# DEVELOPMENT OF MULTI-MODEL INFORMATIONAL WEBSITE USING MERN STACK

**A PROJECT REPORT**

*Submitted by*

**S. RAJESHWARAN**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

*in*

# COMPUTER SCIENCE AND ENGINEERING

# A picture containing text Description automatically generated

**IFET COLLEGE OF ENGINEERING, VILLUPURAM.**

**(An Autonomous Institution)**

# BONAFIDE CERTIFICATE

Certified that this project report “**DEVELOPMENT OF MULTI-MODEL WEBSITE USING MERN STACK**” is the bonafide work of “S. RAJESHWARAN “who carried out the project work under my supervision.

# SIGNATURE

Submitted for the project viva voice examination held on \_

**ACKNOWLEDGEMENT**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CH.NO** | **TITLE** | **PAGE** |
|  | **ABSTRACT** | **VII** |
|  | **LIST OF FIGURES** | **VIII** |
|  | **LIST OF ABBREVATIONS** | **X** |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 OVERVIEW OF PROJECT | **1** |
|  | 1.2 PROBLEM STATEMENT | **4** |
|  | 1.3 EXPLANATION OF MULTI-MODEL WEBSITE | **5** |
|  | 1.4 MERN STACK OVERVIEW | **5** |
| **2.** | **LITERATURE REVIEW** | **7** |
|  | 2.1 A REVIEW OF USABILITY AND SECURITY EVALUATION MODEL OF ECOMMERCE WEB | **7** |
|  | 2.2 E-COMMERCE FOR THE 21ST CENTURY | **7** |
|  | 2.3 OPINION OBSERVER: RECOMMENDATION SYSTEM ON ECOMMERCE WEBSITE | **8** |
|  | 2.4 INTEGRATED APPROACH TO E-COMMERCE WEBSITES EVALUATION WITH THE USE OF SURVEYS AND EYE TRACKING BASED EXPERIMENTS | **9** |
|  | 2.5 LEARNING TO DETECT AND MEASURE FAKE ECOMMERCE WEBSITES IN SEARCH-ENGINE RESULTS | **10** |
|  | 2.6 EVALUATING ECOMMERCE WEBSITES COGNITIVE EFFICIENCY | **11** |
|  | 2.7 THE RESEARCH ON E-COMMERCE WEBSITE SUCCESS MODE | **11** |

|  |  |  |
| --- | --- | --- |
|  | 2.8 DEVELOPMENT OF A DATA ACQUISITION SYSTEM FOR AN ECOMMERCE WEBSITE | **12** |
|  | 2.9 RESEARCH ON THE EVALUATION OF E-COMMERCE WEBSITE UNDER THE ENVIRONMENT OF BIG DATA | **12** |
|  | 2.10 RESEARCH ON E-COMMERCE BUSINESS MODEL BASED ON INTELLIGENT TERMINALS MOBILE | **13** |
|  | 2.11 DESIGNING E-COMMERCE USER INTERFACE | **14** |
| **3.** | **EXISTING SYSTEM** | **21** |
|  | 3.1 SYSTEM ANALYSIS | **21** |
|  | 3.2 DISADVANTAGES | **22** |
| **4.** | **PROPOSED SYSTEM** | **23** |
|  | 4.1 SYSTEM OVERVIEW | **23** |
|  | 4.2 ARCHITECTURE | **23** |
|  | 4.3 REACT JS – FRONT END | **24** |
|  | 4.4 EXPRESS.JS AND NODE.JS SERVER TIER | **24** |
|  | 4.5 MONGODB TIER | **25** |
|  | 4.6 ADVANTAGES | **26** |
| **5.** | **SYSTEM SPECIFICATION** |  |
|  | 5.1 HARDWARE REQUIREMENTS | **27** |
|  | 5.2 SOFTWARE REQUIREMENTS | **27** |
|  | 5.3 PROJECT REQUIREMENTS | **28** |
|  | 5.3.1 FUNCTIONAL REQUIREMENTS | **28** |

|  |  |  |
| --- | --- | --- |
|  | 5.3.2 PERFORMANCE REQUIREMENTS | **28** |
|  | 5.3.3 INTERFACE REQUIREMENTS | **28** |
|  | 5.3.4 OPERATIONAL REQUIREMENTS | **29** |
|  | 5.3.5 SECURITY REQUIREMENTS | **29** |
| **6.** | **IMPLEMENTATION AND RESULTS** | **30** |
|  | 6.1 MOTIVATION | **30** |
|  | 6.2 SERVER-SIDE IMPLEMENTATION | **31** |
|  | 6.3 CLIENT-SIDE IMPLEMENTATION | **34** |
| **7.** | **PERFORMANCE EVALUATION** | **37** |
|  | 7.1 SYSTEM TEST | **37** |
|  | 7.2WHITE BOX TESTING | **37** |
|  | 7.3 BLACK BOX TESTING | **37** |
|  | 7.4 UNIT TESTING | **38** |
|  | 7.5 INTEGRATION TESTING | **38** |
|  | 7.6 ACCEPTANCE TESTING | **39** |
| **8.** | **CONCLUSION AND FUTURE SCOPE** | **40** |
|  | **APPENDIX I** | **41** |
|  | **APPENDIX II** | **62** |

# ABSTRACT

The term "Multi-model Database" refers to a database that can store, index, and query data from various models. Relational databases, document-oriented databases, graph databases, and triple stores have all typically supported only one paradigm in databases. A database with numerous models is called a multi-model database. our proposed project, which can handle with ecommerce site multi-model management. Also, use the b-crypt technique to encrypt and decrypt messages inside the server to safeguard all of your data from outsiders. To eliminate server problems during the millions of transactions performed on the ecommerce site, we can use the async parallel technique and the water-fall method to complete the transaction in our network. This proposed project gives a secure web interface for purchasing items with a JWT token. JWT token for login and registration, Product ID to map the actual products from the database are some of the features of our suggested solution. Search for the specific product category of the products to be matched using an image search. After that, the voice assistant will suggest product IDs as well as positive and negative reviews as future implementations.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **NO** | **TITLE** | **PAGE** |
| **1.1a** | **BCRYPT ARCHITECTURE** | **1** |
| **1.1b** | **JSON WEB TOKEN** | **2** |
| **2.1** | **CREDIT CARD TRANSACTIONS** | **5** |
| **2.2** | **RECOMMENDATION THE PRODUCTS AS FEATURE WISE** | **8** |
| **2.3** | **MANGODB ROLE IN BIG DATA** | **12** |
| **2.4a** | **JWT TOKEN OVERVIEW – FEDERATED** | **15** |
| **2.4b** | **JWT TOKEN REFRESH AND ACCESS TOKEN** | **15** |
| **2.5** | **DIAGRAM OF MODEL, VIEW AND CONTROLLER OF THE APPLICATION** | **17** |
| **2.6** | **FLOW CHART OF MAPPING OF MANAGEMENT** | **18** |
| **2.7** | **FLOW CHART OF TASKS FOR EACH STUDY PARTICIPANT** | **19** |
| **4.1** | **ARCHITECTURE OF MERN STACK(A)** | **23** |
| **4.1** |  | **25** |
|  | **ARCHITECTURE OF MERN STACK(B)** |  |
| **6.1** | **ARCHITECTURE OF ECOMMERCE FULL STACK** | **29** |
| **6.2** | **PROJECT DIRECTORY FOR NODE SERVER** | **31** |
| **6.3** | **PROJECT DIRECTORY FOR ADMIN PANEL** | **32** |
| **6.4** | **PROJECT DIRECTORY FOR SITE** | **32** |
| **6.5** | **REDUX ARCHITECTURE** | **34** |

|  |  |  |
| --- | --- | --- |
| **9.1** | **ADMIN LOGIN** | **58** |
| **9.2** | **ADMIN DASHBOARD** | **59** |
| **9.3** | **ADMIN NEWSLETTER MANAGEMENT** | **59** |
| **9.4** | **SITE LOGIN** | **60** |
| **9.5** | **SITE REGISTER** | **60** |
| **9.6** | **SITE HOMEPAGE** | **61** |
| **9.7** | **SITE NEWSLETTER** | **61** |

# LIST OF ABBREVATIONS

MERN - MongoDB, Express, React, Node API - Application Programming Interface URI - Uniform Resource Identification URL - Uniform Resource Location

NPM - Node Package Manager JWT - JSON Web Token

JSON - Java Script Object Notation JS - Java Script

CSS - Cascading Style Sheet

HTML - Hyper Text Markup Language

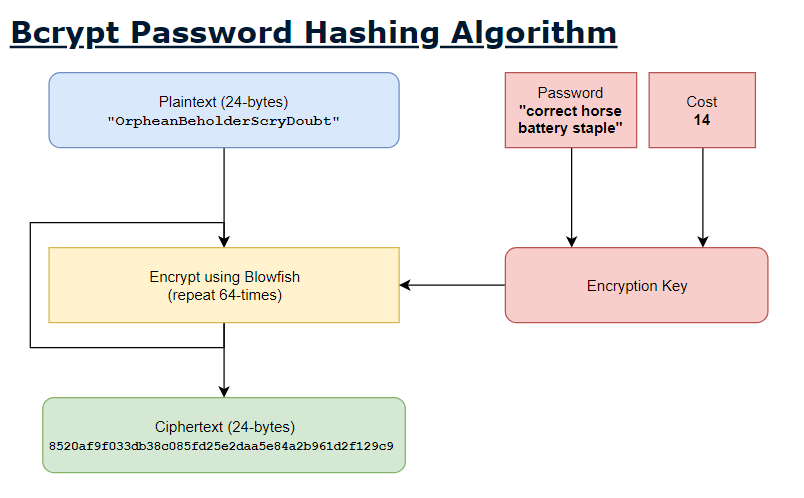
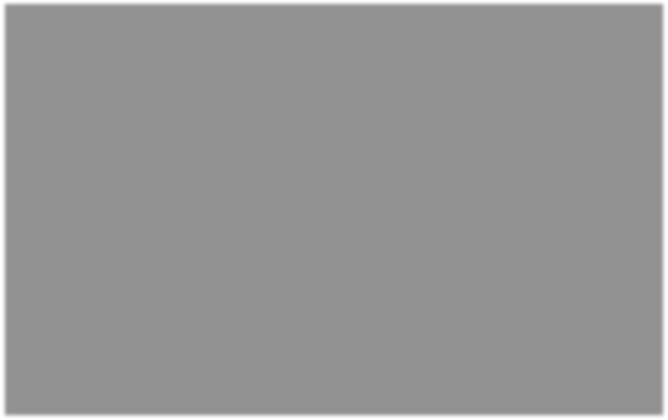
# CHAPTER 1

1. **INTRODUCTION**

# OVERVIEW OF THE PROJECT

Our researched project gives information about the multi-model feature in order to build communication across different models in order to transfer and receive data in a secure manner. The relational data model gained popularity after Edgar F. Codd published it in 1970. After 2009, NoSQL databases became popular due to growing need for horizontal scalability and fault tolerance. Document, graph, and key–value models are among the most prevalent data models in NoSQL databases. The concept of multi-model databases can be traced back to Object–Relational Data Management Systems (ORDBMS) in the early 1990s, and to federated and integrated DBMSs in the early 1980s in a broader sense. By injecting domain specific data types, functions, and index implementations into the DBMS kernels, an ORDBMS system maintains diverse forms of data such as relational, object, text, and spatial data. As defined by Martin Fowler, a multi-model database is a direct answer to the "polyglot persistence" strategy of knitting together various database products, each handling a distinct model, to obtain multi-model capability. The Blowfish Cypher is the basis for Bcrypt, a hashing technique. It takes a plain text password as input and outputs a hash of that password (along with some additional information that we'll discuss later) as a string. Is this a secure connection? No. Encryption is a two-way function, which means that the key can be used to decrypt something. As a result, if an attacker gets the correct key, they can decrypt the encrypted password into plaintext. Hashing, on the other hand, is a one-way function, which means that even if someone knows the key, they won't be able to obtain the plaintext password. As a result, an attacker is forced to undertake brute force assaults to guess the password. We use bcrypt in our suggested project to communicate in an encrypted manner to protect

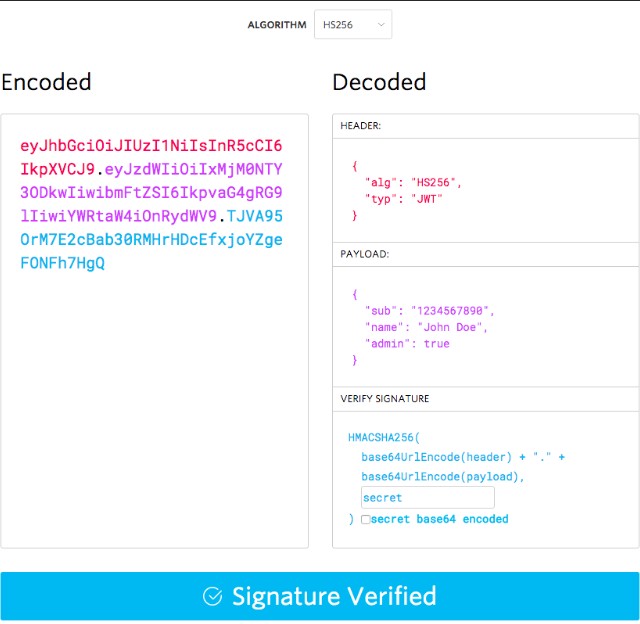
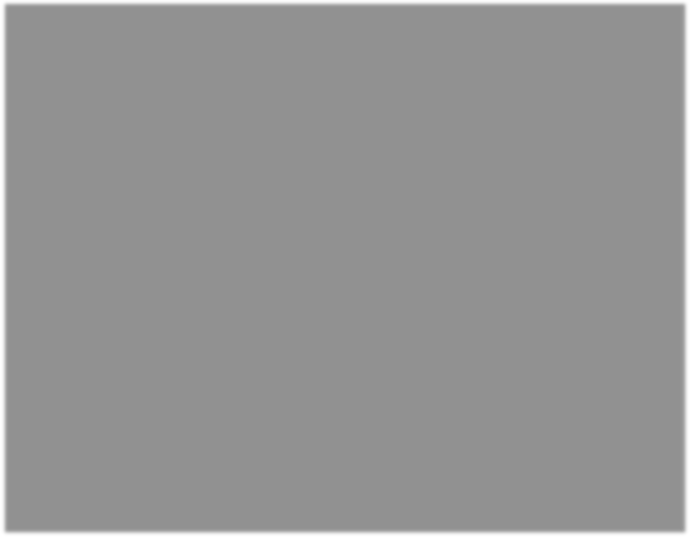
against cyber-attacks outside the environment. There! We now have a rudimentary understanding of how Bcrypt keeps passwords secure. It's one-way, computationally expensive, and each password gets a random salt before hashing.



# [FIGURE 1.1(A) BCRYPT ARCHITECTURE]

Before delving into the structure and development of a JWT, let's have a look at a few real-world examples. This project will give you an idea of the complexity (or simplicity) of some of the most popular JWT-based solutions in use today. For your convenience, all code is available from public repositories. Please keep in mind that the following demos are not intended for usage in production. For production- ready programming, test cases, logging, and security best standards are all required. Because these examples are solely for teaching reasons, they are kept short and to the point. The JSON web token is used to create a secure connection when a user accesses our website for purposes such as login, register, or forget password. The

server generates a JSON web token containing the header, payload, and signature verification while reaching the server in a secure manner. When transmitting data from client to server, it is more dependable and scalable.



# [FIGURE 1.1(B) JSON WEB TOKEN]

JSON web token that provides information about the user who logged in and registered, allowing data to be securely sent between the client and server sides. Between user communication and payload production, it adds a header to ensure that the signature is always checked and validated. These are depicted in the diagram below figure 1.1(b) JSON web token.

The next phase will be to develop a recommendation system that can provide guidance on how to choose products based on positive and negative product reviews. We'll also use product IDs to implement image search and product identification. These can be used to dynamically obtain data and information in order to run the server and client in async mode and use the waterfall approach to perform operations on it.

# PROBLEM STATEMENT

People nowadays are interested in purchasing products from an internet retailer. Books, technological and electrical devices, clothing, furniture, and medicine are all items that can be purchased via an online store. Flipkart and Amazon now play a critical role in the online shop environment, providing 10 crores+ products that may be delivered worldwide. Lenskart uses AI and ChatBot to get customers to buy and demo products from the comfort of their own homes. Our program offers the same functionality, but it also includes a Voice Assistant, a ChatBot, a Product Recommendation System, Secured Login authentication using JWT token and products that can be found using QR codes and their product IDs to map the unique products. One of the most significant difficulties in eCommerce is security breaches. When it comes to eCommerce, there is a lot of information/data involved, and a data-related technical issue can severely harm a retailer's everyday operations as well as brand image. Startups in the ecommerce industry are failing to invest. The first reason is straightforward: ecommerce failure is caused by a lack of investment in reliable website hosting, competent design, and effective marketing. To receive the best ROI in the beginning, you'll need a strategy for where to invest. Various packages such as bcrypt, JWT token, redux, reducer, and action, react- router-dom, nodemon, mongoose, express, node, create-react-app, material-ui, and

bootstrap components overcome these issues to build and construct your website in a terrific and genuine manner.

# 1.3. EXPLANATION OF MULTIMODEL

Multi-model website refers to databases that combine different types of database models into one integrated database engine. They provide a single back end and support multiple data models depending on the applications they support. Such databases can accommodate various data models including relational, object- oriented, key-value, wide-column, document, and graph models. Mark Logic, Arango DB, Orient DB, Azure Cosmos DB, Foundation DB, Couchbase, Apache Ignite are some of the Top Multi-Model Databases.

# 1.4 MERN STACK OVERVIEW:

Although any of these stacks can be used to create complete stack applications, the MERN stack is the most popular among developers. The stack is more user-friendly and well-integrated than others. MERN's Use Cases MERN, like any other web stack, lets you build anything you want, although it's perfect for JSON-heavy, cloud- native, and dynamic web interfaces. Listed below are a few examples: Controlling the flow of work. The practice of obtaining information from numerous sources is referred to as news aggregation. After the four major technologies that make up the stack, MERN stands for MongoDB, Express, React, and Node. MongoDB is a document database system. Node.js web framework Express.js. React.js is a JavaScript client-side framework. The most popular JavaScript web server is Node.js.

The middle (application) tier is made up of Express and Node. Node.js is a popular and capable JavaScript server platform, while Express.js is a server-side web framework. Regardless of which form you select, ME(RVA)N is the best way to

work with JavaScript and JSON from start to finish. The MERN design makes it simple to build a three-tier architecture (frontend, backend, and database) using only JavaScript and JSON.

# CHAPTER 2

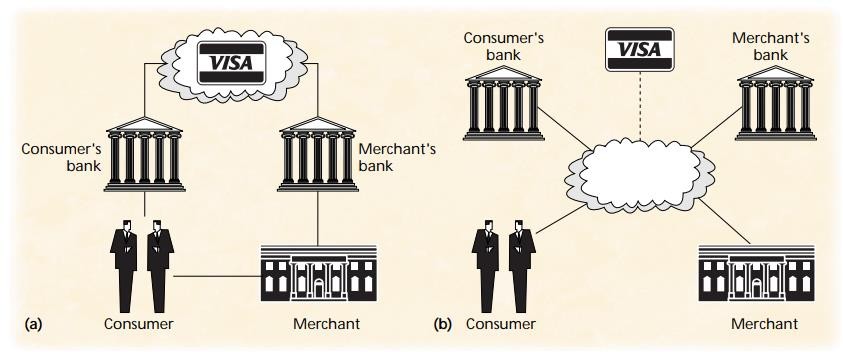
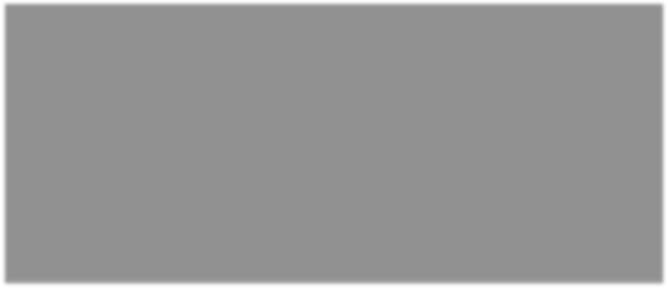
1. **LITERATURE REVIEW**

# A REVIEW OF USABILITY AND SECURITY EVALUATION MODEL OF ECOMMERCE WEB

Websites for electronic trade (E-commerce) have exploded in popularity in recent years. However, because of security and usability concerns, Only, 29% of users make a purchase after conducting an online search. The purpose of this project is to provide an outline of the advantages. ln terms of usability and security factors for an e-commerce website, as well as the shortcomings of existing evaluation techniques. The assessment models that have been used in E-commerce from 2000 to 2018 were examined in this study. The project looked at the usability and security criteria examined by each model were stated for each of the 11 models. Each model's strength and weakness were examined. According to the findings of this project, there is a lack of a single comprehensive model capable of measuring all aspects of usability.[1]

# E-COMMERCE FOR THE 21ST CENTURY

In 1996, more than half of all Internet users in the United States bought something online, with sales totaling $500 million. Despite being a watershed moment, these estimates are grossly overstated, as they do not include people who examined the Internet before making a purchase offline. Pete Ellis' Auto-By-Tel, illustrated in Figure 1, sells $2 billion a year by allowing clients to consult a dealer affiliate database, choose a car, and negotiate a bargain all over the Internet. Payment is made offline, so it isn't counted in the statistics on online purchases. [2]



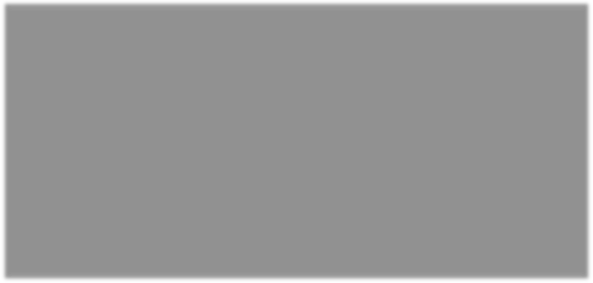
**FIGURE 2.1. CREDIT CARD TRANSACTIONS: (A) TRADITIONAL TRUST MODEL FEATURES A CREDIT PROVIDER AS AN INTERMEDIARY BETWEEN THE MERCHANT’S AND CUSTOMER’S BANKS. (B) THE INTERNET**

# OPINION OBSERVER: RECOMMENDATION SYSTEM ON ECOMMERCE WEBSITE

A variety of recommendation strategies have been developed to help internet customers cope with their product overload. A growing number of E-commerce sites are using recommender systems to assist customers in finding things to buy. The majority of existing systems make recommendations based on the best-selling product, the consumer's demographics, or an analysis of the consumer's previous purchasing activity. Our method was created with the help of consumer reviews and a sophisticated multi-criteria search engine. We employed a text mining approach to extract product attributes, views, and semantic similarity from Web opinion sources in this work. In the eyes of existing customers, the consumer can easily identify the merits and weaknesses of each product. [3]

# INTEGRATED APPROACH TO E-COMMERCE WEBSITES EVALUATION WITH THE USE OF SURVEYS AND EYE TRACKING BASED EXPERIMENTS

The majority of consumers in an online environment are goal-oriented. So, when people go shopping, they have some fundamental notions in mind, such as their needs, desired features, brands, and budget. Consumers can use a multi-criteria search engine to select the desired option, and the system will filter the query and display the shortlisted products. All of these tasks are handled by the E-commerce web interface. Our technology retrieves the desired products via a web interface and gathers user feedback from the merchant's website (Amazon, flip kart, snap deal, eBay etc) [4]



**FIGURE 2.2 RECOMMENDATION THE PRODUCTS AS FEATURE WISE**

# LEARNING TO DETECT AND MEASURE FAKE ECOMMERCE WEBSITES IN SEARCH-ENGINE RESULTS

With so many e-commerce websites offering similar services and products, website accessibility has become one of the most important factors in online business success. As a result, assessing a website's quality and user experience is a significant research undertaking. There are several options available for completing the evaluation, there is a drawback in that retrieving data from the server end is difficult to collect and decode the communication; however, our proposed project's Bcrypt implementation in our project, which allows for continuous communication between client and server, solves this problem. It is highly possible that when searching for a brand name in search engines, you will come across websites that sell counterfeit products. In this work, we look at how to automatically solve and measure this problem.[5]

Our solution comprises of a two-stage learning pipeline. We recognize ecommerce websites (including shop bots) in the search results list and then distinguish between authentic and fraudulent ecommerce websites. We identify appropriate learning features for each stage and demonstrate that this strategy is possible, quick, and highly effective using a prototype system called RI.SI.CO. experimenting with one type of product. [5]

The key to making our suggested React SEO-friendly is to ensure that Google doesn't have to render the content using JavaScript. This can be achieved by utilizing server-side rendering (SSR). It signifies that the files will be provided to the user before the JavaScript code is executed. [5]

# EVALUATING ECOMMERCE WEBSITES COGNITIVE EFFICIENCY

This research uses Data Envelopment Analysis to propose an integrated framework for evaluating ecommerce website efficiency from the user's perspective (DEA). This paradigm is based on ideas from information processing and cognition theories, and it uses website efficiency as a metric of quality and performance. Users engage in a cognitive effort when they interact with website interfaces to complete a task, incurring a cognitive cost to seek, analyze, and process information, and experience either satisfaction or dissatisfaction as a result. This provides the system's performance and efficiency, which is assessed during the transmit and retrieve of data using this assessment perspective.[6]

# THE RESEARCH ON E-COMMERCE WEBSITE SUCCESS MODE

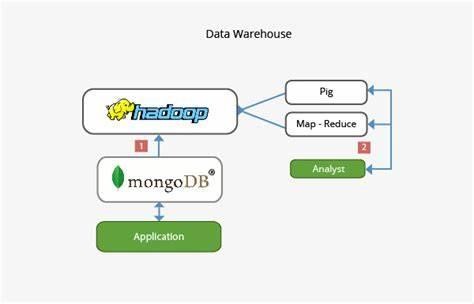
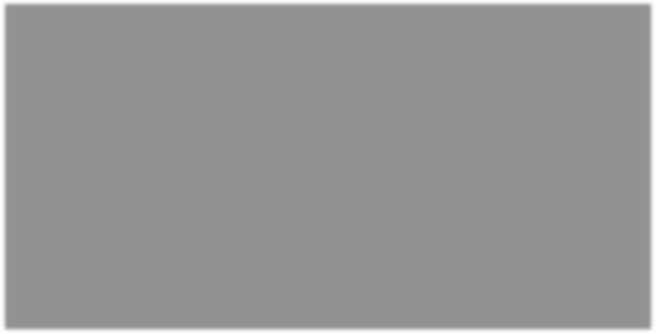
B2B E-business websites are becoming extremely competitive and the number of people that visit a website becomes crucial, an element that contributes to the company's profitability Design, information. The four dimensions of the ecommerce website success model are assurance, communication, and trust. This paper [7] contains the following information: An adaption of Delon and McLean's IS was put to the test. In the context of e-commerce, a success model. We looked into it. The most important elements of an e-commerce website Exploratory, the techniques of factor analysis and confirmatory factor analysis are used. 329 users of ecommerce websites were polled for data, which was gathered through a questionnaire. Our proposed project, based on [7], provides an efficient evaluation and optimum performance in the execution of this server in a production setting. It also provides authentic and SEO-friendly services when conducting business on it.[7]

# DEVELOPMENT OF A DATA ACQUISITION SYSTEM FOR AN ECOMMERCE WEBSITE

The project's objectives are to create a prototype and improve the data collection system based on the current production system. Because the existing system is not automated, processing customer orders is a time-consuming and laborious operation. Whenever orders are placed electronically through the website, and the information is sent to a third party. A third-party credit card processing firm Customer or product order data is missing, as well as the customer's name. The name, address, things requested, and method of payment are all electronically recorded. The website of First Stop Sports.[8]

# RESEARCH ON THE EVALUATION OF E-COMMERCE WEBSITE UNDER THE ENVIRONMENT OF BIG DATA

The function and content of a website, as well as its credit, customer service and enterprise strength, website security, and interface, are all assessed in general. Technology and website design These studies are primarily based on a standard website evaluation method. The advent of the big data era has brought with it new opportunities as well as challenges. Electronics construction and use are plagued by a slew of issues. e-commerce platform in this paper [9], the evaluation index is investigated. Introduce the evaluation approach for an e-commerce website, covering the architecture of the website with the help of big data. In our proposed project gives the database like big data operations were performed in the database to convey the efficient and accurate the classification of the result in MongoDB.



**FIGURE 2.3 MANGODB ROLE IN BIG DATA**

# RESEARCH ON E-COMMERCE BUSINESS MODEL BASED ON INTELLIGENT TERMINALS MOBILE

E-commerce is a new commodity trading pattern that is inextricably linked to telecommunications and computers. With the advancement of both telecommunications and computer technologies, the e-commerce industry has grown quickly. This [10] paper provides a quick overview of the E-commerce business model, which has become a very popular technology for startups, as well as a list of various business benefits and market applications.[10] Our ecommerce website may work in a variety of industries, such as banking and financial services, to change data between the server and the client.[10]

# DESIGNING E-COMMERCE USER INTERFACE

It's not easy to create e-commerce user interfaces. As a result, developers require rules for creating a good user interface. The goal of this paper is to update existing standards and criteria for building a good interface and apply them to the creation of e-commerce websites. An early analysis, consisting of an examination of two ecommerce websites, has been completed. A number of usability concerns were uncovered throughout the examination. Shopping2u.com, a prototype of an e- commerce website, has been created. The prototype will be designed using the guidelines that have been examined and the results of the preliminary analysis. PHP, CSS, JavaScript, and the MySQL database were used to create MyShop2u. The web navigation flow and snapshot of Shopping2u are shown in the concluding section of this article. Furthermore, the proposed system has entire database system is segregated from the rest of the system.

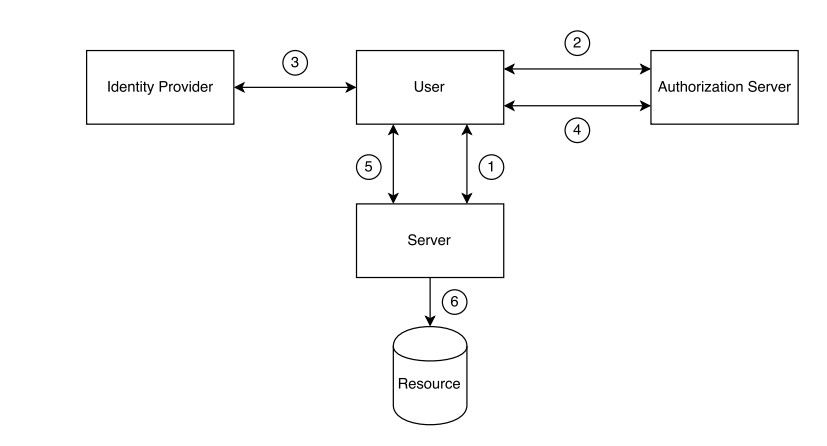
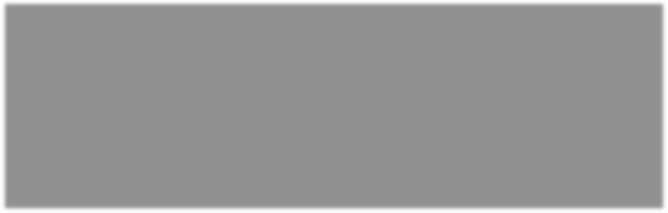
The REST API is used across the system, including the front-end, back-end, and database, and serves a middleware' that can easily be reused for other applications such as mobile software. The MERN stack is a set of technologies that makes application development go faster. It is used by developers all over the world. The fundamental goal of the MERN stack is to create apps that solely use JavaScript. This is due to the fact that the technological stack's four components are all JS-based. The MERN stack is the world's fastest rising tech stack for a variety of reasons. Here are a few examples: Quick Coding Time: For most projects, the MERN stack has one of the quickest coding times, and once you grasp it, you can construct a basic application in a short amount of time without any problem. MERN Stack is a Java script stack that makes full-stack web application deployment easier and faster. MongoDB, Express, React, and Node.js are the four technologies that make up the

MERN Stack. It's intended to make the development process go more smoothly and efficiently.

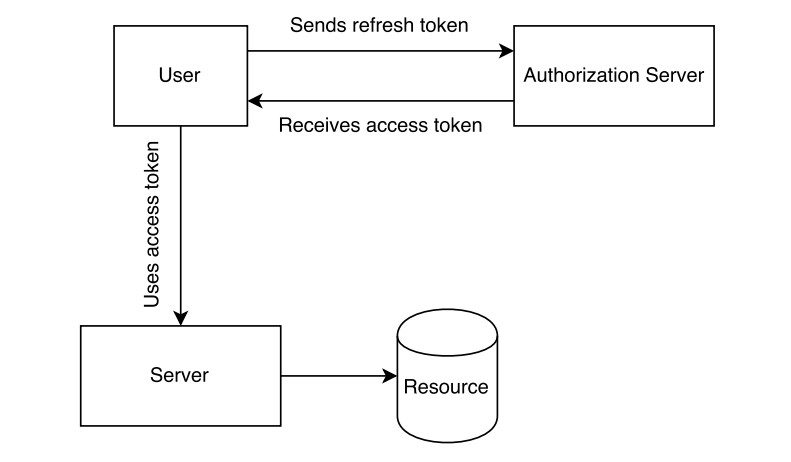
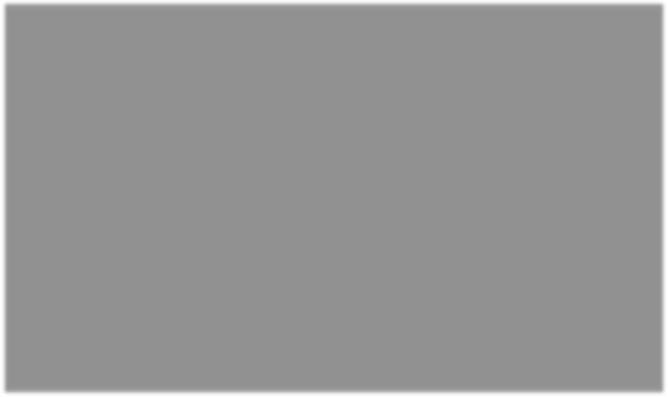
MERN is a full-stack JavaScript framework that is perfect for creating dynamic webpages and applications. It's a free and open-source stack meant to give developers a simple and orderly way to prototype MERN-based web apps quickly. The following is an example of a JSON Web Token (newlines have been added for readability): eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.\seyJzdWIiOiIxMjM0NTY3ODkwIiib mFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.\sTJVA95OrM7E2cBab30R MHrHDcEfxjoYZgeFONFh7HgQ

While this may appear to be nonsense, it is actually a highly compact, printed depiction of a sequence of claims, complete with a signature to ensure its authenticity. [17]

Website performance measurements have long helped organizations enhance their plans and operations. There is a lack of awareness about how performance is monitored for an eCommerce website. Over the last decade, different methods and settings for measuring the performance of a website have been proposed. The owner perspective was employed in this work to theoretically build and practically test a website performance measuring model. The findings point to a second-order factor model for performance. Usage, financial returns, and owner satisfaction are the model's first order factors. The measurements that result serve as a tool for benchmarking the website's performance as well as a foundation for operationalizing the website performance design. [18]



**FIGURE 2.4(a) JWT TOKEN OVERVIEW – FEDERATED**



**FIGURE 2.4(b) JWT TOKEN REFRESH AND ACCESS TOKEN**

The ability to make access tokens easy to validate is a fundamental feature of the separation between access and refresh tokens. The resource server may validate

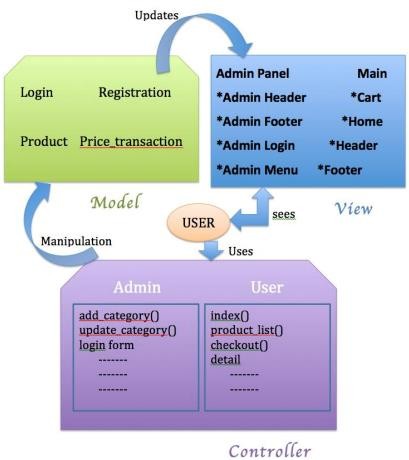
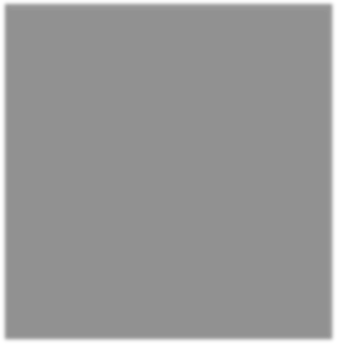
an access token with a signature (such as a signed JWT) on its own. For this purpose, there is no need to contact the authorization server.

On the other hand, refresh tokens necessitate access to the authorization server. Better latency and fewer complex access patterns are attainable by keeping validation separate from queries to the authorization server. Making access tokens as short-lived as possible and incorporating additional checks (such as client checks) within them provides enough security in the event of token leaks.[17]

Despite the fact that OAuth2 makes no mention of the token format, JWTs are an excellent match for its needs. Signed JWTs are good access tokens because they can encode all of the data needed to differentiate access levels to a resource, have an expiration date, and are signed to prevent validation requests against the authorization server. Several federated identity services issue JWT-format access tokens. Refresh tokens can also be deployed with JWTs. However, there are fewer reasons to employ them for this purpose. Because refresh tokens require access to the authorization server, a basic UUID will usually work, as the token does not need to hold a payload (it may be signed, though). [17]

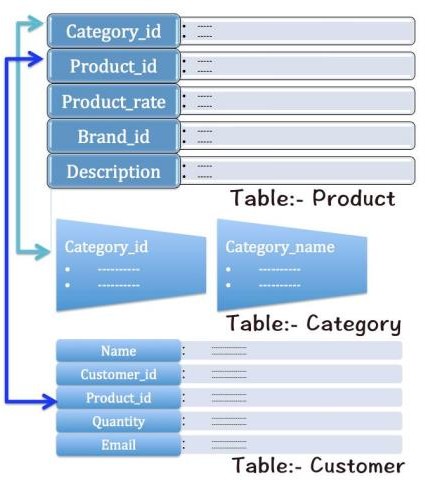
E-commerce is booming at a breakneck pace in this internet age, leaving brick-and-mortar enterprises in the dust. In many cases, brick-and-mortar firms are resorting to having an internet or e-commerce-based alternative. People in the developed world, as well as an increasing number of people in the developing world, now make daily purchases through ecommerce websites. Even Nevertheless, the spread of e-commerce in the developing world is still limited, and there is much to be desired.

This article discusses the various facets of establishing an ecommerce website as well as the best solution to the issues that come with it.



**FIGURE 2.5 DIAGRAM OF MODEL, VIEW AND CONTROLLER OF THE APPLICATION**

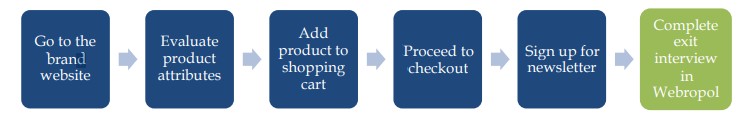
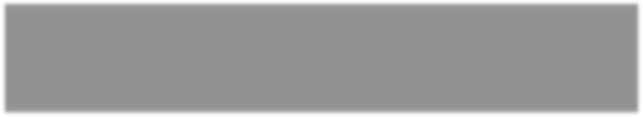
The conventional "Model-View-Controller" pattern was used to create our application. Model view controller (MVC) is a design pattern for creating user interfaces in web applications. It divides the programs into three interconnected components in order to separate internal representations of data from how data is presented to the user for acceptance. As a result, view serves as the user interface. The essential class files for manipulating the data contained in the backend, i.e., the database, are found in the controller. It acts as a conduit between the model and the view. However, it lacks the ability to interface with the database directly.



**FIGURE 2.6 FLOW CHART OF MAPPING OF MANAGEMENT**

The database will contain six tables: promote, brand, category, product, small add, and registration. The figure depicts the construction of three tables. The characteristics of the table "product" are product id, product rate, category id, brand id, and product rate. Where product id is the table's primary key, indicating that each product will have its own identification. The attribute category id serves as a foreign key for this table, forming a link between it and table category. Category id and category name are two attributes. In the event that if a customer registers on our website, his information will be stored on our servers. The attributes name, address, and phone number are maintained in the customer table. Customer id, product id, quantity, and email are all required fields. Again, the unique key for this table is customer id, and the product id is the unique key for this table. A foreign key that connects to the product table.

The following are the six main tasks: visit the brand's website, analyze product attributes, add to basket, checkout, subscribe to the newsletter.



**FIGURE 2.7 FLOW CHART OF TASKS FOR EACH STUDY PARTICIPANT**

Eye tracking is a productive tool in researching the user experience of ecommerce websites. Because information throughout the online path to purchase is communicated visually, gaze behavior is among the most effective and informative means of testing the extent to which a given ecommerce site facilitates a smooth transaction. The process of analysis typically involves examines the characteristics and patterns of visual attention during the online shopping process. Eye-tracking metrics are used in conjunction with data-based visualizations and traditional usability techniques to answer a variety of questions about the online shopping process. Principles of appropriate design, execution and analysis of an ecommerce eye-tracking study are discussed, along with relevant case examples. [16]

# CHAPTER 3

1. **EXISTING SYSTEM**

# SYSTEM ANALYSIS:

In the existing system, we have Amazon and Flipkart, these are the sites which will delivers the product in online stores to customer. In the Flipkart had a recommendation to recommend the products which customer was already visited in their site. Flipkart gives the recommendation based on searching or already visited products. It will increase the productivity and readability of their site to attract the customer to buy their products In Amazon, it provides the various services like It acts as the bank transaction medium. It will have the voice assistant as ALEXA, to search for products. And also have the product recommendation system like Flipkart.

It's not easy to create e-commerce user interfaces. As a result, developers require rules for creating a good user interface. The goal of this paper is to update existing standards and criteria for building a good interface and apply them to the creation of e-commerce websites. An early analysis, consisting of an examination of two ecommerce websites, has been completed. A number of usability concerns were uncovered throughout the examination. Shopping2u.com, a prototype of an e- commerce website, has been created. The prototype will be designed using the guidelines that have been examined and the results of the preliminary analysis. PHP, CSS, JavaScript, and the MySQL database were used to create MyShop2u. The web navigation flow and snapshot of Shopping2u are shown in the concluding section of this article. Furthermore, the proposed system has entire database system is segregated from the rest of the system.

# DISADVANTAGES:

* + - The biggest problem is that it takes at least a day to deliver a product to the customer.
    - Takes more time to complete the orders.
    - It will need more suggestion on product, which should more durable and quality.
    - It should not work for quantity and works for quality.

# CHAPTER 4

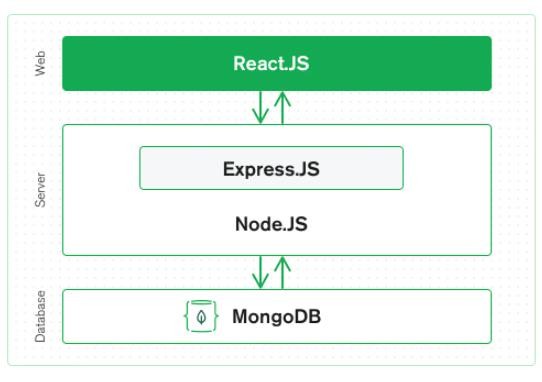
1. **PROPOSED SYSTEM**

# SYSTEM OVERVIEW:

This is the architecture for voice assistant, which will suggest the users wants to looking for the products suggestions to buy. The product suggestion which analysis the product rating and review of the customer has gave for that product. This will map each product by using the respective product ID in the database, The Product ID can be generated by MongoDB. When the user will give the Product ID to our site, our site will give the exact match on our product from the Database.

# ARCHITECTURE:

It's a step-by-step guide for creating a create-read-update-delete (CRUD) online application from the ground up using the MERN (Mongo, Express, React, Node.js) stack, which is one of the most popular Node.js web application stacks. The MEAN and MERN stacks are nearly identical, with the exception that the MEAN stack uses Angular as its front-end framework. UberEATS. UberEATS has been providing solutions aimed at establishing a strong domain. Instagram is one of the food delivery services. MERN Stack is also used by another application that generates a lot of traffic. It becomes critical to have a great user interface and user experience... Walmart. Another major player in the business is influencing the market in a different way. We make it a point to make sure that the solutions. These are a few behemoths who make the most of technology in order to optimize earnings. If you're looking to break into the market.



**FIGURE 4.1 ARCHITECTURE OF MERN STACK(A)**

# REACT.JS FRONT END

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

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

# EXPRESS.JS AND NODE.JS SERVER TIER

The next level down is the Express.js server-side framework, running inside a Node.js server. Express.js bills itself as a “fast, unopinionated, minimalist web framework for Node.js,” and that is indeed exactly what it is. Express.js has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses.

By making XML HTTP Requests (XHRs) or GETs or POSTs from your React.js front-end, you can connect to Express.js functions that power your application. Those functions in turn use MongoDB’s Node.js drivers, either via callbacks for using Promises, to access and update data in your MongoDB database.

# MONGODB DATABASE TIER

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

That’s where MongoDB comes in: JSON documents created in your React.js front end can be sent to the Express.js server, where they can be processed and (assuming they’re valid) stored directly in MongoDB for later retrieval. Again, if you’re building in the cloud, you’ll want to look at Atlas. If you’re looking to set up your own MERN stack, read on!

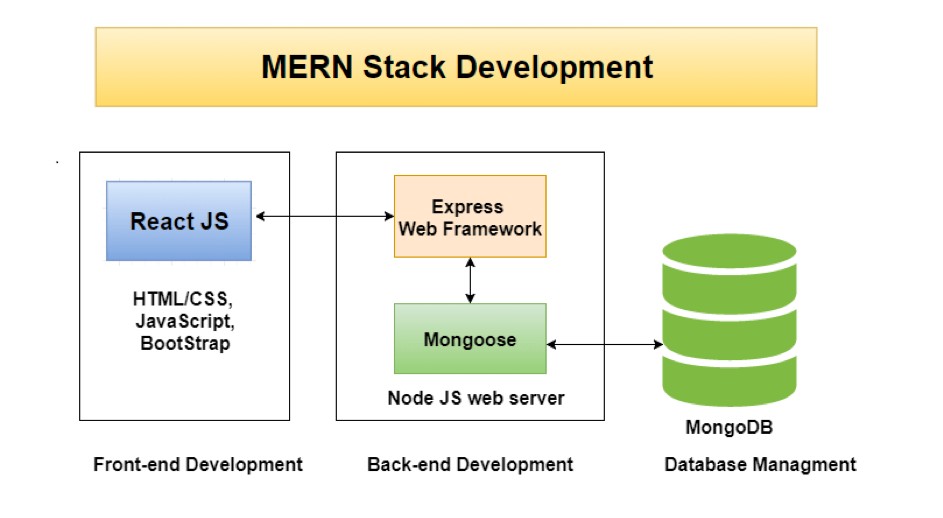
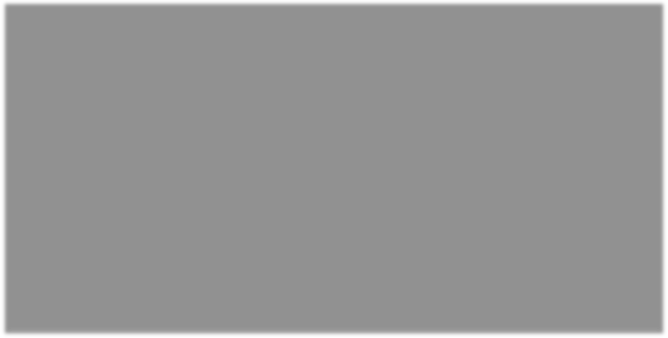
Let’s start with MongoDB, the document database at the root of the MERN stack. MongoDB was designed to store JSON data natively (it technically uses a binary version of JSON called [BSON](https://www.mongodb.com/json-and-bson)), and everything from its command line interface to its query language (MQL, or MongoDB Query Language) is built on JSON and JavaScript.

MongoDB works extremely well with Node.js, and makes storing, manipulating, and representing JSON data at every tier of your application incredibly easy. For cloud-native applications, [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) makes it even easier, by giving you an auto-scaling MongoDB cluster on the cloud provider of your choice, as easy as a few buttons clicks.

Express.js (running on Node.js) and React.js make the JavaScript/JSON application MERN full stack, well, full. Express.js is a server-side application framework that wraps HTTP requests and responses, and makes it easy to map URLs to server-side functions. React.js is a frontend JavaScript framework for building interactive user interfaces in HTML, and communicating with a remote server.

The combination means that JSON data flows naturally from front to back, making it fast to build on and reasonably simple to debug. Plus, you only have to know one programming language, and the JSON document structure, to understand the whole system!

MERN is the stack of choice for today’s web developers looking to move quickly, particularly for those with React.js experience.



# ADVANTAGES:

**FIGURE 4.1 ARCHITECTURE OF MERN STACK(B)**

* + - It will useful for the customer to buy products from remotely.
    - It will give the suggestion based on the positive review on the products
    - It will identify the required product by using the image search to map the image from our site to give the related images in our site
    - It is fully responsive and UI friendly website, which can operate anyone without any prior knowledge in digital skills.
    - It will give more recommendation than the other sites globally.
    - It has QR scan feature to identify the products to looks for our products exact match on required products.

# CHAPTER 5

1. **SYSTEM SPECIFICATION**

They are JavaScript experts who handle front-end operations using HTML, CSS, and JavaScript, and back-end operations with JavaScript and Node. js. Learn how to leverage the MERN stack to create declarative web user interfaces and connect them to backend APIs. Master JavaScript, MongoDB, React, and NodeJS.

# HARDWARE REQUIREMENTS:

Processor: Intel i5/i3

Speed: 2.40 GHz (Recommended) RAM: 8GB (Recommended) Hard Disk: 256 GB

Monitor: 15’’ Inch LCD/LED Input Devices: Keyboard, Mouse

* 1. **SOFTWARE REQUIREMENTS:** Operating System: Windows 10/11 Coding Language: Java Script, web 3.0 IDE: Visual Studio Code

Database Tool: Mongo DB Application Server: Nodemon

# PROJECT REQUIREMENTS:

* + 1. **FUNCTIONAL REQUIREMENTS**

React knowledge is required. It is recommended that you have a basic understanding of React hooks. It is suggested, but not required, that you have a basic understanding of MongoDB, Express, and Node.

# PERFORMANCE REQUIREMENTS

In today's fast-paced business world, it's critical to be able to anticipate and understand client need as efficiently as possible. If our customers could do business with us online and have access to our products or services at their fingertips, it would have a greater impact on their daily lives, creating an ecosystem of doing business online and serving customers on a broad scale.

For the younger generation, shopping and doing business online has become a way of life. People now have access to everything from basic necessities to luxury goods thanks to e-commerce online applications that sell a variety of things.

# INTERFACE REQUIREMENTS

The best user interfaces are practically imperceptible. They stay away from extraneous features and utilize simple language on labels and in communications. Use common UI elements to create uniformity. Users will feel more at ease and be able to complete tasks more quickly if you employ common aspects in your UI. To develop the interactive design in our proposed project, we will use built-in CSS plugins such as material-UI and bootstrap.

# OPERATIONAL REQUIREMENTS

Make sure everything is clear and precise. The less complicated, the better. Ascertain that the user comprehends them. Avoid having unrealistic expectations. In our suggested project, we provide activities such as adding products to our website or performing CRUD operations in the respective management in the respective admin panel.

# SECURITY REQUIREMENTS

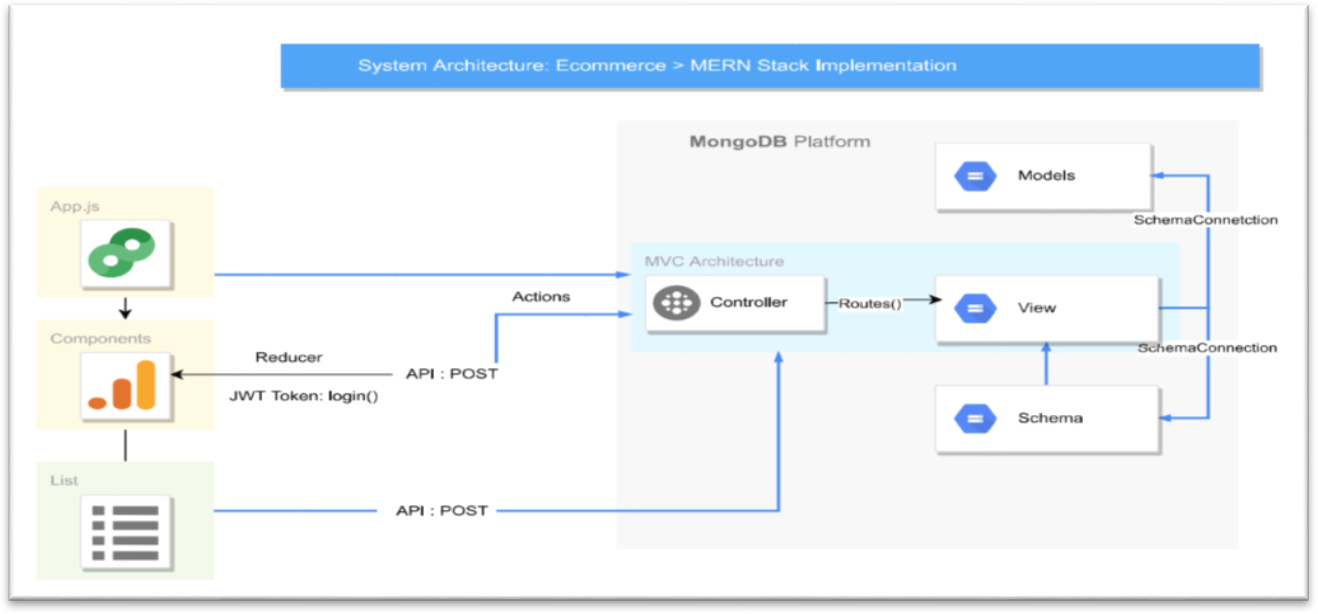
Because the algorithm's mathematical foundations define this as the top bound, passwords employing bcrypt are usually limited to 72 characters. In practice, 56 characters are used in most implementations.

# CHAPTER 6

1. **IMPLEMENTATION AND RESULTS:**

# MOTIVATION:

MERN is one of numerous MEAN stack (MongoDB Express Angular Node) variants in which React.js replaces the usual Angular.js frontend framework. MEVN (MongoDB, Express, Vue, Node) is another variation, and any frontend JavaScript framework can be used. In this part of the proposed project, we'll learn about the MERN stack, which is currently the most popular technology stack. If you're wondering what the difference is between the MEAN and MERN stacks, let me clarify that the only major difference is front-end technology, so you can skip Angular and consider React as front-end technology in this article.



**FIGURE 6.1 ARCHITECTURE OF ECOMMERCE FULL STACK**

# SERVER-SIDE PROGRAM

The config directory can be used to commence communication between the database and the client using the Node server. It's used to connect to MongoDB using localhost and interact with it using URL, hostname, and port. Nodemon is a popular tool for developing node.js-based applications. It just restarts the node application whenever it notices a change in a file in your project's working directory.

Furthermore, nodemon does not require any specific code or development mode changes. By replacing the wrapper for it, it functions as a facilitator in the node. To utilize nodemon, simply substitute the term node on the CLI with the word nodemon before running your script. The controller directory contains the admin and site directories, which are used to communicate with MongoDB and the site.

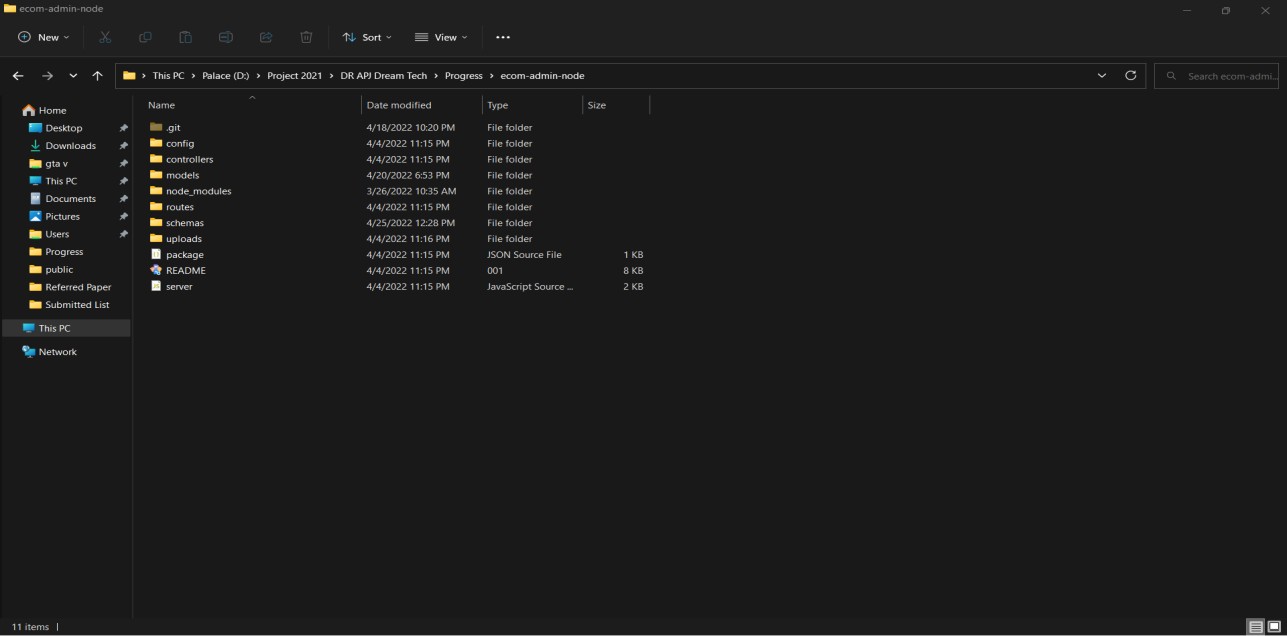
Either through cloning with git or by using npm (the recommended way):

# npm install -g nodemon # or using yarn: yarn global add nodemon

nodemon can be installed as:

# npm install --save-dev nodemon # or using yarn: yarn add nodemon -D

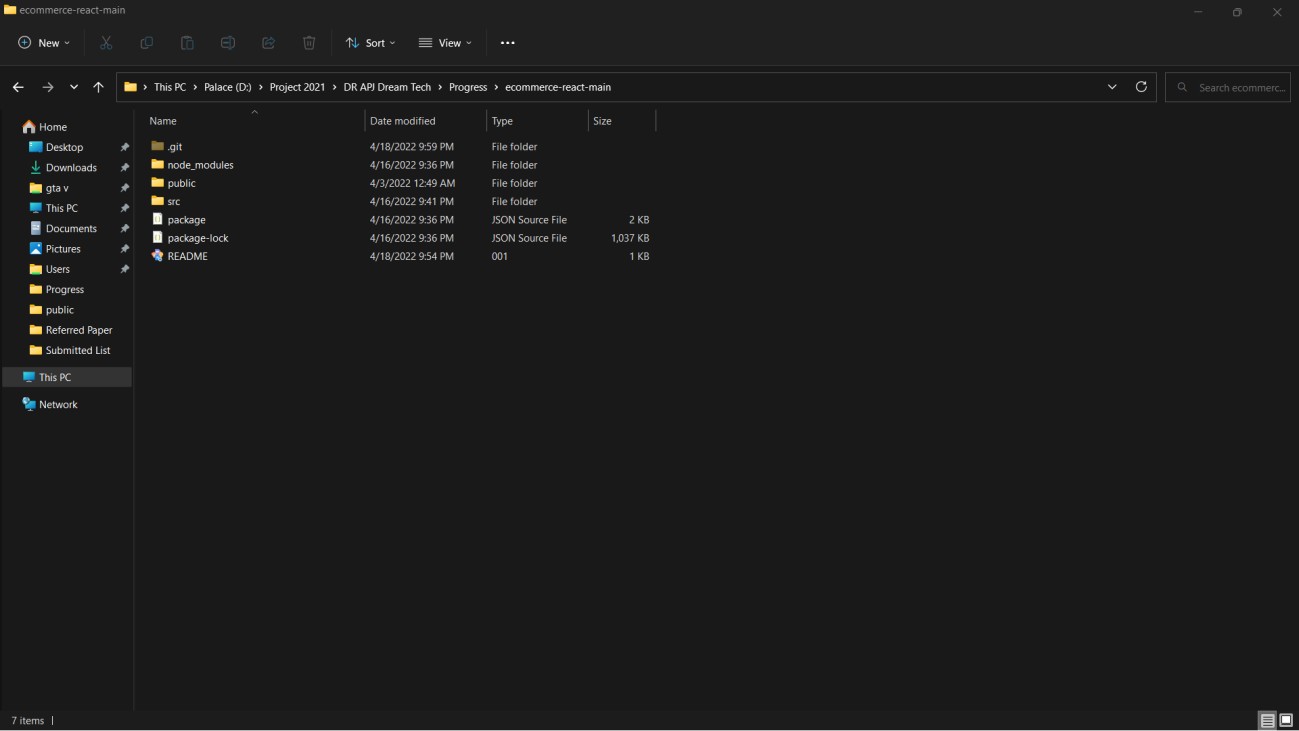
The admin directory houses the management of FAQs, Pages, Configuration, Currency, Language, Category, and Newsletters, as well as the ability to transmit and get data from MongoDB and Clients. The JWT components that communicate with their external components in the site are stored in the site directory. It provides an end-to-end encrypted method of communicating with the server. The schema connection is stored in the model directory, and the mailer is used to communicate with the client and server in an encrypted way. To use Node.js core or NPM modules, you must first import them using the need () function, as demonstrated below. require ('module name'); var module = require ('module\_name'); In the require () function, specify the module name according to the syntax above. To get data from the site, go to the controller directory, which has management such as FAQ, Pages, Configuration, Category, Currency, Language, and Newsletter by running using nodemon command: npx nodemon server.js.



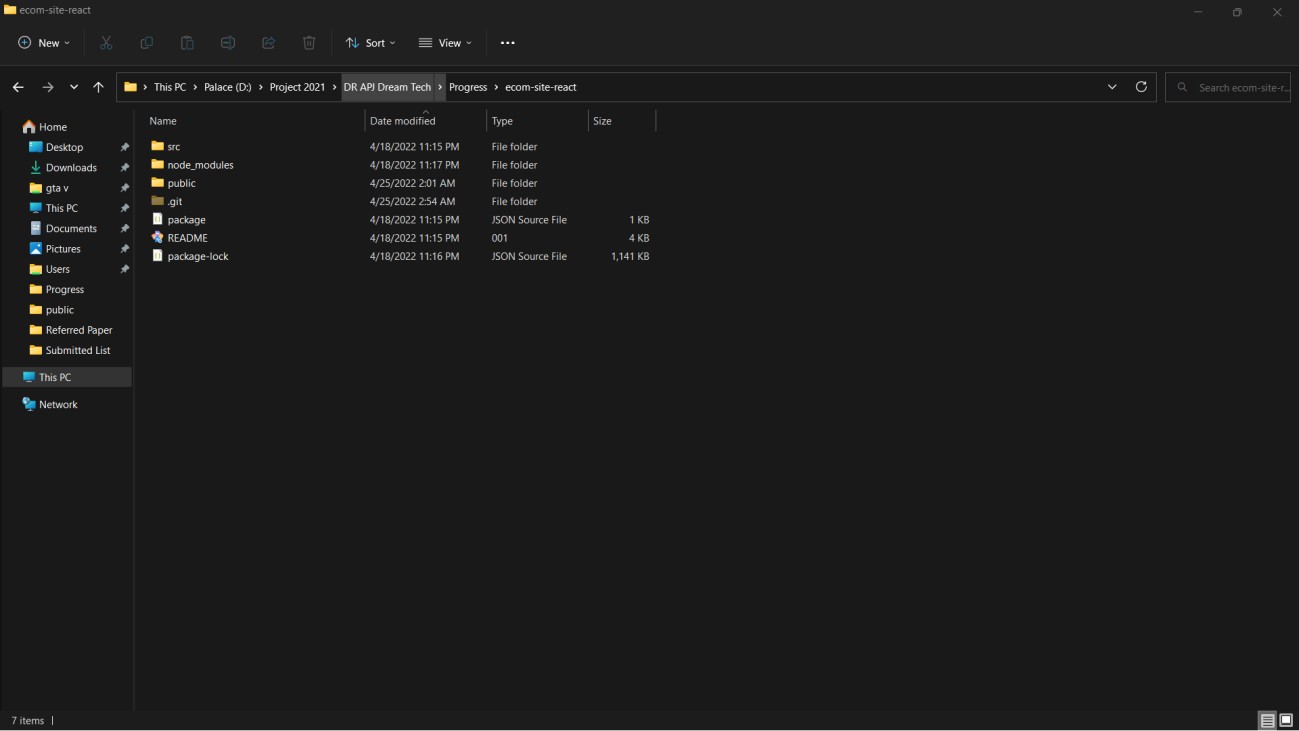
**FIGURE 6.2 PROJECT DIRECTORY FOR NODE SERVER**

The administration of the Schema directory, which contains each management, was done on both the server and the client. FAQ, Pages, Currency, Category, Language, Newsletter, and Configuration in MongoDB, as well as their collections. The structure and contents of your data are defined by a schema, which is a JSON object. Realm's BSON schemas, which extend the JSON Schema standard, can be used to design your app's data model and validate documents as they're produced, modified, or destroyed.

Initiate MongoDB, which can obtain and post connections in order to transport data from the server to the client in a secure manner using bcrypt. It is possible to arrange the nodemon to start from the server.js



**FIGURE 6.3 PROJECT DIRECTORY FOR ADMIN PANEL**



**FIGURE 6.4 PROJECT DIRECTORY FOR SITE**

# CLIENT-SIDE PROGRAM

In the client site, the front-end framework's admin panel is used to obtain a token, which is then kept on local storage and required to continue accessing the server. The token is used to login to the site in an encrypted manner in order to gain access to the server. This directory structure shown in the Figure 9.3 Project directory for Admin Panel. The API connection for each CRUD operation to be executed on the server is located in the action directory.

AC ADD FAQ, AC LIST FAQ, AC VIEW FAQ, AC DELETE FAQ, AC

EDIT FAQ are some examples.

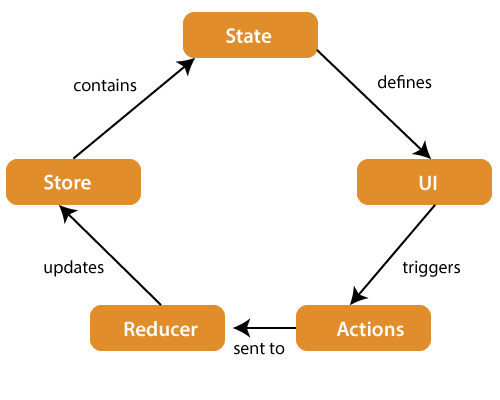
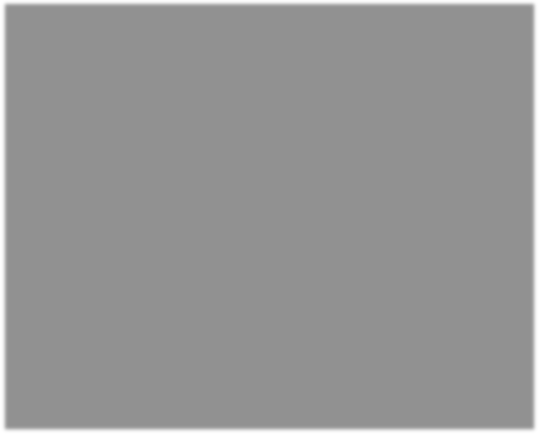
In the login directory, which comprises a list of states to be displayed, such as email, password, and their related error states

setState() schedules an update to a component’s state object. When state changes, the component responds by re-rendering.

props (short for “properties”) and state are both plain JavaScript objects.

While both hold information that influences the output of render, they are different in one important way: props get passed to the component (similar to function parameters) whereas state is managed within the component (similar to variables declared within a function).

More can and should be done with mapStateToProps functions than merely returning state. someSlice. They're in charge of "reshaping" store data to fit the needs of that component. Returning a result as a specific prop name, mixing data from multiple regions of the state tree, and modifying store data in various ways are examples of this.



**FIGURE 6.5 REDUX ARCHITECTURE**

Redux is a JavaScript library for managing application state that is open- source. The user interface in React is built with Redux. Dan Abramov and Andrew Clark were the first to launch it in 2015.

Redux's official React binding is React Redux. It enables React components to read data from a Redux Store and to update data by dispatching Actions to the Store. Redux aids app scaling by providing a logical mechanism to handle state via a one-way data flow architecture. The premise of React Redux is straightforward. It joins the Redux store, checks if the data your component requires has changed, and then re-renders your component.

The official Redux + JS template or Redux + TS template for Create React App, which takes advantage of Redux Toolkit and React Redux's interaction with

React components, is the recommended approach to create new apps with React and Redux.

# Redux + Plain JS template

# npx create-react-app my-app --template redux

# Redux + TypeScript template

# npx create-react-app my-app --template redux-typescript

To use React Redux with your React app, install it as a dependency: # If you use npm:

# npm install react-redux

# Or if you use Yarn:

# yarn add react-redux

Our site's navbar, sidebar, and other user interface components are located in the layout directory.

The title of each component from the react application is dynamically accessed in the util directory, which has the title change dynamically.

This directory can be controlled by the reducer directory, which contains all reducers. The root-reducer.js file in the src folder will manage all exported reducers.

App.css is a file that holds the react site's styling. public directory, which contains index.html for accessing the site's root and returning to index.js. That will contact App.js in order to connect with the other components in the directory.

# CHAPTER 7

1. **PERFORMANCE EVALUATION:**

# SYSTEM TEST:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

# WHITE BOX TESTING:

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

# BLACK BOX TESTING:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

# UNIT TESTING:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

TEST STRATEGY AND APPROACH:

Field testing will be performed manually and functional tests will be written in detail. TEST OBJECTIVES:

* + - All field entries must work properly.
    - Pages must be activated from the identified link.
    - The entry screen, messages and responses must not be delayed. FEATURES TO BE TESTED:
    - Verify that the entries are of the correct format
    - No duplicate entries should be allowed
    - All links should take the user to the correct page.

# INTEGRATION TESTING:

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

# ACCEPTANCE TESTING:

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered

# CHAPTER 8

1. **CONCLUSION AND FUTURE SCOPE:**

# CONCLUSION:

Let us conclude on our project review, there is various e-commerce site is rises exponentially in global. But Project is unique to communicate the customer to buy their products from our site, without any hassle. This may useful for the people who have basic knowledge in the digital skilled person cannot operate on our site without any prior knowledge on digital skills. This is will suggest the people to make a good decision to buy their product from our site. Our main goal is to get trust from the customer to make a brand on our site like global sites. We will implement our project the globalized standards to attract the customer locally and globally.

# FUTURE SCOPE:

For voice assistant, which will suggest the users wants to looking for the products suggestions to buy. The product suggestion which analysis the product rating and review of the customer has gave for that product. This will be the future scope of the project, which is used to connect the audience with feasible solution to the customer with reliable communication between the server and client.

# APPENDIX I CODING AND SNAPSHOTS

* 1. **SERVER-SIDE PROGRAM**

Path: controller/faq.js

const express = require('express'); const Router = express.Router();

const DB = require('../../models/db');

const HELPERFUNC = require('../../models/commonfunctions'); var mongoose = require('mongoose');

const verifyToken= require('./auth/verifyToken') Router.get('/listFaqs',verifyToken,function(req,res) {

const response = { status : 0,

}

DB.GetDocument('faqs',{}, {}, {}, function(err, result) { if(err) {

res.send(response);

} else {

response.status = 1; response.data = result; response.count = result.length; res.send(response);

}

});

});

Router.post('/viewFaq',verifyToken,function(req,res) { const response = {

}

DB.GetOneDocument('faqs',{\_id:req.body.id}, {}, {}, function(err, result) { if(err) {

res.send(response);

} else {

response.question = result.question; response.answer=result.answer; response.status=result.status; response.id=result.id; res.send(response);

}

});

});

Router.post('/addUpdateFaq',verifyToken,function(req,res) { const response = {

status : 0,

message : 'Something went wrong in your code!'

}

req.checkBody('question', 'question is required.').notEmpty(); req.checkBody('answer', 'answer is required.').notEmpty();

var errors = req.validationErrors(); if (errors) {

return res.status(422).json({ errors: errors});

}

const question = req.body.question; const answer = req.body.answer; const status = req.body.status; const faqFormData = {

question : HELPERFUNC.Capitalize(question), answer : HELPERFUNC.Capitalize(answer), status : status

}

if(!req.body.id){

DB.GetOneDocument('faqs', {question : question}, {}, {}, function(err, result) {

if(result){ response.status = 0;

response.message = 'Data you have entered is already exist!'; res.send(response);

} else {

DB.InsertDocument('faqs', faqFormData, function(err, result1) { if(err) {

res.send(response);

} else { response.status = 1;

response.message = 'Faq added successfully'; response.id = result1.\_id; res.send(response);

}

});

}

});

} else {

DB.FindUpdateDocument('faqs',{\_id:req.body.id}, faqFormData, function(err, result) {

if(err) { res.send(response);

} else {

DB.GetOneDocument('faqs', {\_id:req.body.id}, {}, {}, function(err, result1) {

if(err) {

res.send(response);

} else {

const faqData = {

id : result1.\_id, question : result1.question, answer : result1.answer, status : result1.status

}

});

}

})

}

});

}

response.status = 1;

response.message = 'Faq updated successfully'; response.data = faqData;

res.send(response);

Router.post('/deleteFaq',verifyToken,function(req,res) { const response = {

status : 0,

message : 'Something went wrong in your code!'

}

DB.DeleteDocument('faqs', {\_id:req.body.id}, function(err, result) { if(err) {

res.send(response);

} else {

DB.GetDocument('faqs', {}, {}, {}, function(err, result) { if(err) {

res.send(response);

} else {

response.status = 1;

response.message = 'Faq deleted successfully'; response.data = result;

response.count = result.length; res.send(response);

});

})

}

});

}

module.exports = Router;

Path:models/shemmaconnection.js

const mongoose = require('mongoose');

// importing schemas to create model

const importedfaqSchema = require('../schemas/faqschema'); const importedpageSchema = require('../schemas/pageschema');

const importedcountrySchema = require('../schemas/countryschema'); const importedImageSchema = require('../schemas/imageschema'); const importedcurrencySchema = require('../schemas/currencySchema')

const importedlanguageSchema = require('../schemas/languageSchema'); const importednewsletterSchema = require('../schemas/newsletterSchema'); const importedpaymentSchema = require('../schemas/paymentSchema'); const importedconfigSchema = require('../schemas/configschema');

const importedLoginSchema = require('../schemas/loginSchema'); const importeduserSchema = require('../schemas/userschema');

const importedconfigOptionSchema = require('../schemas/configOptionSchema');

// Creating schema

const FaqSchema = mongoose.Schema(importedfaqSchema, { timestamps: true, versionKey: false });

const PageSchema = mongoose.Schema(importedpageSchema, { timestamps: true, versionKey: false });

const CountrySchema = mongoose.Schema(importedcountrySchema, { timestamps: true, versionKey: false });

const ImageSchema = mongoose.Schema(importedImageSchema,{timestamps: true, versionKey: false });

const CurrencySchema = mongoose.Schema(importedcurrencySchema,{timestamps: true, versionKey: false });

const LanguageSchema = mongoose.Schema(importedlanguageSchema,{timestamps: true, versionKey: false });

const NewsletterSchema = mongoose.Schema(importednewsletterSchema,{timestamps: true, versionKey: false });

const PaymentSchema = mongoose.Schema(importedpaymentSchema,{timestamps: true, versionKey: false });

const configSchema = mongoose.Schema(importedconfigSchema,{timestamps: true, versionKey: false });

const LoginSchema = mongoose.Schema(importedLoginSchema,{timestamps: true, versionKey: false });

const userSchema = mongoose.Schema(importeduserSchema, { timestamps: true, versionKey: false });

const configOptionSchema = mongoose.Schema(importedconfigOptionSchema,{timestamps:true,versionKey:false});

// Creating models

const FaqModel = mongoose.model('faqs', FaqSchema); const PageModel = mongoose.model('pages', PageSchema);

const CountryModel = mongoose.model('countries', CountrySchema); const ImageModel = mongoose.model('image',ImageSchema);

const CurrencyModel = mongoose.model('currency',CurrencySchema); const LanguageModel = mongoose.model('languages', LanguageSchema); const NewsletterModel= mongoose.model('newsletter',NewsletterSchema) const PaymentModel= mongoose.model('payments',PaymentSchema)

const ConfigModel= mongoose.model('config',configSchema) const LoginModel= mongoose.model('logins',LoginSchema) const userModel = mongoose.model('user', userSchema);

const ConfigOptionModel=mongoose.model('configOption',configOptionSchema)

module.exports = { faqs: FaqModel, pages: PageModel,

countries: CountryModel, image :ImageModel, currency:CurrencyModel, languages: LanguageModel, newsletter:NewsletterModel, payments:PaymentModel, config:ConfigModel, logins:LoginModel,

user: userModel, configOption:ConfigOptionModel

}

Path:routes/index.js

The base url for each management generated in the admin panel in the async parallel and waterfall methods can be found in the route’s directory.

const BaseUrl = '/api/v1/'; module.exports = function(app) {

app.use(BaseUrl+"faqs", require("../controllers/admin/faq")); app.use(BaseUrl+"pages", require("../controllers/admin/pages")); app.use(BaseUrl+"countries", require("../controllers/admin/country")); app.use(BaseUrl+"category", require("../controllers/admin/category")); app.use(BaseUrl+"currency", require("../controllers/admin/currency")); app.use(BaseUrl+"languages", require("../controllers/admin/language")); app.use(BaseUrl+"newsletters", require("../controllers/admin/newsletter")); app.use(BaseUrl+"payments", require("../controllers/admin/payment"));

app.use(BaseUrl+"config", require("../controllers/admin/config")); app.use(BaseUrl+"login", require("../controllers/admin/login")); app.use(BaseUrl+"auth", require("../controllers/admin/auth/verifyToken")); app.use(BaseUrl+"user", require("../controllers/site/user")); app.use(BaseUrl+"configOption", require("../controllers/admin/configOption"));

}

Path: schema/faqschema.js

var FAQSCHEMA = {

question : String, answer : String,

status : Boolean,

};

module.exports = FAQSCHEMA;

Path: ./server.js

const express = require('express');

const bodyParser = require('body-parser'); const cors = require('cors');

const app = express();

const mongoose = require('mongoose');

const dbConfig = require('./config/db.js');

const expressValidator = require('express-validator'); app.use(expressValidator())

app.use(cors())

const DBURL = dbConfig.url;

const PORT = dbConfig.port; const LOCAL\_ADDRESS = dbConfig.hostname;

// Connecting to the database mongoose.connect(DBURL, {

useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true, useFindAndModify: false

}).then(() => {

console.log("Successfully connected to the database");

}).catch(err => {

console.log('Could not connect to the database. Exiting now...', err);

process.exit();

});

app.use('/uploads', express.static('uploads'));

// parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: false, useUnifiedTopology: true }))

// parse application/json app.use(bodyParser.json());

app.use(cors());

// importing routes files require('./routes/')(app);

app.listen(PORT, LOCAL\_ADDRESS, () => {

console.log(`Server running at http://${LOCAL\_ADDRESS}:${PORT}/`);

});

# CLIENT-SIDE PROGRAM:

Path:./common/authHeaders.js

const token= localStorage.getItem('token'); const config={

headers:{Authorization:`Bearer ${token}`}

}

export default config;

Path:./actions/faq.js

import axios from "axios";

import config from "../../common/authHeaders"; const LIST\_FAQ = 'LIST\_FAQ';

const ADD\_FAQ = 'ADD\_FAQ'; const DELETE\_FAQ='DELETE\_FAQ' const VIEW\_FAQ='VIEW\_FAQ' const UPDATE\_FAQ='UPDATE\_FAQ'

export function AC\_ADD\_FAQ(userData) { return function (dispatch) {

return axios.post("http://localhost:8000/api/v1/faqs/addUpdateFaq", userData,config)

.then(({ data }) => {

dispatch({ type: ADD\_FAQ, payload: data })

});

};

}

export function AC\_LIST\_FAQ() { return function (dispatch) {

return axios.get("http://localhost:8000/api/v1/faqs/listFaqs",config)

.then(({ data }) => {

console.log('=======List Faq========', data) dispatch({ type: LIST\_FAQ, payload: data })

});

}

}

export function AC\_DELETE\_FAQ(formdata){ console.log('===-=-=action -=-=-',formdata) return function(dispatch){

return axios.post("http://localhost:8000/api/v1/faqs/deleteFaq",formdata,config)

.then(({data}) => {

dispatch({type:DELETE\_FAQ,payload:data})

});

};

}

export function AC\_VIEW\_FAQ(formdata){ console.log('===-=-=action -=-=-',formdata) return function(dispatch){

return axios.post("http://localhost:8000/api/v1/faqs/viewFaq",formdata,config)

.then(({data}) => {

dispatch({type:VIEW\_FAQ,payload:data})

});

};

}

export function AC\_HANDLE\_INPUT\_CHANGE(name,value){ return function(dispatch){

dispatch({type:UPDATE\_FAQ, name:name, value:value})

};

}

Path: ./login/login.js

import React from 'react';

import { bindActionCreators } from 'redux'; import { connect } from 'react-redux'; import { AC\_LOGIN } from '../actions/login'; import swal from 'sweetalert';

class Login extends React.Component { constructor(props) {

super(props); this.state = {

loginError: false, email: "", passwordError: false, password: "",

}

this.validation = this.validation.bind(this); this.handleinputchange = this.handleinputchange.bind(this);

}

validation() {

const email=this.state.email; const password=this.state.password if (email) {

if (email.length < 3) {

this.setState({ loginError: false, color0: "red" })

}

else {

this.setState({ loginError: false, color0: "" })

}

}

else {

this.setState({ loginError: true, color0: "red" })

}

if (password) {

if (password.length < 2) {

this.setState({ passwordError: false, color1: "red" })

}

else {

this.setState({ passwordError: false, color1: "" })

}

}

else {

this.setState({ passwordError: true,color1: "red" })

}

if (email && password) { swal("Login Success!", {

buttons: false, timer: 2000, icon:"success"

});

this.setState({ email: '', password: '' });

}else{

swal("Please enter email and password", { buttons: false,

timer: 2000, icon:"error"

});

console.log(this.props.loginreducer.loginInfo)

}

const formData = {

email: this.state.email, password: this.state.password,

}

this.props.AC\_LOGIN(formData); console.log('-=value-=', formData)

}

handleinputchange(event) {

const fieldId = event.target.id; const fieldValue = event.target.value;

if (fieldId === "login") { this.setState({ email: fieldValue }) if (fieldValue) {

if (fieldValue.length < 5) {

this.setState({ loginError: false, color0: 'red' })

}

else {

this.setState({ loginError: false, color0: '' })

}

}

else {

this.setState({ loginError: true, color0: '' })

}

}

if (fieldId === "password") { this.setState({ password: fieldValue }) if (fieldValue) {

if (fieldValue.length < 5) {

this.setState({ passwordError: false, color1: 'red' })

}

else {

this.setState({ passwordError: false, color1: '' })

}

}

else {

this.setState({ passwordError: true, color1: '' })

}

}

}

render() {

// Redirect={} return (

<>

<div class="container-scroller">

<div class="container-fluid page-body-wrapper full-page-wrapper">

<div class="content-wrapper d-flex align-items-center auth px-0">

<div class="row w-100 mx-0">

<div class="col-lg-4 mx-auto">

<div class="auth-form-light text-left py-5 px-4 px-sm-5">

<h6 class="fw-light">Sign in to continue...</h6>

<form class="pt-3">

<div class="form-group">

<div>Enter your email</div>

<input type="text" class="form-control form-control-lg" id="login" onChange={this.handleinputchange} value={this.state.login} placeholder="Email" autoComplete='off'/>

{this.state.loginError ? <label className="mt-2" style={{ color: 'red' }}>Email is required</label> : ""}

</div>

<div class="form-group">

<div>Enter Password</div>

<input type="password" class="form-control form-control- lg" id="password" onChange={this.handleinputchange} value={this.state.password} placeholder="Password" />

{this.state.passwordError ? <label className="mt-2" style={{ color: 'red' }}>Password should be atleast 5 characters</label> : ""}

</div>

<div class="mt-3">

<a class="btn btn-block btn-primary btn-lg font-weight- medium auth-form-btn" onClick={this.validation}>Sign in</a>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

</div>

</>

);

}

}

function mapStateToProps(state) { console.log('map state', state); return {

loginreducer: state.loginReducer

}

}

function mapDispatchToProps(dispatch) {

return bindActionCreators({ AC\_LOGIN }, dispatch)

}

export default connect(mapStateToProps, mapDispatchToProps)(Login)

Path:./reducers/faqred.js

const initialState = { faqList: [],

faqAdd: [],

faqDelete:[], faqInfo : {

question : '',

answer : '',

status : '',

id : ''

}

}

function FAQ\_Reducer(state = initialState, action) { console.log("-=-=-=Reducer=-=-=", action)

switch (action.type) { case 'LIST\_FAQ':

return {

...state,

faqList: action.payload.data

};

case 'ADD\_FAQ': return {

...state,

faqAdd: action.payload

};

case 'DELETE\_FAQ':

return {

...state,

faqDelete: action.payload

};

break;

case 'VIEW\_FAQ': return {

...state,

faqInfo: action.payload

};

case 'EDIT\_FAQ': return {

...state,

editFaq: action.payload

}

break;

case 'UPDATE\_FAQ':

return Object.assign({},state,{ faqInfo : {

...state.faqInfo, [action.name] : action.value

}

})

default: return state;

}

}

export default FAQ\_Reducer;

Path:./util/DyanamicTab.js

export const TabTitle = (newTitle) =>{ return document.title=newTitle;

}

Path:App.js

// For Package Import Statement import React from 'react'; import './App.css';

import { BrowserRouter as Router, Switch, Route } from "react-router-dom";

// For Files Import Statement

import Login from './components/login/login'; import mainboard from './mainboard';

class App extends React.Component { constructor(props) {

super(props); this.state = {

login: false,

}

}

render() {

var token = localStorage.getItem("token") if (!token) {

return (

<Router>

<Switch>

<Route exact path="/" component={Login} />

</Switch>

</Router>

)

}

else {

return (

<Router>

<Switch>

<Route exact path="/" component={mainboard} />

</Switch>

</Router>

);

}

}

}

export default App;

Path:mainboard.js

import React from 'react'; import logo from './logo.svg'; import './App.css';

import { BrowserRouter as Router, Switch, Route } from "react-router-dom"; import Sidebar from './components/layouts/sidebar';

import Navbar from './components/layouts/navbar'

// import Footer from './components/layouts/footer'; import Dashboard from './components/layouts/dashboard';

//Login Component

// import Login from './components/login/login';

// Faq Management

import addFaqs from './components/faq/addFaq'; import ListFaq from './components/faq/listFaq'; import editFaq from './components/faq/editfaq'; import viewFaq from './components/faq/viewFaq';

// Page Management

import addPages from './components/pages/addPages'; import listPages from './components/pages/listPages'; import editPage from './components/pages/editPage'; import viewPage from './components/pages/viewPage';

// Country Management

import addCountry from './components/country/addCountry'; import listCountry from './components/country/listCountry'; import editCountry from './components/country/editCountry'; import viewCountry from './components/country/viewCountry';

// Category Management

import addImage from './components/Category/addImage'; import listImages from './components/Category/listImages'; import editImage from './components/Category/editImage'; import viewImage from './components/Category/viewImage';

import addCurrency from './components/Currency/addCurrency'; import listCurrency from './components/Currency/listCurrency'; import editCurrency from './components/Currency/editCurrency'; import viewCurrency from './components/Currency/viewCurrency';

import Addlanguages from './components/language/addlanguage'; import Listlanguage from './components/language/listlanguage'; import Editlanguage from './components/language/editlanguage'; import Viewlanguage from './components/language/viewlanguage'

import AddNewsletter from './components/newsletter/addNewsletter'; import ListNewsletter from './components/newsletter/listNewsletter'; import editNewsletter from './components/newsletter/editNewsletter'; import viewNewsletter from './components/newsletter/viewNewsletter'

import addPayment from './components/payment/addPayment'; import listPayment from './components/payment/listPayment'; import editPayment from './components/payment/editPayment'; import viewPayment from './components/payment/viewPayment';

import addConfiguration from './components/configuration/addConfiguration'; import listConfiguration from './components/configuration/listConfiguration'; import editConfiguration from './components/configuration/editConfiguration'; import viewConfiguration from './components/configuration/viewConfiguration';

//User management

import listUser from './components/user/listUser'; import viewUser from './components/user/viewUser';

// Dynamic Tab Changes

import {TabTitle} from './components/util/DynamicTab' class mainboard extends React.Component {

constructor(props) { super(props); this.state = {

login:false,

}

}

render(){

TabTitle('Admin Panel'); return (

<Router>

<div class="container-scroller">

<Navbar/>

<div class="container-fluid page-body-wrapper">

<Sidebar/>

<Switch>

<Route exact path="/" component={Dashboard}/>

<Route exact path="/addFaq" component={addFaqs}/>

<Route exact path="/listFaq" component={ListFaq}/>

<Route exact path='/editFaq/:id' component={editFaq}/>

<Route exact path='/viewFaq/:id' component={viewFaq}/>

<Route exact path="/addPage" component={addPages}/>

<Route exact path="/listPage" component={listPages}/>

<Route exact path='/editPage/:id' component={editPage}/>

<Route exact path='/viewPage/:id' component={viewPage}/>

<Route exact path="/addCountry" component={addCountry}/>

<Route exact path="/listCountry" component={listCountry}/>

<Route exact path='/editCountry/:id' component={editCountry}/>

<Route exact path='/viewCountry/:id' component={viewCountry}/>

<Route path="/addImage" component={addImage} />

<Route path="/listImages" component={listImages} />

<Route path="/viewImage/:id" component={viewImage} />

<Route path="/editImage/:id" component={editImage}/>

<Route exact path="/addCurrency" component={addCurrency}/>

<Route exact path="/listCurrency" component={listCurrency}/>

<Route exact path='/editCurrency/:id' component={editCurrency}/>

<Route exact path='/viewCurrency/:id' component={viewCurrency}/>

<Route exact path="/addlanguage" component={Addlanguages}/>

<Route exact path="/listlanguage" component={Listlanguage}/>

<Route exact path='/editlanguage/:id' component={Editlanguage}/>

<Route exact path='/viewlanguage/:id' component={Viewlanguage}/>

<Route exact path="/addNewsletter" component={AddNewsletter}/>

<Route exact path="/listNewsletter" component={ListNewsletter}/>

<Route exact path='/editNewsletter/:id' component={editNewsletter}/>

<Route exact path='/viewNewsletter/:id' component={viewNewsletter}/>

<Route exact path="/addPayment" component={addPayment}/>

<Route exact path="/listPayment" component={listPayment}/>

<Route exact path="/editPayment/:id" component={editPayment}/>

<Route exact path="/viewPayment/:id" component={viewPayment}/>

<Route exact path="/addConfig" component={addConfiguration}/>

<Route exact path="/listConfig" component={listConfiguration}/>

<Route exact path='/editConfig/:id' component={editConfiguration}/>

<Route exact path='/viewConfig/:id' component={viewConfiguration}/>

<Route exact path='/listUser' component={listUser}/>

<Route exact path='/viewUser/:id' component={viewUser}/>

<Route exact path="/" component={Dashboard}/>

</Switch>

</div>

{/\* <Footer/> \*/}

</div>

</Router>

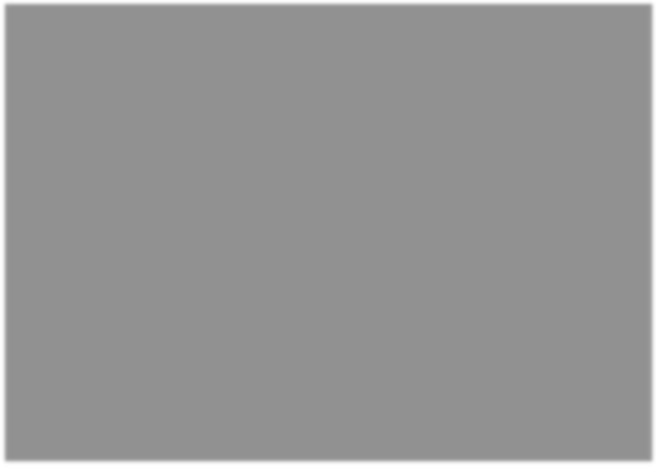
);

}

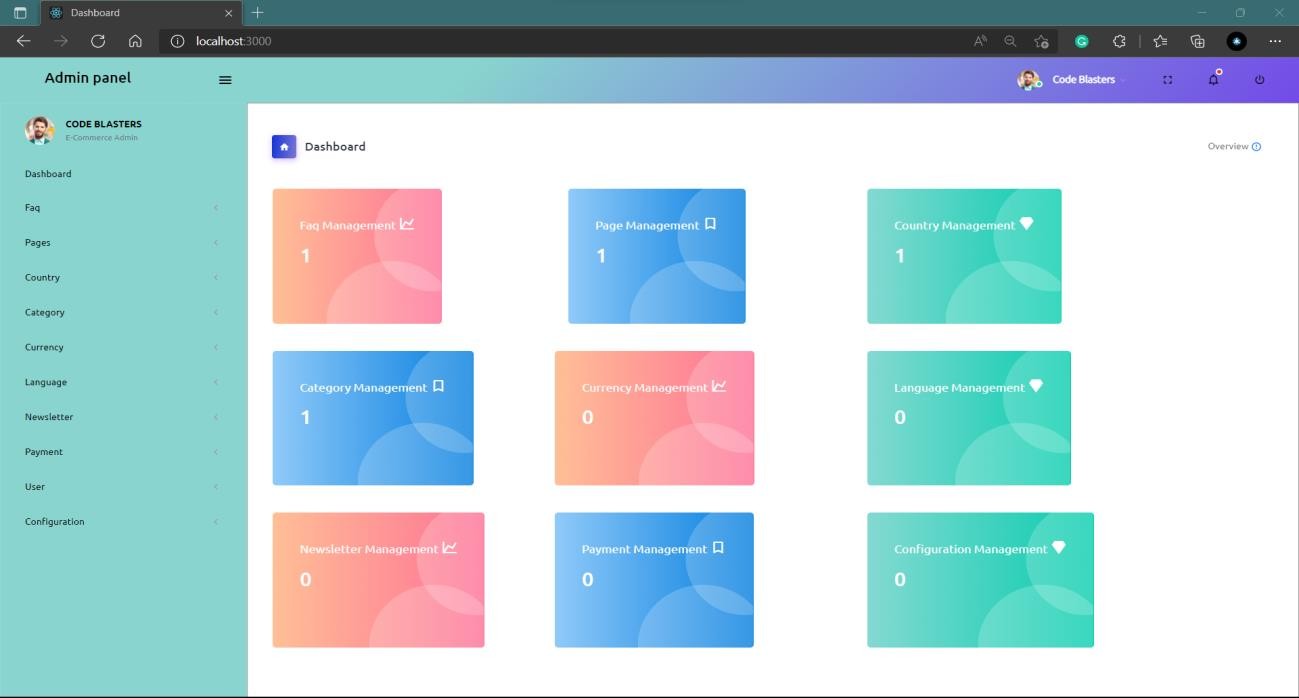
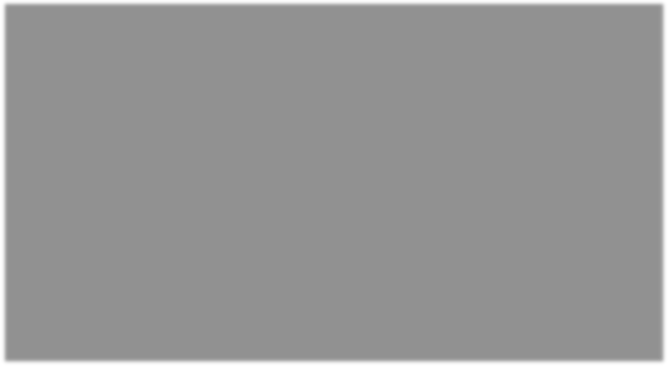
}

export default mainboard;

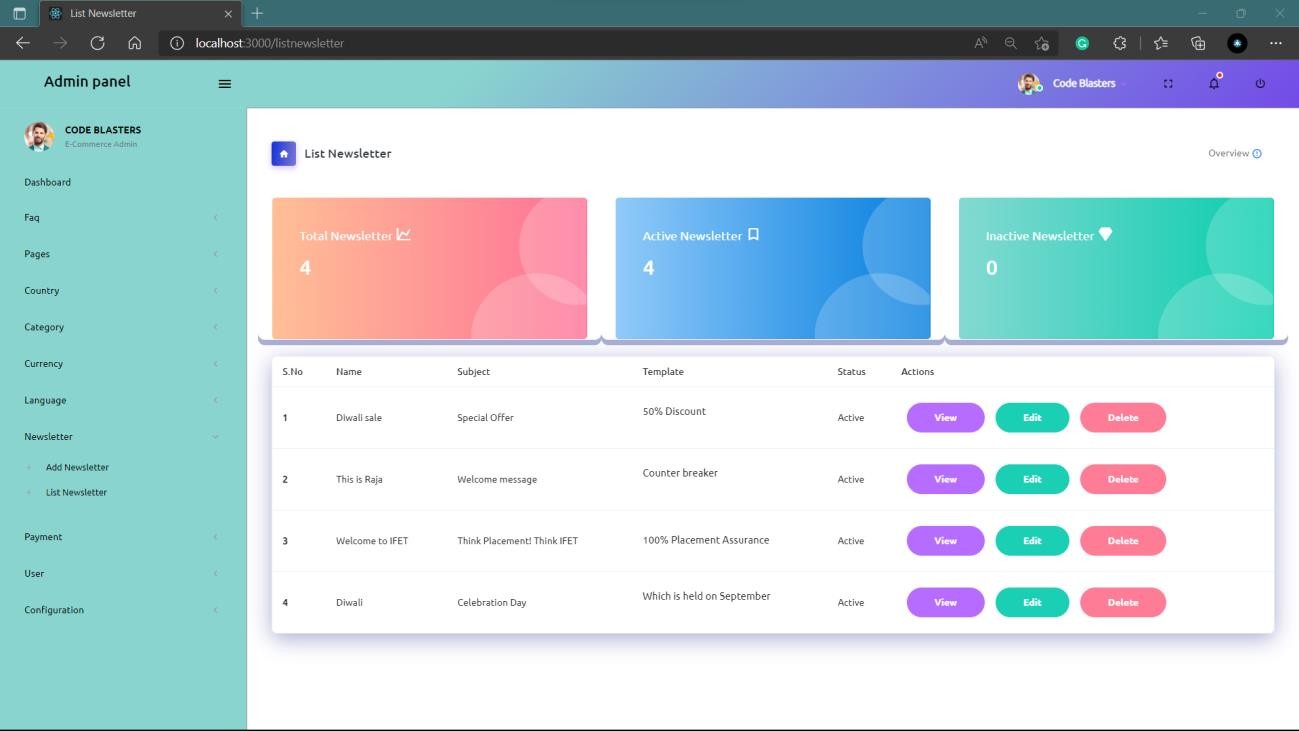
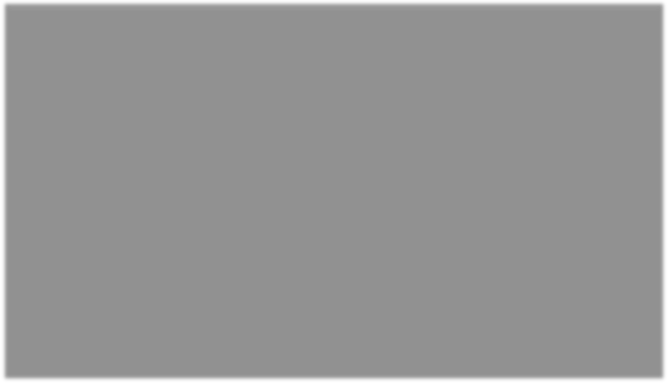
# SNAPSHOTS:



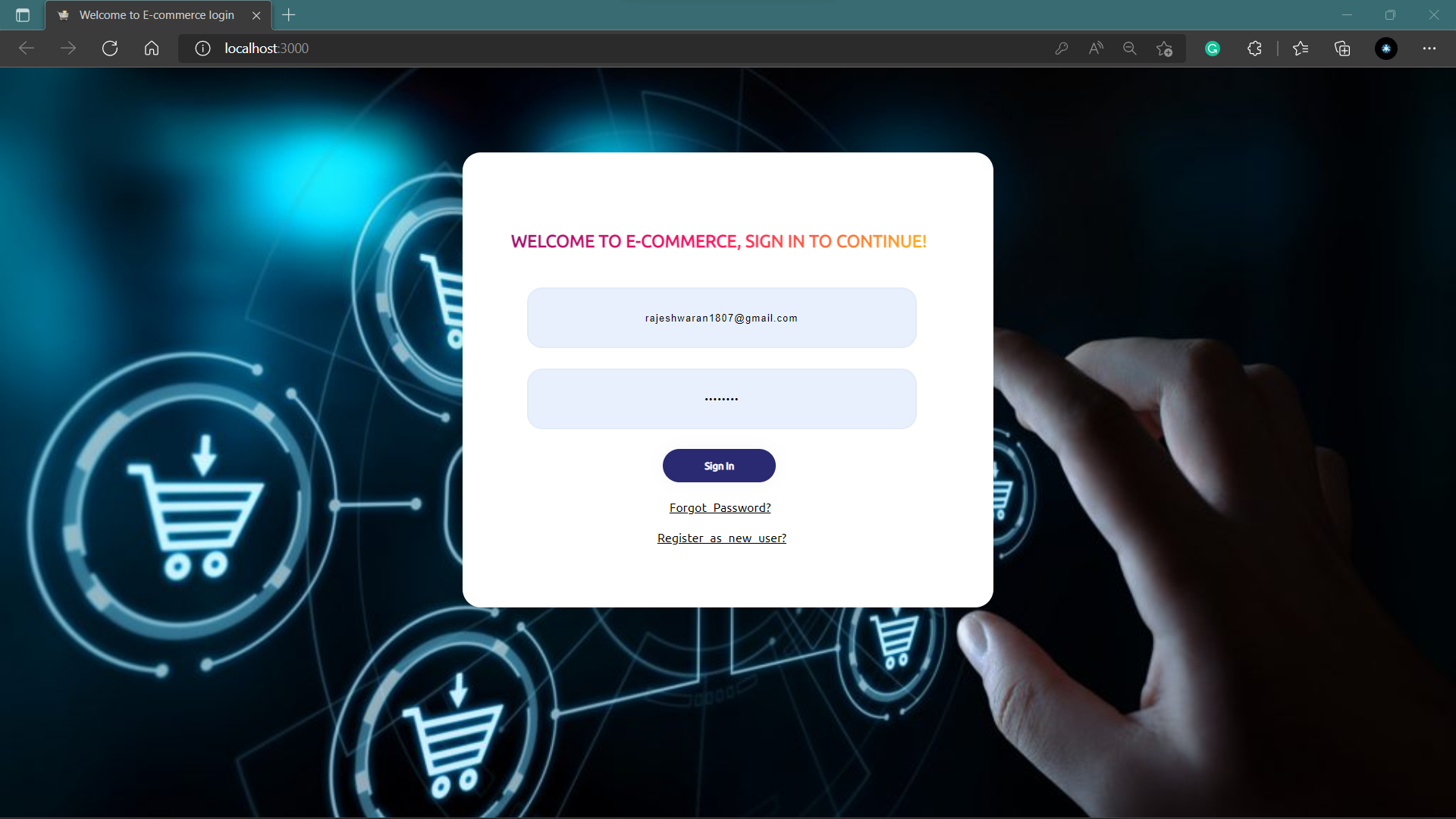
**FIGURE 9.1 ADMIN LOGIN**



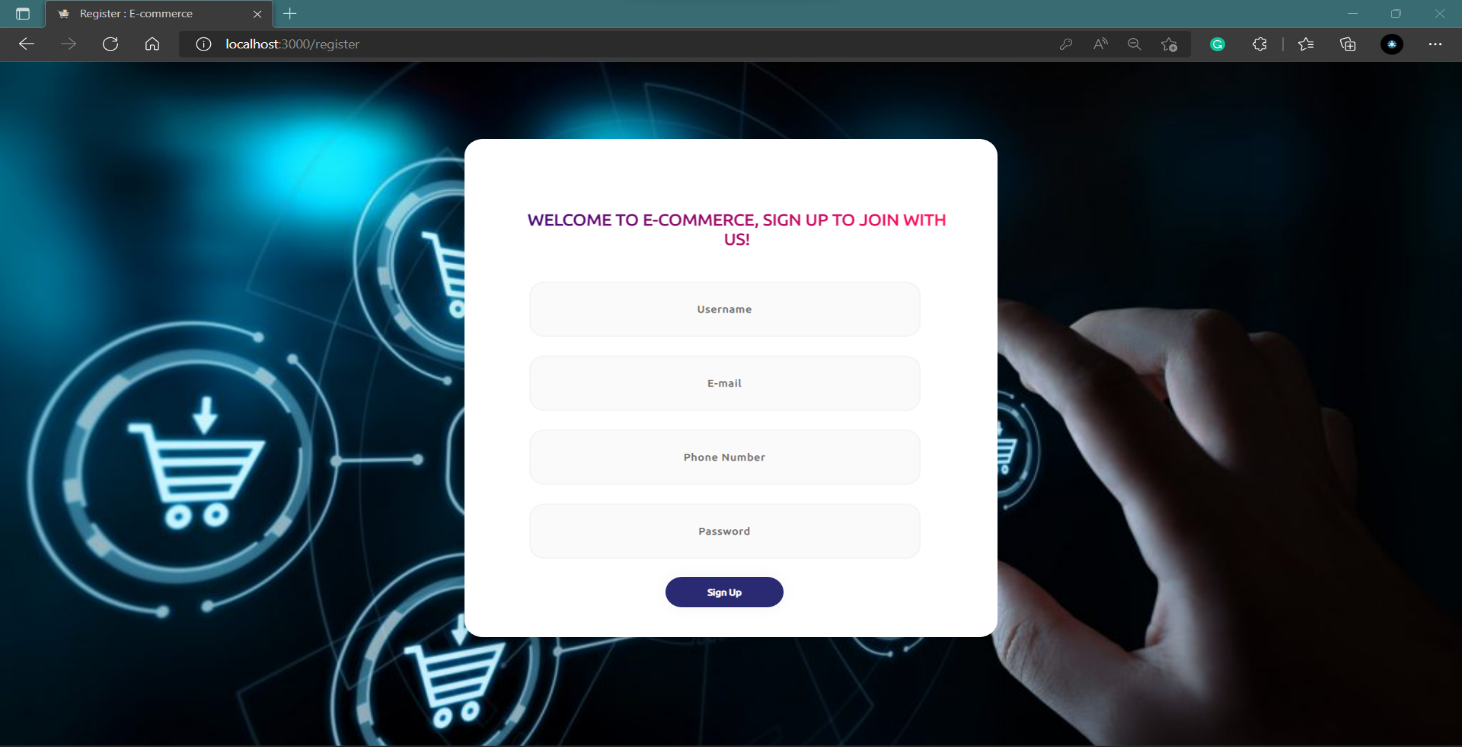
**FIGURE 9.2 ADMIN DASHBOARD**



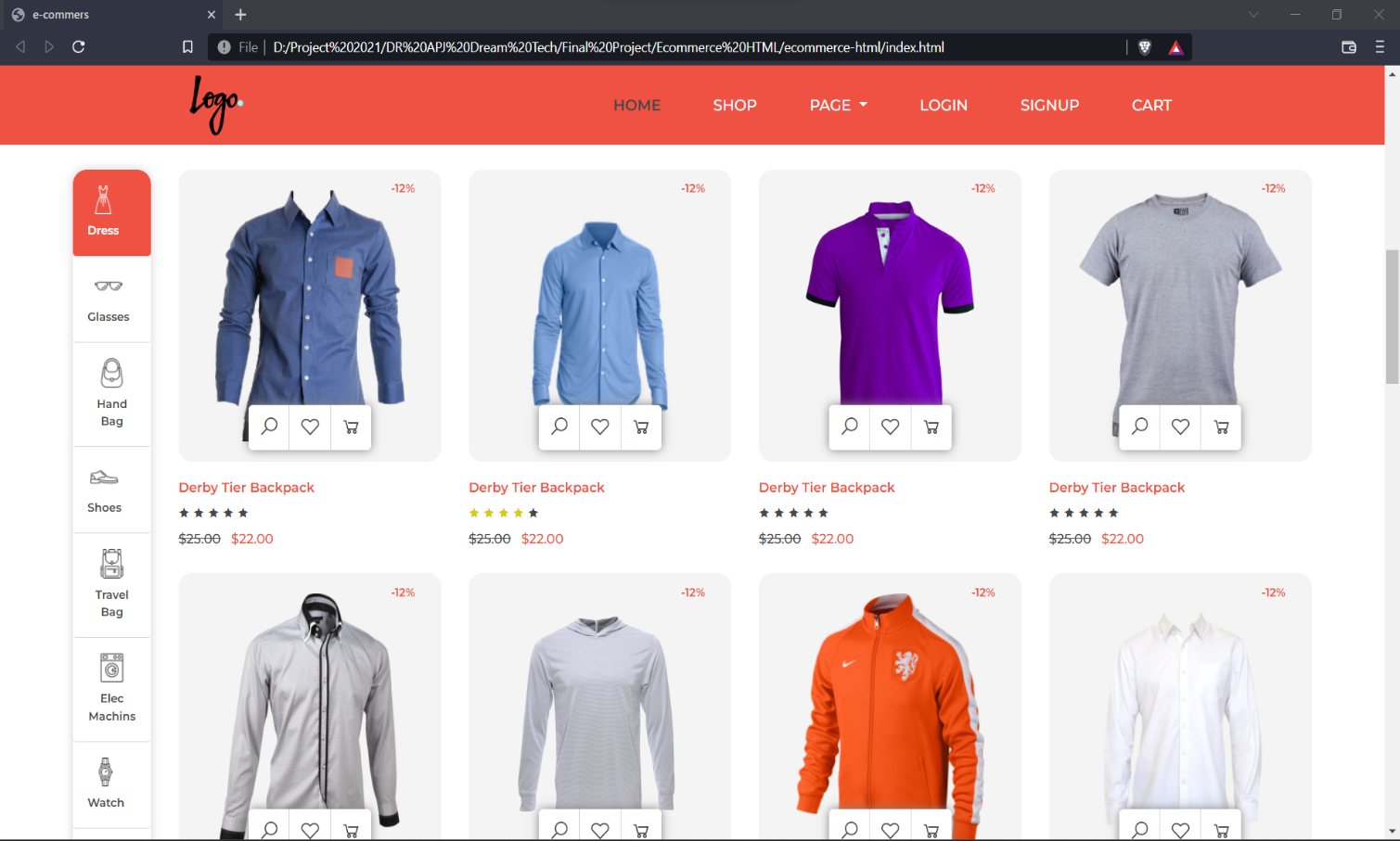
**FIGURE 9.3 ADMIN NEWSLETTER MANAGEMENT**



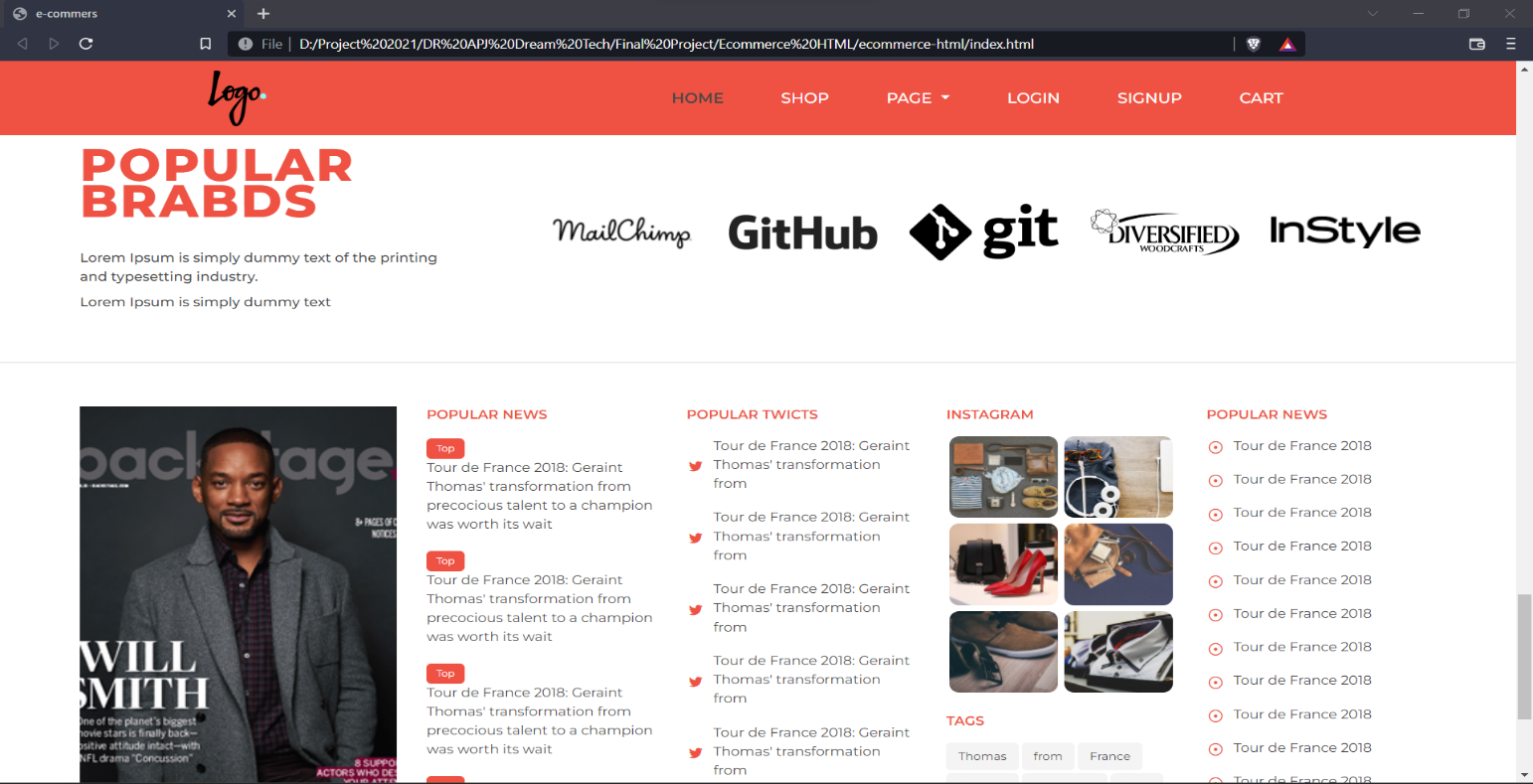
**FIGURE 9.4 SITE LOGIN**



**FIGURE 9.5 SITE REGISTER**



**FIGURE 9.6 SITE HOMEPAGE**



**FIGURE 9.7 SITE NEWSLETTER**

# APPENDIX II REFERENCES

1. A Review of Usability and Security Evaluation Model of Ecommerce Web - Nur Azimah bt Mohd, Zarul Fitri Zaaba
2. E-Commerce for the 21st Century - Scott Hamilton
3. Opinion Observer: Recommendation System on ECommerce Website -

Mohammad Daoud

1. Integrated Approach to e-Commerce Websites Evaluation with the Use of Surveys and Eye Tracking Based Experiments - Paweł Ziemba
2. Learning to detect and measure fake ecommerce websites in search-engine results - Claudio Carpineto
3. Evaluating ecommerce websites cognitive efficiency: An integrative framework based on data envelopment analysis - Corrado lo Storto
4. The Research on E-commerce Website Success Mode - Xu Hao
5. Development of a Data Acquisition System for an Ecommerce Website - Paul Hughes Regis University
6. Research on the evaluation of e-commerce website under the environment of big data - Pingping Dong
7. Research on E-commerce Business Model based on Intelligent Terminals Mobile - Hongyan Xin
8. Designing E-commerce User Interface - Noorfadzilah Md Zainudin
9. Measuring the Performance of Ecommerce Websites - Ahmad Ghandour University of Otago, [aghandour@infoscience.otago.ac.nz](mailto:aghandour@infoscience.otago.ac.nz)
10. Developing an E-Commerce Website - Syed Emdad Ullah, Tania Alauddin and Hasan U. Zaman
11. Subjective Judgment, Cognitive Style and Ecommerce Website Evaluation:

A Non-Parametric Approach - Corrado lo Storto

1. USER EXPERIENCE ON AN ECOMMERCE WEBSITE – A CASE STUDY Jyväskylä University School of Business and Economics Master’s Thesis
2. Practical Eye Tracking of the Ecommerce Website User Experience - Wilkey Wong, Mike Bartels, and Nina Cheroot
3. JWT HANDBOOK – BY Sebastian pyrott