed, and ensure responsiveness across devices. Regular analysis of user behaviour can help make data-driven improvements for an enhanced user experience on the e-commerce home page.

3.7 Product Component

SingleProduct.js 2

src > SingleProduct.js > SingleProduct

```
import { useEffect } from "react";
import styled from "styled-components";
import { useParams } from "react-router-dom";
import { useProductContext } from "./context/productcontex";
import PageNavigation from "./components/PageNavigation";
import MyImage from "./components/MyImage";
import { Container } from "./styles/Container";
import FormatPrice from "./Helpers/FormatPrice";
import { MdSecurity } from "react-icons/md";
import { TbTruckDelivery, TbReplace } from "react-icons/tb";
import Star from "./components/Star";
import AddToCart from "./components/AddToCart";

const API = "https://localhost:5000/api/products";

const SingleProduct = () => {
  const { getSingleProduct, isSingleLoading, singleProduct } =
    useProductContext();

  const { id } = useParams();

  const {
    id: alias,
    name,
    company,
    price,
    description,
    stock,
    stars,
    reviews,
    image,
  } = singleProduct;

  useEffect(() => {
    getSingleProduct(`${API}?id=${id}`);
  },[]
  );

  if (isSingleLoading) {
    return <div className="page_loading">Loading.....</div>;
  }

  return (
```

Figure 30. Screenshot of product.js file

Figure 30. The product component for an e-commerce website is a crucial element that showcases the product's image, title, price, rating, description, and an 'Add to Cart' button. It provides users with a clear and concise overview of the product, allowing them to make informed decisions. Including product variants, availability status, and customer reviews enhances the user experience. Additionally, incorporating related products can encourage further exploration. Designing a visually appealing and responsive product component is essential to attract and engage users on your e-commerce platform.

3.8 Cart Component

Cart.js

src > Cart.js > Cart

```
import styled from "styled-components";
import { useCartContext } from "./context/cart_context";
import CartItem from "./components/CartItem";
import { NavLink } from "react-router-dom";
import { Button } from "./styles/Button";
import FormatPrice from "./Helpers/FormatPrice";
import { useAuth0 } from "@auth0/auth0-react";

const Cart = () => {
  const { cart, clearCart, total_price, shipping_fee } = useCartContext();


  const { isAuthenticated, user } = useAuth0();

  if (cart.length === 0) {
    return (
      <EmptyDiv>
        <h3>No Cart in Item </h3>
      </EmptyDiv>
    );
  }

  return (
    <Wrapper>
      <div className="container">
        {isAuthenticated && (
          <div className="cart-user--profile">
            <img src={user.profile} alt={user.name} />
            <h2 className="cart-user--name">{user.name}</h2>
          </div>
        )}

        <div className="cart_heading grid grid-five-column">
          <p>Item</p>
          <p className="cart-hide">Price</p>
          <p>Quantity</p>
          <p className="cart-hide">Subtotal</p>
          <p>Remove</p>
        </div>
        <hr />
        <div className="cart-item">
          {cart.map((curElem) => {
            return <CartItem key={curElem.id} {...curElem} />;
          })}
        </div>
        <hr />
        <div className="cart-two-button">
          <NavLink to="/products">
            <Button> continue Shopping </Button>
          </NavLink>
          <Button className="btn btn-clear" onClick={clearCart}>
            clear cart
          </Button>
        </div>
```

Figure 31. Screenshot of cart.js file

Figure 31. The cart component displays the product details, quantities, subtotals, and a total cost. Users can easily adjust quantities, remove items, and proceed to checkout. Additionally, a clear message is shown when the cart is empty, prompting users to continue shopping. The cart component should be responsive, allowing seamless usage on mobile devices. Including features like a save for later option enhances the user experience, making it convenient to manage and review chosen items before finalizing the purchase.

3.9 IDE

Visual Studio is an **Integrated Development Environment (IDE)** developed by Microsoft to develop GUI (Graphical User Interface), console, Web applications, web apps, mobile apps, cloud, and web services, etc. With the help of this IDE, you can create managed code as well as native code. It uses the various platforms of Microsoft software development software like Windows store, Microsoft Silverlight, and Windows API, etc. It is not a language-specific IDE as you can use this to write code in C#, C++, VB (Visual Basic), Python, JavaScript, and many more languages. It provides support for 36 different programming languages. It is available for Windows as well as for macOS. **Evolution of Visual Studio:** The first version of VS (Visual Studio) was released in 1997, named as Visual Studio 97 having version number 5.0. The latest version of Visual Studio is 15.0 which was released on March 7, 2017. It is also termed as Visual Studio 2017. The supported *.Net* Framework Versions in latest Visual Studio is 3.5 to 4.7. Java was supported in old versions of Visual Studio but in the latest version doesn't provide any support for Java language.

VS Code is a free code editor, which runs on the macOS, Linux, and Windows operating systems.

Follow the platform-specific guides below:

- macOS
- Linux
- Windows

VS Code is lightweight and should run on most available hardware and platform versions. You can review the System Requirements to check if your computer configuration is supported.

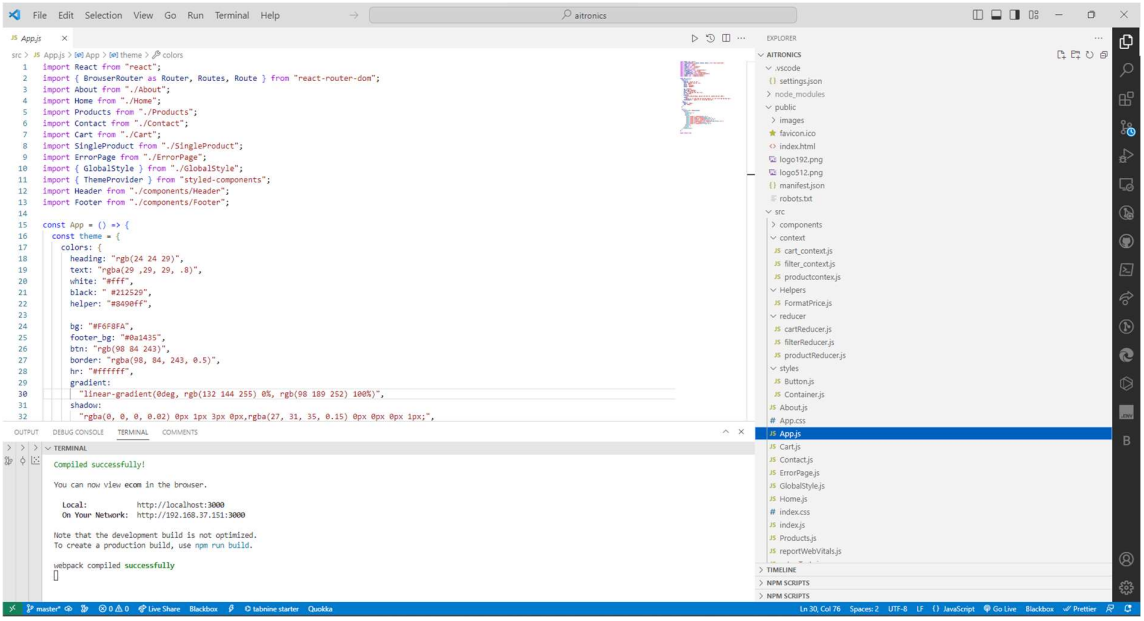


Figure 32. Screenshot of Visual Studio Code IDE.

3.10 Hosting

MongoDB Atlas is a fully managed cloud database service provided by MongoDB. It allows you to deploy, scale, and manage MongoDB databases on various cloud platforms.

To host your database on MongoDB Atlas, follow these steps:

- Sign up for MongoDB Atlas: Go to the MongoDB Atlas website (https://www.mongodb.com/cloud/atlas) and sign up for an account if you haven't already. You may need to provide some basic information and choose a cloud provider and region.

- Create a new cluster: Once you're logged in, click on the "Build a New Cluster" button to create a new cluster. Choose the cloud provider, region, and other configuration options that best suit your needs. You can also select the cluster tier based on the performance and storage requirements of your application.

- Configure cluster settings: Set a name for your cluster and configure any additional settings, such as enabling backup options, enabling encryption at rest, etc. You can also choose the version of MongoDB you want to use.

- Set up network access: In the "Security" section, configure the network access settings to allow incoming connections to your cluster. You can specify IP addresses, IP ranges, or allow access from anywhere (0.0.0.0/0) if you want to make it publicly accessible.

- Create a database user: In the "Database Access" section, create a new database user with a username and password. This user will be used to authenticate your application's connection to the database.

- Connect to your cluster: Once your cluster is created and running, click on the "Connect" button. You can choose to connect using MongoDB Compass, the MongoDB shell, or your application code. Follow the instructions provided based on your preferred method of connection.

- Update your application configuration: Update your application's database connection string or configuration file with the connection details obtained from the previous step. Make sure to include the username, password, cluster address, and other necessary information.

Your database is now hosted on MongoDB Atlas, and you can start using it for your application. MongoDB Atlas provides various features like automated backups, monitoring, scaling, and more to help you manage your database effectively. Remember to secure your connection details and follow best practices for database security.

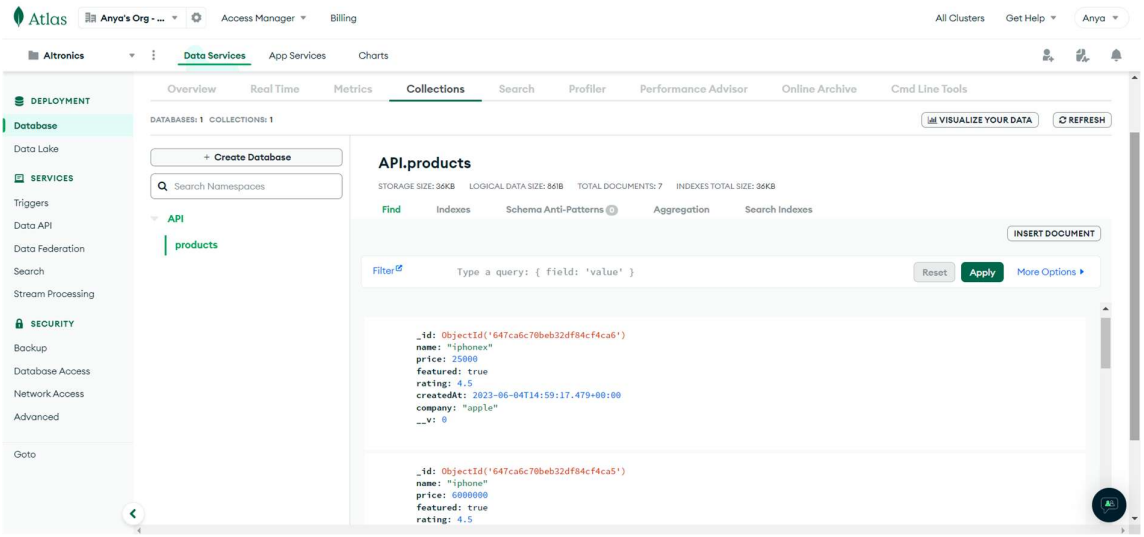


Figure 32. Screenshot of MongoDB Atlas Cloud

Figure 32. Atlas Cloud is an online platform that allows developers to view their Atlas projects and track their Atlas GitHub Action CI runs. The cloud platform gives full visibility into your Atlas schema, by displaying an entity relation diagram (ERD) that shows the schema changes, as well as the entire database schema.

4. ER Diagram

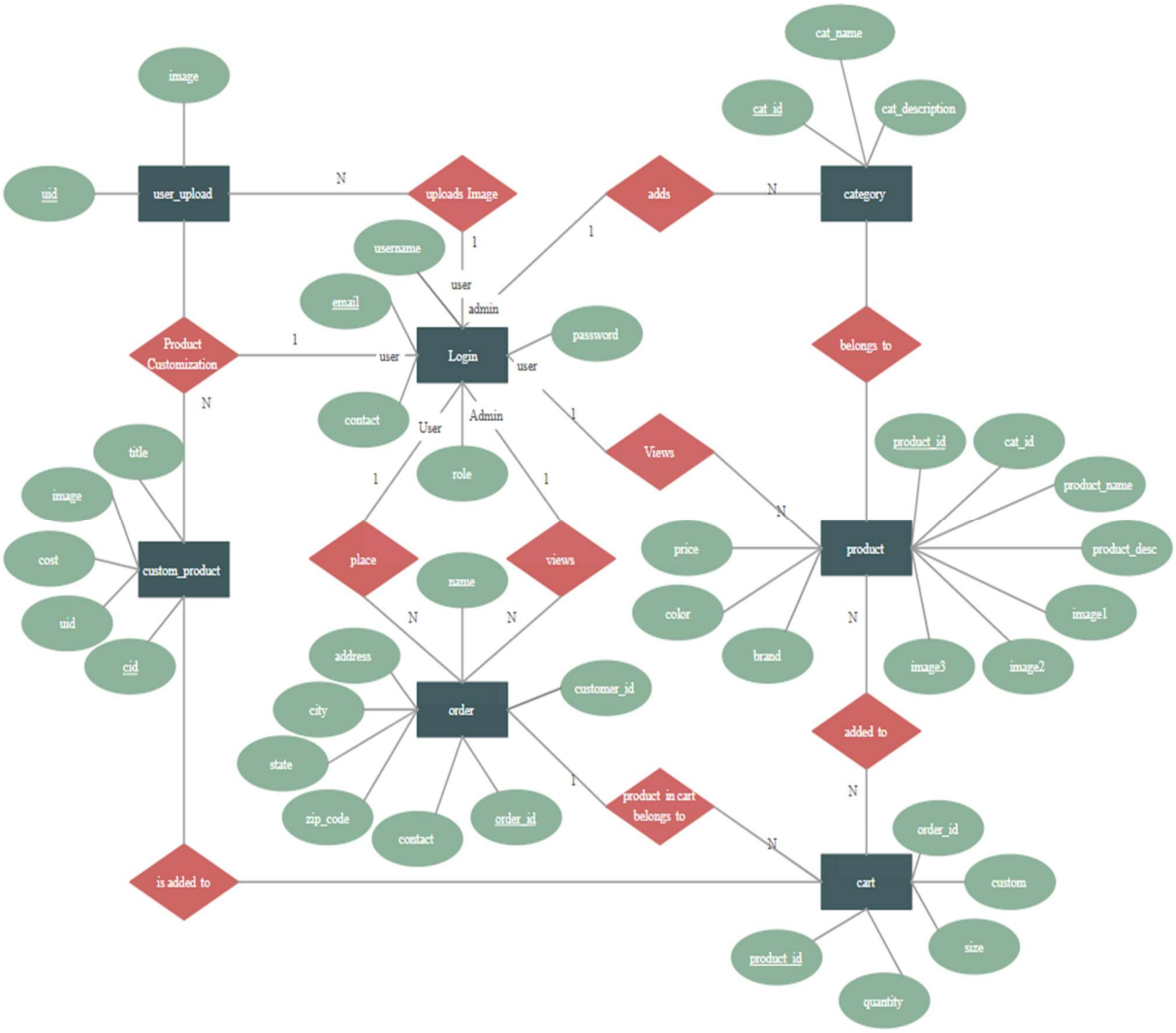


Figure 33. ER Diagram of the E-Commerce Website.

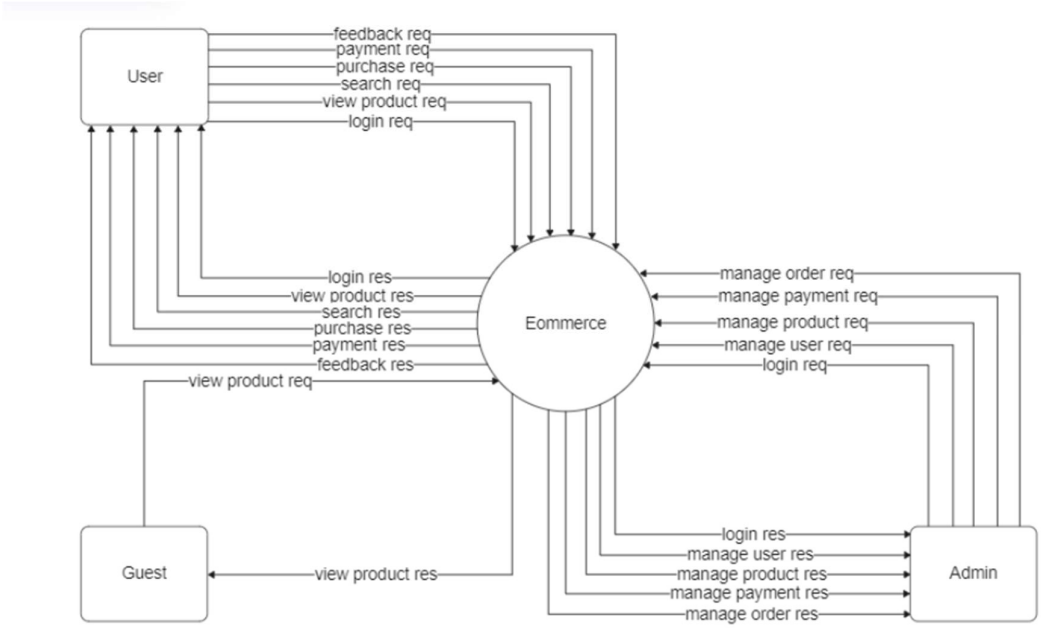


Figure 34. 0 Level Data Flow Diagram

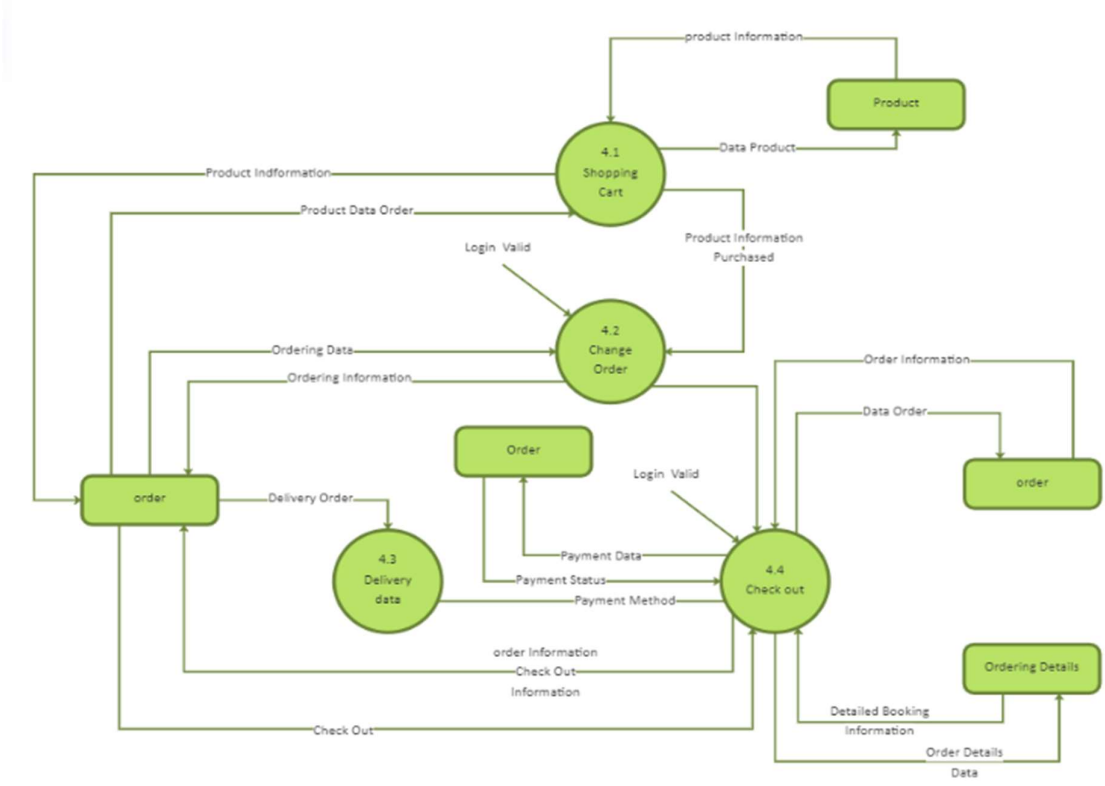


Figure 35. 1 Level Data Flow Diagram

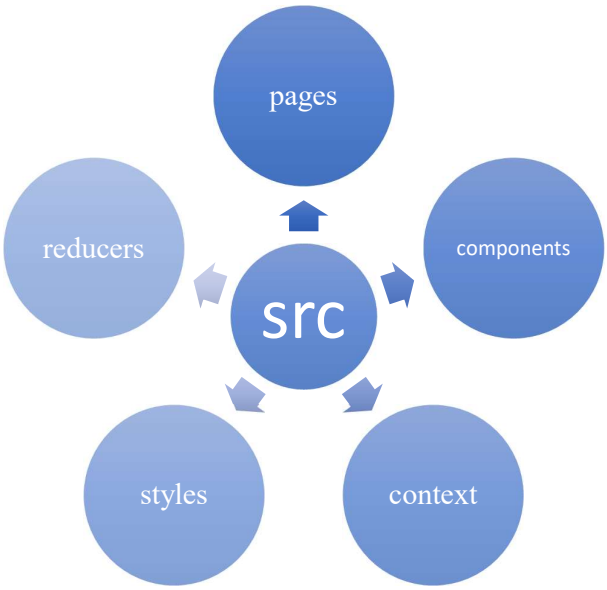


Figure 36. Relation Schema of Folders

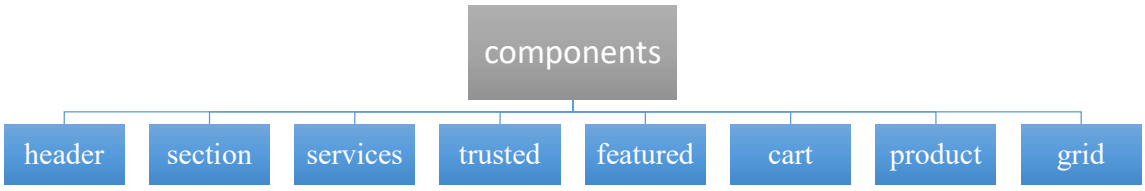


Figure 37. Entity Schema of Components

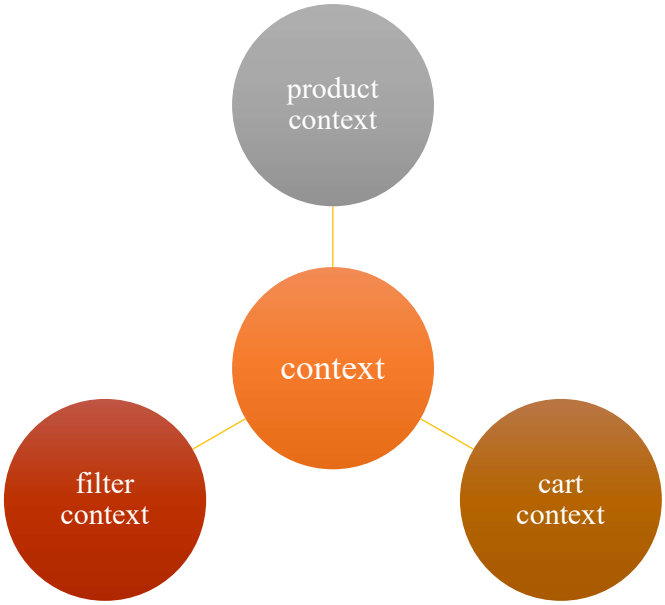


Figure 38. Entity Schema of Context Files

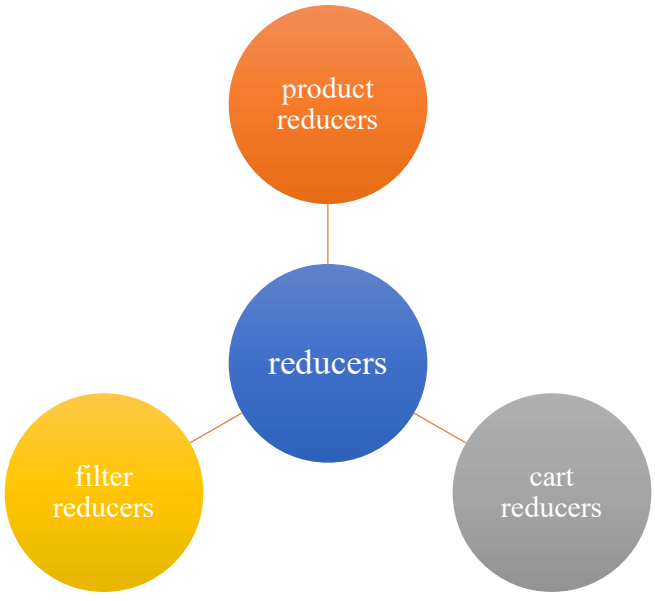


Figure 39. Entity Schema of Reducers Files

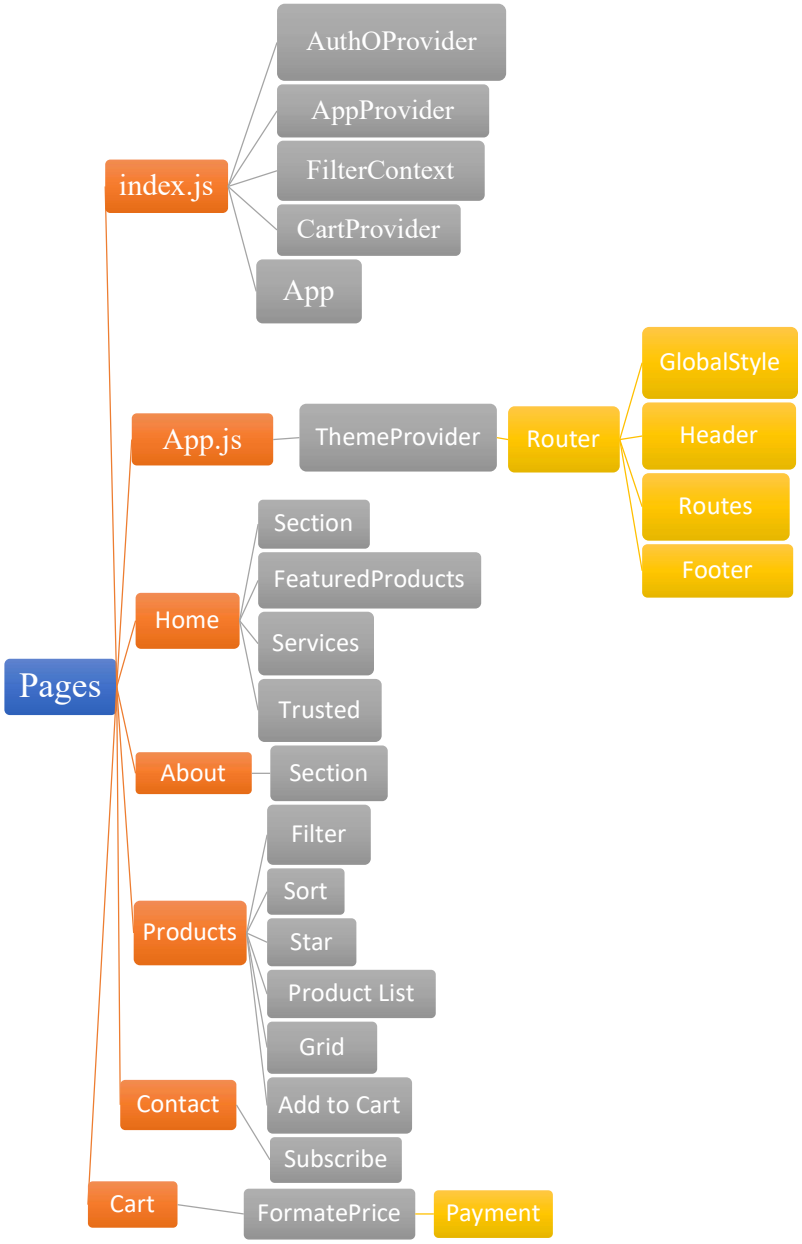
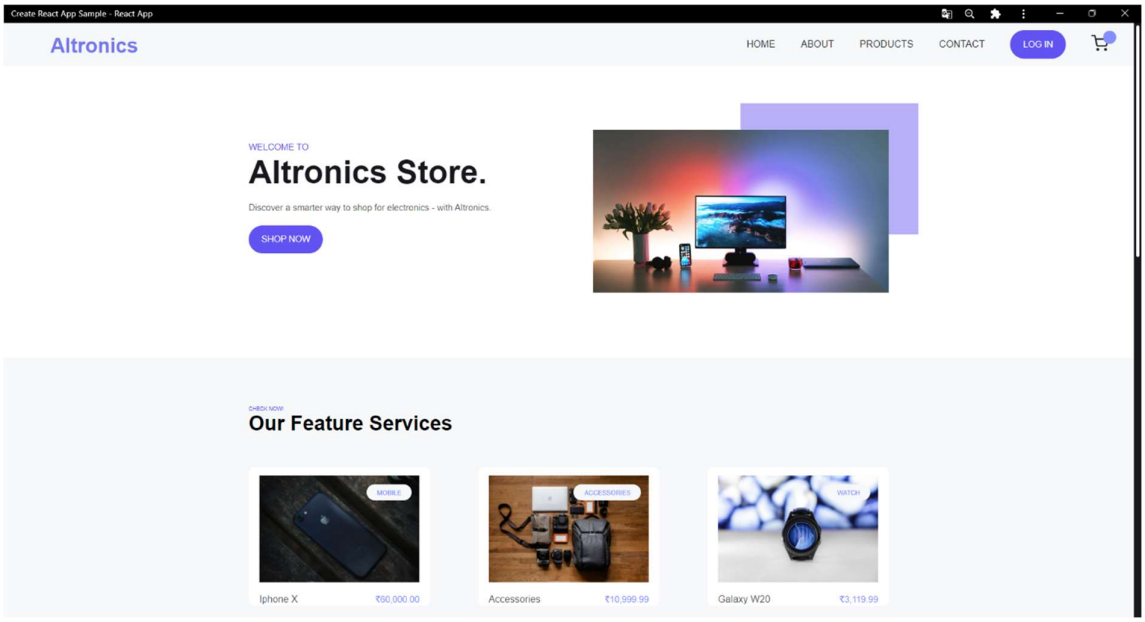


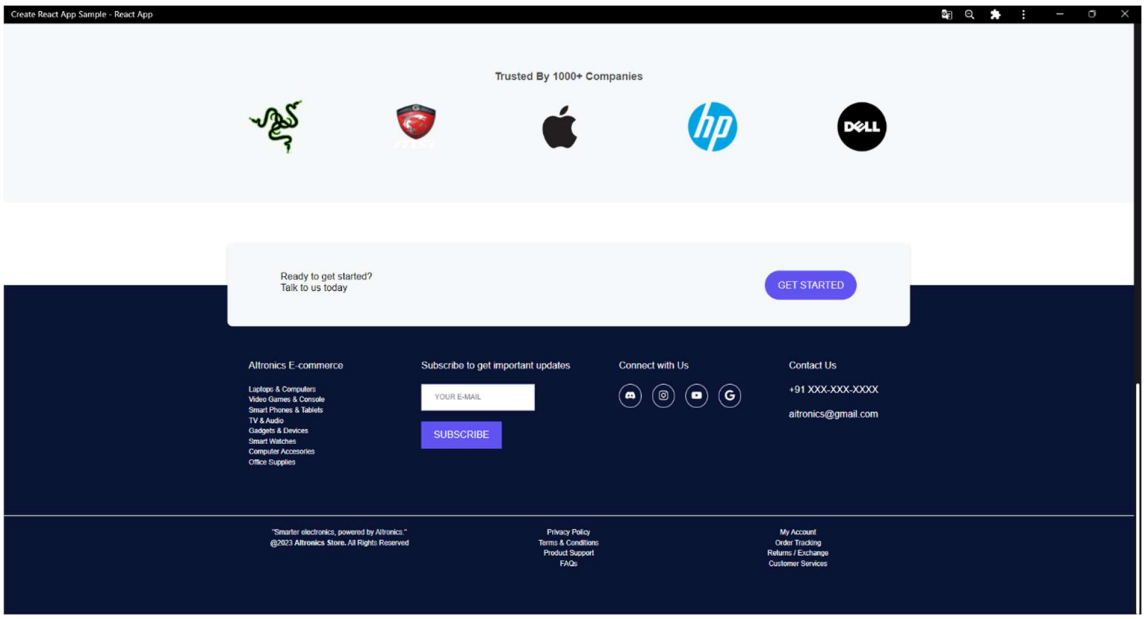
Figure 40. Entity Schema of Web Pages

5. User Interface Design

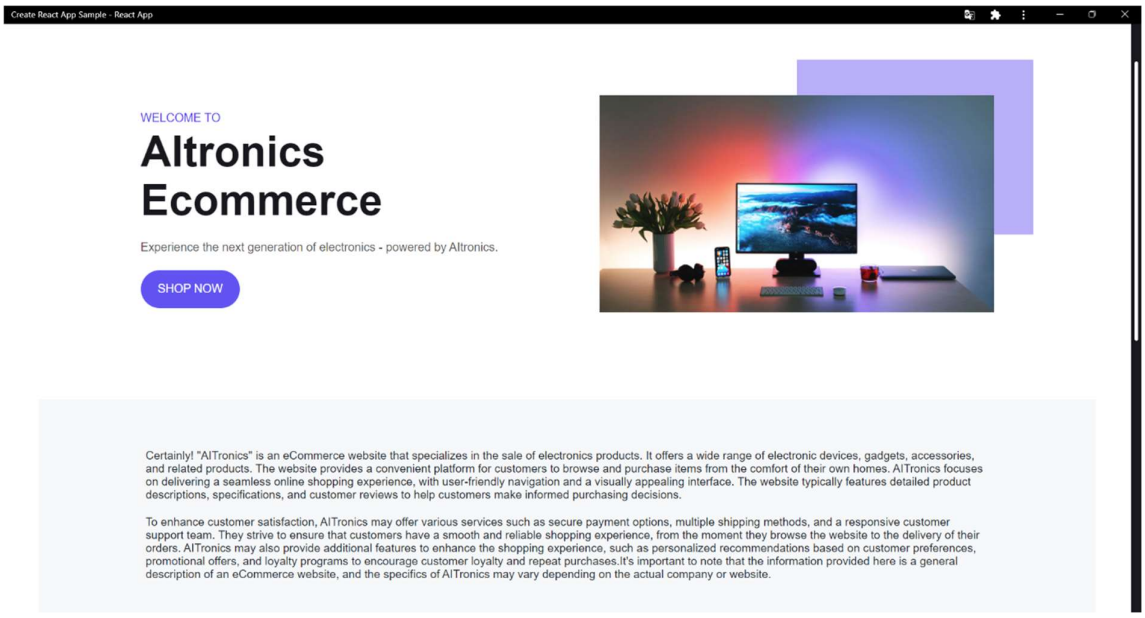
The following pictures illustrate the outcome of the project – a webstore prototype with simple user interface.



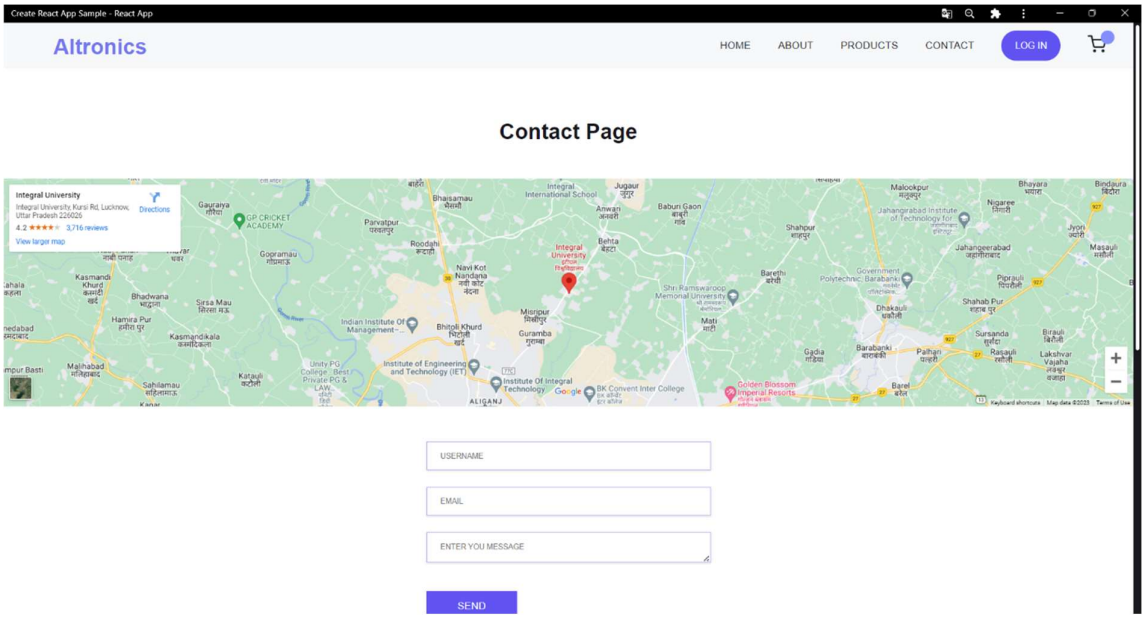
Picture 1. Home Page



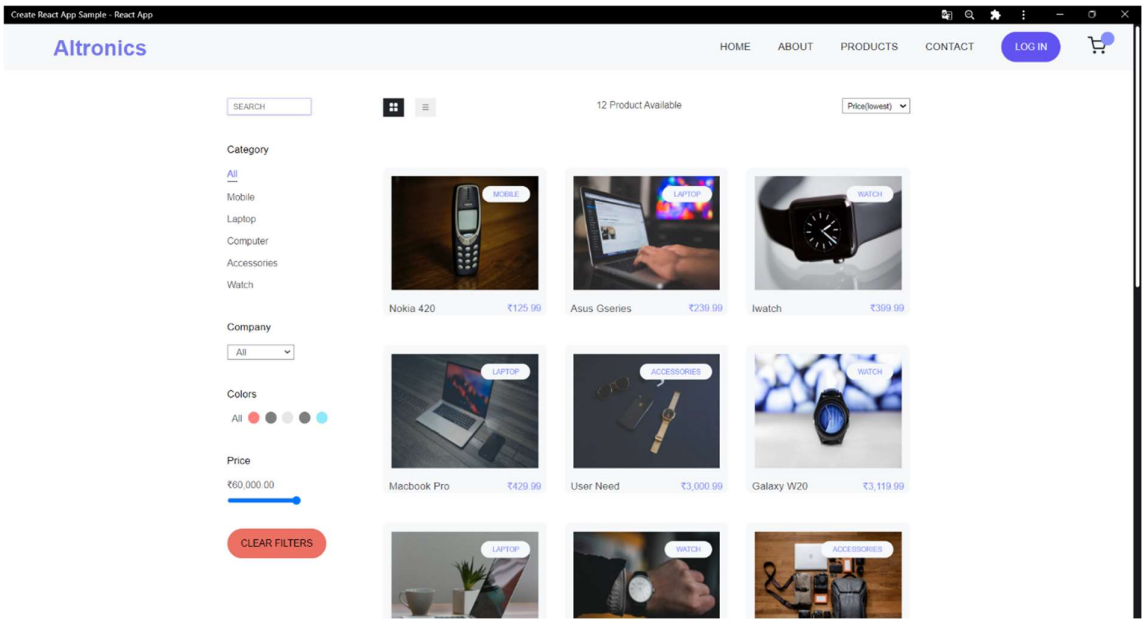
Picture 2. Footer Section



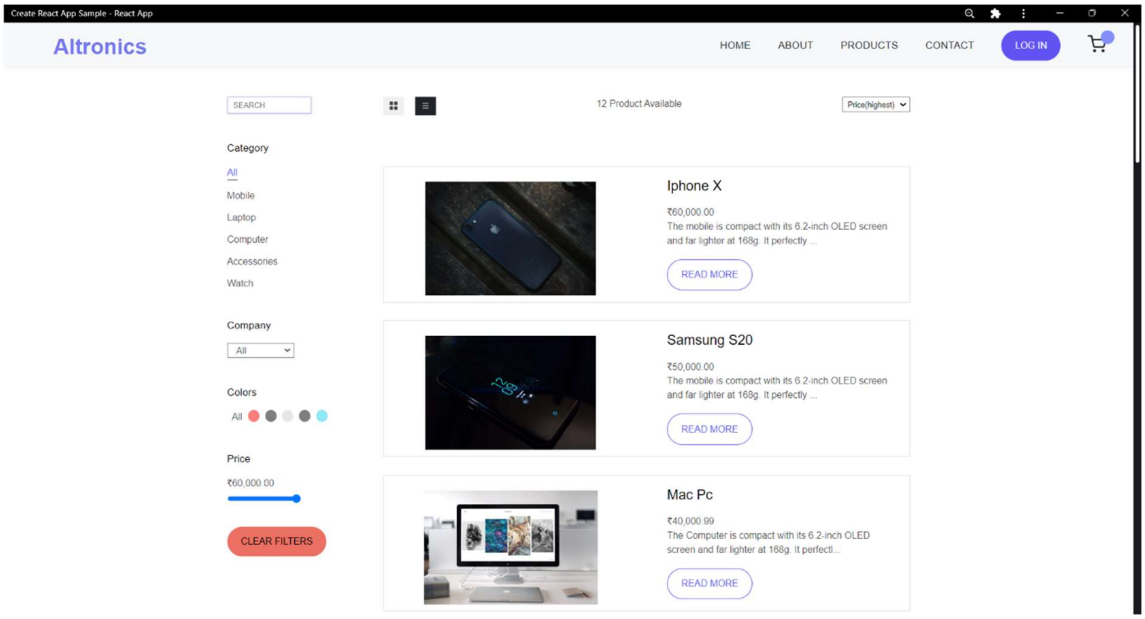
Picture 3. About Section



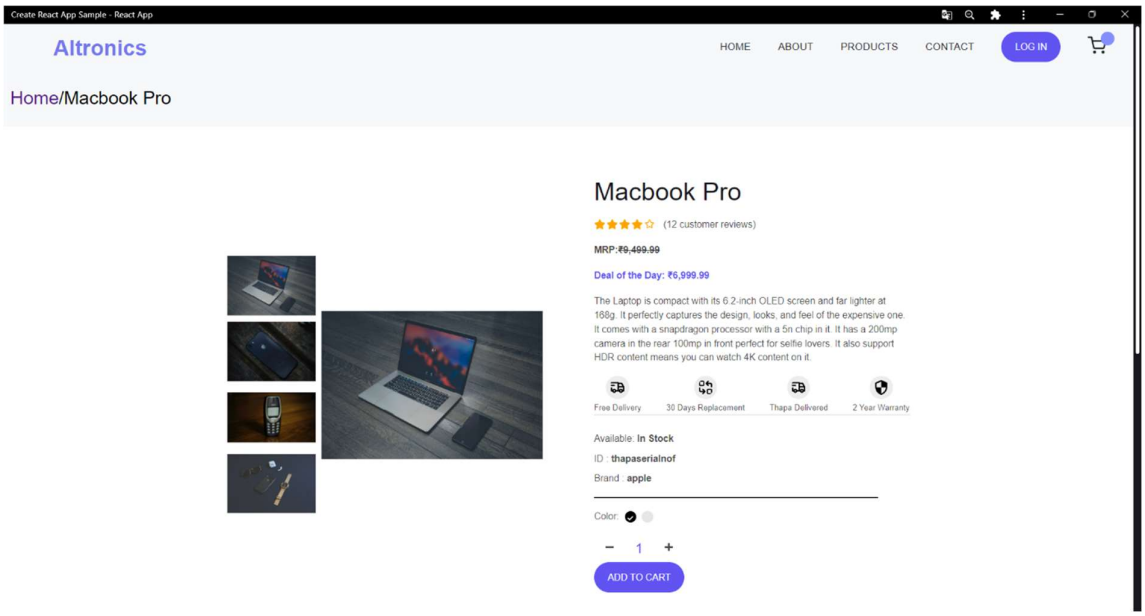
Picture 4. Contact Page



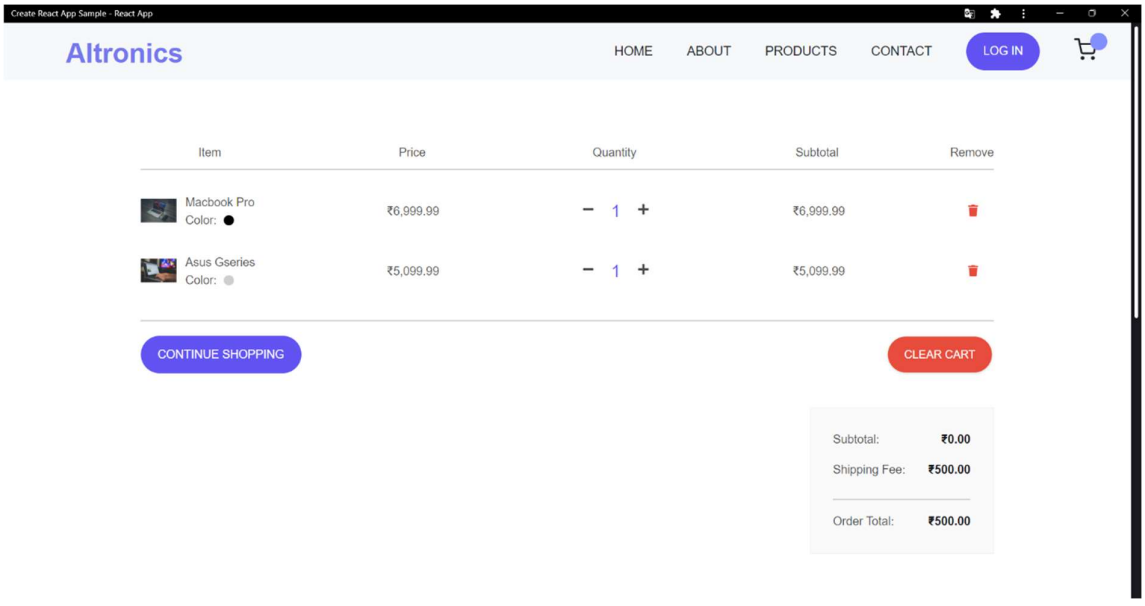
Picture 5. Products Page



Picture 6. Featured Products Page



Picture 6. Products Order Page



Picture 7. Cart and Payment Page

4. Discussion

4.1 Application Evaluation

Performance is critical factor to evaluate. It's important to test the application's response time, page loading speed, and scalability. An ecommerce platform should be able to handle the expected user load and data volume without significant performance degradation. Slow performance can lead to user frustration and cart abandonment, so it's crucial to ensure the application performs well under typical usage scenarios. Security is of utmost importance in an ecommerce application. It's essential to evaluate the security measures implemented, such as secure user authentication, encryption of sensitive data (e.g., passwords and payment details), and protection against common vulnerabilities like cross-site scripting (XSS) and SQL injection. Robust security measures are vital to protect user data and maintain customer trust.

The backend architecture and code quality should be carefully evaluated. Assess the backend implementation using Node.js and Express.js, looking for modular and well-structured code, efficient database queries, proper error handling, and appropriate use of middleware. Scalability and performance optimizations should also be considered to ensure that the backend can handle increasing traffic and data requirements. The database, typically MongoDB in a MERN stack application, needs to be evaluated for its design and implementation. Look for proper indexing, efficient querying, and data integrity measures. Assess if the database can handle the expected data volume and evaluate the presence of backup and disaster recovery mechanisms to ensure data safety and availability.

Despite of all the fundamental and useful implemented functionalities in the application, these features in the following list can be considered for further development:

- User sign up using Gmail or Facebook
- Email notification after successful payment
- Each product has a review and rating section

Eventually, in order for this e-commerce application to switch from development mode to production mode, testing and deployment have to be handled appropriately as the development process may have some unseen bugs.

Integration with third-party services and APIs is essential for an ecommerce platform. Evaluate how well the application integrates with payment gateways, shipping providers, and analytics tools. Seamless integration with external services is crucial for a fully functional ecommerce platform and enables smooth transactions and business operations. The documentation and maintenance should not be overlooked. Evaluate the availability and quality of documentation for the application. Clear and up-to-date documentation helps developers understand and maintain the codebase effectively, reducing the learning curve and facilitating future updates, maintenance and enhancements.

4.2 Future Scope

The future scope of the MERN (MongoDB, Express.js, React.js, Node.js) stack in the ecommerce domain is quite promising. Here are some potential advancements and opportunities:

1. Enhanced User Experience: The MERN stack provides a solid foundation for creating highly interactive and responsive user interfaces. In the future, we can expect even more immersive and engaging user experiences in ecommerce applications built with MERN. Advancements in React.js and frontend technologies will enable the creation of dynamic and personalized shopping experiences, resulting in increased customer satisfaction and conversion rates.

2. Scalability and Performance Improvements: As ecommerce businesses grow, the ability to scale the application becomes crucial. The MERN stack, with its modular and flexible architecture, allows for horizontal scaling by adding more servers or utilizing cloud-based services. Ongoing improvements in Node.js and MongoDB will further enhance the stack's performance, enabling ecommerce platforms to handle larger user bases, increased traffic, and higher volumes of data.

3. Real-time Analytics and Personalization: The MERN stack, combined with technologies like WebSockets and real-time databases, enables the collection and analysis of real-time data. In the future, ecommerce platforms built with MERN can leverage this capability to provide personalized product recommendations, targeted marketing campaigns, and real-time inventory updates. By utilizing data-driven insights, businesses can deliver a more tailored shopping experience, improving customer engagement and driving sales.

4. Integration with Emerging Technologies: The MERN stack can easily integrate with emerging technologies, opening up opportunities for ecommerce applications to leverage advancements in artificial intelligence (AI), machine learning (ML), augmented reality (AR), and virtual reality (VR). For instance, AI-powered chatbots can enhance customer support, ML algorithms can provide intelligent product recommendations, and AR/VR technologies can enable virtual product try-ons. As these technologies mature, integrating them into the MERN stack will enable innovative and immersive ecommerce experiences.

5. Headless Commerce: Headless commerce, the decoupling of the frontend and backend of an ecommerce application, is gaining traction. The MERN stack's modular architecture makes it well-suited for implementing headless commerce solutions. By separating the frontend and backend, businesses have more flexibility to experiment with different frontend technologies, create custom experiences, and adapt to changing market trends. The future of MERN stack ecommerce will likely include increased adoption of headless commerce architecture to empower businesses with greater customization and agility.

5. Conclusion

The goal of this project was to study different characteristic of each technology in the MERN stack and develop a full-stack e-commerce web application based on it. It took the author a considerable amount of time to research and study in-depth each modern technology in order to carry on the web application. All the technologies that comprise of the MERN stack were discussed precisely, from fundamental concepts to advanced features as well as their usage in the application to ensure the reader understand about this paper. The thesis also provided all the steps needed in the development process to implement an e-commerce application.

The application was successfully developed at the end. A fully functional end-to-end ecommerce application featuring an online bookstore was released and aimed to help the startup develop their business strategy in general. This application was meant to solve the problem that is mentioned in the first section of this thesis: to help the book retailer startup become more widely known and gain more potential customers. People and book lovers also have one more source to expand their knowledge. The stack's popularity and active community support ensure a vast array of resources, tutorials, and libraries available to aid in development. This, coupled with the extensive documentation and community-driven knowledge sharing, makes the MERN stack an excellent choice for major projects. Moreover, the MERN stack offers flexibility in integrating with third-party services and APIs, facilitating seamless integration of payment gateways, shipping providers, and analytics tools, among others. It also allows for easy deployment and hosting on cloud platforms.

Overall, the major project can be used as a tutorial or documentation of the MERN stack in particular or full-stack web development in general. By researching and studying this paper, the author gained much more useful knowledge and understood why MERN stack is rising its popularity and plays a leading role in web development recently. Although the application still has some drawbacks and needs more further improvements, both in styling issue and new features, it is a combination of one of the most widely used web stack technology with one of the most emerging business ideas nowadays – ecommerce.

References

1. Learn JavaScript: https://developer.mozilla.org/en-US/docs/Learn/JavaScript
2. Learn React. What is React.js? https://react.dev/
3. Node.js Documentation: https://nodejs.dev/en/learn/
4. Nodejs API: https://nodejs.org/docs/latest/api/
5. NPM Documentation: https://www.npmjs.com/
6. Mongoose: https://www.npmjs.com/package/mongoose
7. MongoDB Documentation: https://www.mongodb.com/docs/manual/
8. Multer Documentation: https://www.npmjs.com/package/multer
9. Express js Documentation: http://expressjs.com/
10. Download Postman: https://www.postman.com/downloads/
11. Rest API. REST API Tutorial. Available at: https://restfulapi.net/
12. React Training/ React Router. Primary Components. Available at: https://react-training.com/react-router/web/guides/primary-components
13. Stack overflow What does body-parser do with express? Available at: https://stackoverflow.com/questions/38306569/what-does-body-parser-do-with-express
14. Facebook Inc. React Available at: https://reactjs.org/docs/components-and-props.html