# CHAPTER 1

# INTRODUCTION

## 1.1 INTRODUCTION

The project entitled " **CUSTOMCORE** " is a web-based application designed to streamline the process of building and customizing personal computers, as well as exploring pre-built PC configurations. Developed using the MERN (MongoDB, Express, React, Node.js) stack, this platform provides users with an intuitive interface to select compatible components and optimize their builds effortlessly With access to a vast component database and an intelligent build wizard, users can ensure compatibility and performance while receiving real-time pricing updates, component reviews, and expert recommendations. Community-driven features, such as forums and build-sharing capabilities, enable knowledge sharing and collaboration among PC enthusiasts and novices alike.

**1.2 ABOUT THE TECHNOLOGY**

This technology selected for implementing **CUSTOMCORE** in MERN (MongoDB, Express, REACT, NodeJS) stack. The MERN stack is a technology stack for building full-stack web applications, and it is not tied to any specific server. Instead, it is a combination of four technologies: MongoDB for data storage, Express for the back-end web application framework, React for the front-end user interface, and Node.js for the server-side JavaScript runtime environment.

**MongoDB**

MongoDB is a popular open-source NoSQL database management system that stores data in flexible, JSON-like documents rather than fixed tables. Instead of rows and columns, MongoDB uses collections and documents, allowing for varied and evolving data structures. This design provides high performance, scalability, and ease of use for modern web applications.When setting up a MongoDB database, you typically start by installing the MongoDB server locally or by using a cloud-based service like MongoDB Atlas. Once installed, you create a database by simply naming it, and then you create collections within that database. Collections in MongoDB are similar to tables in relational databases, but they are schema-less by default. This means you can insert documents into a collection without needing to define a strict structure beforehand.There are several ways to create and manage your MongoDB database. You can use the MongoDB shell to execute commands for creating databases, collections, and documents, or you can opt for a graphical tool like MongoDB Compass, which offers a user-friendly interface to perform these tasks. Additionally, MongoDB provides drivers for various programming languages, enabling you to integrate database operations directly into your application code.This flexibility makes MongoDB a strong choice for applications that require rapid development and the ability to handle diverse data formats without the constraints of a rigid schema.

#### EXPRESS

Express is a popular minimalist web application framework for Node.js that provides a simple, yet powerful set of features for building web applications and APIs. It provides a robust set of HTTP utility methods and middleware functions, allowing developers to easily build web applications that can handle a variety of HTTP requests, routing, and more. Express is also highly modular and can be extended with additional middleware and plugins to enhance its functionality, making it a popular choice for developers looking for a lightweight, flexible web application framework.

**REACT**

React is a popular JavaScript library used for building user interfaces in web applications. It provides a component-based architecture that allows developers to create reusable and independent building blocks of the UI. React uses a virtual representation of the DOM, which makes it efficient in updating the UI. It follows a unidirectional data flow model, where data flows from the parent components to the child components, making it easier to manage the state of the application. React uses JSX syntax, which allows developers to write HTML-like code within their JavaScript code. React has a large and active ecosystem with many tools, libraries, and frameworks available for state management, routing, testing, and more.

**NODE.JS**

Node.js is a server-side JavaScript runtime built on the V8 JavaScript engine used for building scalable network applications. It provides an event-driven, non-blocking I/O model that makes it lightweight and efficient, making it well-suited for building real-time, data-intensive applications. Node.js uses modules that provide functionality for specific tasks, making it highly modular and easy to use. Its popularity is largely due to its ability to handle large amounts of traffic with low latency and high concurrency, as well as its large and active community of developers who contribute to its ecosystem of tools and libraries.

## 1.3 ABOUT THE PROJECT

**CUSTOMCORE** is a web application that features three main modules: user, admin, and PC builder. The user module enables individuals to design and customize their personal computers by selecting compatible components from an extensive database. The admin module provides tools for managing the component database, updating prices, and ensuring compatibility checks. Finally, the PC builder module assists users in assembling their selected components, providing guidance on compatibility, performance optimization, and build completion.

Overall, **CUSTOMCORE** aims to revolutionize the traditional PC-building process by offering a more efficient, secure, and user-friendly platform that caters to both beginners and experienced enthusiasts.

**CHAPTER 2**

# SYSTEM STUDY

## 2.1 INTRODUCTION

System analysis or study is an important phase of any system development process. The system is studied to the minutest details and analysed. The system analyst plays the role of an interrogator and dwells deep into the working of the present system. The system is viewed as a whole and the inputs to the system are identified. The outputs from the organization are traced through the various processing that the input phases through in the organization. detailed study of this process must be made by various techniques like interviews; questionnaires etc. the data collected by these sources must be scrutinized to arrive to a conclusion.

## 2.2 SYSTEM STUDY

System study is the process of gathering and interpreting facts, using the information for further studies on the system. It is a problem by solving activity that requires intensive communication between the system users and system developers. It does various feasibility studies. In these studies, a rough figure of the system activities can be obtained, from which the decisions about the strategies to be followed for effective system study and analysis can be taken. Preliminary study also identifies the method collection to be followed.

At the system study conducted an initial picture about the system working was got. From the information got form the study, the data collection methods are identified. Even in the first investigation itself drawback of the existing system could be identified.

## 2.3 REQUIREMENT GATHERING AND ANALYSIS

The analyst starts the requirements gathering and analysis activity by collecting all information from the customer, which could be used to develop the requirement of the system. He then analyses the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to removing all ambiguities and inconsistency from the initial customer perception of the problem. This may sound like a simple task. However, in practice it is difficult to gather the necessary information and to form a non-ambiguity understanding problem. No elaborates the two main activities involved in the requirements gathering and analysis phase:

**Requirement gathering** - This activity typically involves interviewing the end users and customers and studying the existing documents to collect all possible information regarding the system. If the project involves automating some existing procedures, then the task of the system analyst becomes a little easier as he can immediately obtain the input and output data format and the details of the operational procedures.

**Analysis of gathered requirements** - The main purpose of this activity is to clearly understand the exact requirements of the customer. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain the good grasp of the problem:

* Identify and define the problem?
* Why is it important to solve the problem?
* What is the possible solution to the problem?
* What exactly are the data input to the system and what exactly are the data output required of the system?
* What is the likely complexity that might arise while solving the problem?
* If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange format with the external system be?

After the analyst has understood the exact customer requirements he proceeds to identify and resolve the various requirements problems. The most important requirement’s problems that the analyst has to identify are anomalies, inconsistencies and incompleteness.

**Anomaly**: An anomaly is an ambiguity in the requirement. When a requirement is anomalous, several interpretations of the requirement are possible.

**Inconsistency:** The requirement becomes inconsistent, if anyone of the requirement contradicts another.

**Incompleteness:** An incomplete requirement is one where some of the requirements have been overlooked. Often, incompleteness is caused by the inability of the customer to visualize and anticipate all the features that would be required in a system to be developed. An experienced analyst can usually detect these missing features easily and suggest them to the customer for his consideration.

The solutions are given as a proposal. The proposal is then weighed with the existing system analytically and the best one is selected. The proposal is presented to the user. The proposal is reviewed on user request and suitable changes are made. This is a loop as the user is satisfied with the proposal.

**2.4 EXISTING SYSTEM**

The existing system for building and customizing personal computers is highly complicated, requiring users to manually research component compatibility, track multiple specifications, and compare prices across different platforms. Managing these details manually is time-consuming, error-prone, and can result in suboptimal PC configurations. Additionally, the lack of an integrated system makes it difficult for users to keep track of their builds, get real-time pricing, and receive expert recommendations.

**DRAWBACKS OF EXISTING SYSTEM**

The main drawbacks of the existing system include being time-consuming, as users must manually research and verify component compatibility, which can be tedious and overwhelming. It is also costly since purchasing individual components without price comparison tools often leads to overspending. Data management is another issue, as keeping track of PC builds, component specifications, and pricing manually is difficult and inefficient. Additionally, the lack of automation makes it challenging for users to optimize their builds and ensure compatibility. To overcome these limitations, the CUSTOMCORE system provides a computerized solution that simplifies the entire PC-building process.

## 2.5 PROPOSED SYSTEM

### The proposed system eliminates the need for manual tracking and research by providing a fully digital and automated platform for PC building. Users can easily select compatible components without worrying about compatibility issues, as the system ensures seamless integration. An intuitive interface allows users to build custom PCs or explore optimized pre-built configurations based on their needs. A real-time component database keeps track of specifications, pricing, and availability, ensuring accurate and up-to-date information. The system includes features for managing user accounts, storing purchase records, tracking build histories, and ensuring easy retrieval of past configurations. It also simplifies the process of selecting and comparing components, reducing errors and saving time.

### 2.5.1 ADVANTAGES OF PROPOSED SYSTEM

The main advantage of the proposed system is user convenience. Each module is designed to require minimal input from the user, ensuring a seamless and intuitive experience. The system is highly secure, allowing access only to authorized users through a username and password authentication mechanism. Users can submit their feedback and complaints effortlessly, ensuring prompt responses. Another key advantage is that it fosters collaboration by allowing users to connect, share ideas, discuss PC builds, and work on projects together, creating a more interactive and engaging experience.

**2.6 FEASIBILITY STUDY**

The feasibility study is carried out to determine whether the proposed system can be developed with available resources. A feasibility study is a test of a system proposal according to its workability, impact on organization, ability to meet user needs, and effective use of resources.

The results of the feasibility study are given below.

### 2.6.1 ECONOMIC FEASIBILITY

A broadband infrastructure is already available in the market. Use of Internet based network would thus minimize the expenses as no separate WAN will be required. Open source technology could easily be adopted to further minimize the cost of software. The client computers would merely be ordinary PCs connected to the Internet, which all the formations already have. Thus, the “**CUSTOMCORE**” is economically feasible.

The given system can be developed under optimal expenses with the available hardware and software. Besides it is a good economic to invest in such a kind of software from the project manager’s point of view as the benefits overweighs the costs. The resources needed to run the above projects should be less in cost and highly reliable so that there might be no hanging and minimum level of expense to implement the software. Economic feasibility is the most frequently used method for evaluating the effectiveness of a candidate system. More commonly known cost/benefit analysis the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs.

### 2.6.2 TECHNICAL FEASIBILITY

A feasibility study is carried out to select the best system that meets performance requirements. The input to the feasibility assessment study is the statement of user needs from the information analysis phase. Assessing technical feasibility is to evaluate whether the “**CUSTOMCORE**” will perform adequately and whether a PDS has ability to construct a proposed system or not. The application is the fact that it has been developed on Windows 10 platform and a high configuration of 8GB RAM on Intel I5 processor. This is technically feasible. To ensure our system is technically feasible we evaluate some technical feasibility measurement criteria’s are:

**Project Size:**

Project size can be determined by the number of members on the project team, project duration time, number of department involved, or the effort put in programming.

**Project Structure:**

The project “**CUSTOMCORE**” that its requirements are highly structured and well define will have lower risk than the one that the requirements are subject to the judgment of an individual. Familiarity with Technology or Application area: The project is less risky because the development and the user group is familiar with the technology mainly the MERN stack. Therefore, it would be less risky the development team uses the standard development tool and hardware environments. The “**CUSTOMCORE**” completely ensures all factors mentioned above so we can conclude our proposed system technically feasible. It should be based on specified technology. The system under study must be portable and platform independent.

**BEHAVIOURAL FEASIBILITY:**

People are inherently resistant to change, and computers have been known to facilitate change. An estimate should be made of how strong a reaction the user staff is likely to have toward the development of a computerized system. It is common knowledge that computer installations have something to do with turnover, transfers, retraining, and changes in employee job status. Therefore, it is understandable that the introduction of a candidate system requires special effort to educate, sell, and train the staff on new ways of conducting business. In our safe deposit example, three employees are more than 50 years old and have been with the bank over 14 years, four years of which have been in safe deposit. The remaining two employees are in their early thirties. They join safe deposit about two years before the study. Based on data gathered from extensive interviews, the younger employees want the programmable aspects of safe deposit (essentially billing) put on a computer. Two of the three older employees have voiced resistance to the idea. Their view is that billing is no problem. The main emphasis is customer service-personal contacts with customers. The decision in this case was to go ahead and pursue the project.

### 2.6.3 OPERATIONAL FEASIBILITY

The present system is easily understandable and the maintenance and working of the new system needs less human effort. The proposed project is beneficial to the society and is user friendly. The system is directly used by the users directly with much knowledge. So the system can be judged operationally feasible.

### 2.6.4 HARDWARE AND SOFTWARE FEASIBILITY

Assessing technical feasibility includes evaluating the ability of computer hardware and software to handle workloads adequately. Figure below shows the steps the systems analyst takes in ascertaining hardware and software needs. First, all current computer hardware the organization owns must be inventoried to discover what is on hand and what is usable. The systems analyst needs to work with users to determine what hardware will be needed. Hardware determinations can come only in conjunction with determining human information requirements. Knowledge of the organizational structure (as discussed in Chapter 2) and how users interact with technologies in an organizational setting can also be helpful in hardware decisions. Only when systems analysts, users, and management have a good grasp of what kinds of tasks must be accomplished can hardware options be considered.

### 2.6.5 LEGAL FEASIBILITY

Using this feasible study, we determine whether the proposed system conflicts with legal requirements. All the user data and privacy will be website. Our system is provided authentication by using username and password. At any time, user can delete/remove their account from our application, which will remove entire data of the user. Email addresses are not being used for any other purpose except the CUSTOMCORE application.so I can conclude our proposed system legally feasible.

### 2.6.6 SCHEDULE AND RESOURCE FEASIBILITY

It is also an essential part of a feasibility study. It includes questions regarding time required to complete the project, type and amount of resources required and dependent factors. It also takes care whether the project is interrupting any current business activity.

# CHAPTER 3

# SOFTWARE REQUIREMENT SPECIFICATIONS

# 3.1 INTRODUCTION

After the analyst has collected all the required information regarding the software to be developed, and has removed all completeness, inconsistence, and anomalies from the specification, he starts to systematically organize the requirements in the form of an SRSdocument. The software developers refer to the SRSdocument to make sure that they develop exactly what the customer requires. The SRS document helps the maintenance engineers to understand the functionality of the system. An SRS document should be clearly specifying:

* Functional requirements.
* Non-functional requirements.
* Goals of implementation.

## 3.2 OPERATING SYSTEM: WINDOWS 11

Windows 11 Professional integrates the strengths of previous Windows operating systems such as advanced security, manageability, and reliability, along with a modern and user-friendly interface. This combination makes it an ideal operating system for business and personal use. Windows 11 enhances productivity with features like Snap Layouts, a redesigned Start Menu, and integrated Microsoft Teams for easy communication.The operating system also supports advanced technologies such as virtual desktops, high-definition video conferencing, and gaming improvements. It includes built-in security features like Windows Hello, BitLocker, and secure boot for better protection against threats. Windows 11 offers improved performance, faster boot times, and better hardware support.

Following are the key features of Windows 11:

* **Improved User Interface:** New Start Menu, rounded corners, and smooth design.
* **Snap Layouts:** Easy multitasking by arranging windows.
* **Microsoft Teams Integration:** Quick communication from the taskbar.
* **Enhanced Gaming:** DirectX 12 and Auto HDR for superior gaming.
* **Better Security:** Built-in antivirus, firewall, and encryption support.

Windows 11 provides a powerful and flexible operating system designed to enhance user productivity and deliver a seamless computing experience.

## 3.3 FRONT END:

React is a popular JavaScript library for building user interfaces. It is designed to make it easy to create complex, interactive web applications with a declarative and component-based approach.

React was created by Facebook and was first released in 2013. Since then, it has become one of the most widely used front-end libraries, with a large and active community of developers.

These are the key features of React:

Component-based architecture: React applications are built using components, which are reusable and independent building blocks of the user interface. Each component can have its own state, properties, and lifecycle methods, making it easy to manage and update the UI.

Virtual DOM: React uses a virtual representation of the DOM, which is a lightweight copy of the actual DOM. This makes it faster and more efficient to update the UI, as React can compare the virtual DOM to the actual DOM and only update the necessary changes.

JSX syntax: React uses a syntax called JSX, which allows developers to write HTML-like code within their JavaScript code. This makes it easier to create complex user interfaces and maintain a clear separation of concerns between the UI and logic.

Unidirectional data flow: React follows a unidirectional data flow model, where the data flows from the parent components to the child components. This makes it easier to manage the state of the application and maintain a clear flow of data throughout the application.

Large ecosystem: React has a large and active ecosystem, with a variety of tools, libraries, and frameworks that can be used to enhance and extend its functionality. This includes tools for state management, routing, testing, and more.

React is commonly used in combination with other technologies to build full-stack web applications, such as the MERN stack (MongoDB, Express, React, and Node.js) or the JAMstack (JavaScript, APIs, and Markup). Its flexibility and robustness make it an ideal choice for building modern, scalable, and interactive web applications.

## Characteristics of React

React is a JavaScript library used for building user interfaces. Some of its key characteristics include.

* Declarative: React allows you to describe how your UI should look and behave in a declarative way. You define the state of your application and React takes care of updating the UI to match that state.
* Component-based: React is based on a component-based architecture. Components are reusable building blocks that can be composed to create complex user interfaces.
* Virtual DOM: React uses a virtual DOM to improve performance. The virtual DOM is a lightweight copy of the actual DOM that React can manipulate more efficiently.
* JSX: React uses JSX, a syntax extension to JavaScript, to allow you to write HTML-like code in your JavaScript files. JSX is compiled to plain JavaScript before being executed.
* Unidirectional data flow: React follows a unidirectional data flow pattern, where data flows down from parent components to child components. This makes it easier to reason about how data is flowing through your application.
* React Native: React can also be used to build native mobile apps for iOS and Android using React Native. React Native allows you to write cross-platform code using JavaScript, while still delivering a native app experience.

## 3.4 BACKEND

**Express**

Express is a back-end web application framework for Node.js that provides a robust set of features for building web APIs and handling HTTP requests. It is designed to be minimalistic and flexible, allowing developers to build complex web applications with ease.

Express provides a wide range of features for building server-side applications, including:

Routing: Express makes it easy to define routes for handling HTTP requests. Developers can define a route for a specific URL, HTTP method, and middleware functions that will be executed when the route is accessed.

Middleware: Express allows developers to define middleware functions that are executed before or after a request is handled. Middleware functions can be used to perform authentication, logging, error handling, and other tasks.

Template engines: Express supports a variety of template engines, such as EJS, Handlebars, and Pug, which can be used to generate HTML pages dynamically based on data from the server.

Error handling: Express provides a robust error-handling system that allows developers to define custom error-handling middleware functions. This can help to catch and handle errors that occur during the request-response cycle.

Static file serving: Express can be used to serve static files, such as images, CSS files, and JavaScript files, which can be used to build web applications with rich user interfaces.

Express is designed to be flexible and easy to use, making it an ideal choice for building web APIs and server-side applications. Its minimalistic approach means that it can be easily extended with other middleware and third-party packages, allowing developers to customize it to their needs.

**Node.JS**

Node.js is an open-source, server-side JavaScript runtime that allows developers to build fast, scalable, and highly performant applications. It uses an event-driven, non-blocking I/O model, which makes it highly efficient and allows it to handle large volumes of data and users.

Node.js was first released in 2009 by Ryan Dahl and has since become one of the most popular and widely used platforms for building web applications, APIs, and micro services. It has a large and active community of developers and is supported by major companies like Microsoft, IBM, and Google.

Here are some key features of Node.js:

Asynchronous programming: Node.js is designed to handle I/O-intensive tasks and is optimized for asynchronous programming. This means that multiple requests can be handled simultaneously without blocking the event loop, making it highly scalable and efficient.

Large module library: Node.js has a large and comprehensive library of modules and packages that can be used to enhance its functionality. These modules can be easily installed and managed using the Node Package Manager (NPM).

Cross-platform compatibility: Node.js is designed to be cross-platform compatible, which means that it can be run on multiple operating systems like Windows, macOS, and Linux.

Single-threaded architecture: Node.js uses a single-threaded event loop architecture, which makes it highly efficient and allows it to handle a large number of concurrent connections.

Easy to learn: Node.js is built on top of JavaScript, which is one of the most widely used programming languages in the world. This means that developers who are familiar with JavaScript can easily learn and use Node.js to build server-side applications.

Node.js is commonly used in combination with other technologies to build full-stack web applications, such as the MERN stack (MongoDB, Express, React, and Node.js) or the MEAN stack (MongoDB, Express, Angular, and Node.js). Its flexibility, efficiency, and scalability make it an ideal choice for building modern, high-performance web applications.

**MongoDB**

MongoDB, the most popular open-source NoSQL database management system, is developed, distributed, and supported by MongoDB Inc. MongoDB Atlas is the most reliable, secure, and up-to-date version of MongoDB for cost-effectively delivering modern applications, large-scale data processing, and real-time analytics. It is a fully integrated document-based, schema-less database with high scalability, automatic failover, and horizontal scaling capabilities. MongoDB delivers the ease of use, flexibility, and performance that has made it the world's most popular NoSQL database.

MongoDB supports high-volume data storage and is widely used in modern web applications, especially for handling large amounts of unstructured or semi-structured data. With its JSON-like document model, it provides a natural way to store and query data, enabling faster and more efficient development processes.

### 3.4.1 FEATURES OF MongoDB COMMUNITY

**3.4.1.1 IT IS A DATABASE MANAGEMENT SYSTEM.**

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MongoDB Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

**3.4.1.2 MongoDB DATABASE IS DOCUMENT-ORIENTED**

document-oriented database stores data in flexible, JSON-like documents instead of traditional tables. The database structures are organized into collections rather than tables, optimized for scalability and high performance. The document model, with fields and data types, offers a flexible and dynamic programming environment. You can create, read, update, and delete data with simple queries using MongoDB Query Language (MQL). MongoDB uses BSON (Binary JSON) format to store data, allowing rich data types and fast processing.

**3.4.1.3 IT IS OPEN SOURCE.**

Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MongoDB software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MongoDB software uses the Server Side Public License (SSPL), to define what you may and may not do with the software in different situations. If you feel uncomfortable with the SSPL or need to embed MongoDB code into a commercial application, you can buy a commercially licensed version from MongoDB Inc.

#### 3.4.1.4 MongoDB DATABASE SERVER IS FAST, RELIABLE, SCALABLE, AND EASY TO USE.

If that is what you are looking for, you should give it a try. MongoDB Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MongoDB, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MongoDB can also scale up to clusters of machines, networked together. MongoDB is designed to handle large-scale data applications efficiently and provide high performance, scalability, and reliability.

#### 3.4.1.5 THIS WORKS CLIENT / SERVER / EMBEDDED SYSTEMS

The MongoDB Database Software is a client/server system that consists of a multi-threaded server that supports different backend, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs). MongoDB also provides embedded database solutions for IoT and mobile devices, ensuring lightweight and fast data processing.

**3.4.1.6 LARGE AMOUNT OF CONTRIBUTED MongoDB IS AVAILABLE**

MongoDB Server has a practical set of features developed in close cooperation with the open-source community. It is very likely that your favourite application or language supports the MongoDB Database Server. The official way to pronounce "MongoDB" is "Mongo Dee Bee," but some users also refer to it as "Mongo."

* + 1. **INTERNALS AND PORTABILITY WRITTEN IN C++**
* Works on many different platforms.
* For portability, uses CMake in the build process.
* Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool.
* Uses multi-layered server design with independent modules.
* Designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available.
* Provides transactional and non-transactional storage engines.
* Uses WiredTiger as a default storage engine for high performance.
* Designed to make it relatively easy to add other storage engines.
* Uses a very fast thread-based memory allocation system.
* Provides an in-memory engine for temporary data storage.
* Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications.

### 3.4.3 FEATURES OF MongoDB 5.0

* The following features have been added to MongoDB 5.0Native Time Series collections.
* Live resharding to manage large datasets.
* Versioned API to ensure compatibility.
* Improved query performance with indexing.
* Enhanced aggregation framework.
* Better data encryption and security controls.
* Multi-document ACID transactions.
* Automated and distributed backup solutions.
* Improved cluster management.

**WEB SERVER**

Node.js is a high-performance JavaScript runtime environment built on Chrome's V8 engine. It is widely used to develop server-side applications and can seamlessly integrate with MongoDB for backend data storage. Node.js allows fast data processing and supports asynchronous, non-blocking I/O operations, making it ideal for real-time web applications. MongoDB is often used with Node.js to build full-stack web applications with high performance and scalability.

## 3.5 HARDWARE REQUIREMENTS

Selection of hardware configuration is very important task related to the software development. The processor should be powerful to handle all the operations. The hard disk should have the sufficient capacity to solve the database and the application.

### 3.5.1 MINIMUM HARDWARE REQUIREMENTS

Processor : 1.5 GHz or Higher processors

RAM : 4 GB RAM

Hard Disk Drive : 200 GB SSD

Memory : 512 MB or higher

Keyboard : Standard key board

Mouse : Standard Mouse

Monitor : Digital colour/LCD/LED monitor

### 3.5.2 MINIMUM SOFTWARE REQUIREMENTS

Front end : React

Back end : MYSQL, Node.JS, Express

Web server : V8 JavaScript Engine

Operating System : Windows 10

**CHAPTER 4**

# SYSTEM DESIGN

## 4.1 INTRODUCTION

System design is an interactive process through which requirements are transmitted to a “blue print” for constructing the software initial; the blue print depicts a holistic view of software that is design is represented at a high level of abstraction a level that can be directly traced to specific data, functional and behavioural requirements. As design interaction occur subsequent refinement leads to design representation at much lower levels of abstraction. System design is a creative art of inventing and developing inputs, databases, offline files, methods and procedures, for processing data to get meaningful output that satisfy the organization objectives. Through the design phase consideration to the human factors, i.e., inputs to the users will have on the system.

Some of the main factors that have to be noted using the design of the system are:

### 4.1.1 PRACTICABILITY

System must be capable of being operated over a long period of time and must have ease of use.

### 4.1.2 EFFICIENCY

Make better use of available resources. Efficiency involves accuracy, timeliness and comprehensiveness of system output

**4.1.3 COST**

Aim of minimum cost and better results.

**4.1.4 SECURITY**

Ensure physical security of data.

## 4.2 BASIC MODULES

### 4.2.1 ADMIN MODULE

This section is intended to the admin part of the application.

The facilities of the admin are:

* Login with valid username and password.
* Manage Users details.
* Manage PC builder details.
* Insert new components into the database.
* Manage District And Places.
* Manage Complaints.
* Mange Enquiry.
* Logout.

### 4.2.2 USER MODULE

This section covers all functionalities related to users. Users can perform the following actions:

* Login with username and password.
* Profile management.
* User registration.
* View and select PC components for the custom pc build.
* View and purchase pre-built PCs from a curated selection.
* Manage Custom Pc Buliding.
* Add Feedback.
* Add Complaint.
* Add Enquiry.
* Logout.

**4.2.3 PC BUILDER MODULE**

This module is designed specifically for PC builders to streamline the building process. PC builders can:

* Login with a username and password.
* Profile management.
* Add Feedback.
* Add Complaint.
* Manage Custom Pc Bookings.
* Mange Pre-Bulid pc Stock.
* Logout.

**4.3 DESIGN**

### 4.3.1 INPUT DESIGN

Input design is the process of converting user-oriented input to a based format. Inaccurate input data are the most common cause of errors in data processing. Errors entered by data entry operators can be controlled by input design.

The goal of designing input data is to make data entry as easy, logical and free from errors. When we approach input data design; we design the data source documents that capture the data and then select the media used to enter them into the computer.

User–friendly screen format can reduce the burden on end users, who are not highly proficient in computers. An important step in input design stage is a design of source document. Source document is the form in which the data are initially captured. The next step is the design of document layout. In the layout organizes the document by placing information, where it will be noticed and establishes the appropriate sequence of items.

In our system, almost all inputs are being taken from the databases. To provide adequate inputs we have to select necessary values from the databases and arrange it to the appropriate controls.

### 4.3.2 OUTPUT DESIGN

Computer Output is the most important and direct source of information to the user. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system’s relationship and helps user decision-making. In the output design it is determined how the implementation is to be played for immediate need and also the hard copy output. A major form of input is a hard copy from the printer. Printouts should be designed around the output requirements of the user. Printers, CRT screen display are the examples for providing computer based output. The output design associated with the system includes the various reports of table generations and query executions. Output design is one of the, most important features of the information system. The logical design of an information system is analogous to an engineering blue print of an auto mobile. It shows the major features and how they are related to one another. The outputs, inputs and databases are designed in this phase.

## 4.4 TABLE DESIGN

Database design is one of the most important parts of the system design phase. In a database environment common data available are used by several users. Instead of each program managing its own data, Authorized users share data across application with the database software managing the data as an entity. The primary objective of a database design is fast response time to enquiries, more information at low cost, control of redundancy, clarity and ease of use, date and program independence, accuracy and integrity of the system, fast recovery and availability of powerful end-user languages. The theme behind a database is to handle information as an integrated whole thus the main objective is to make information as access easy, quick, inexpensive and flexible for the users. Data directory specifies the major element in the system, and care should be taken while designing, in order to avoid unnecessary duplication of data. The entire package depends on how the data are maintained in the system. Several tables are maintained in the system to store data that are required for processing of various data as well as storing intermediate or final processed results.

### 4.4.1 TABLE: collectionadmins

This table contains details of Admin

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId (auto) | Stores admin id Auto-generated by MongoDB | PRIMARY KEY |
| adminName | String | Stores admin name | Required: true |
| adminEmail | String | Stores admin email | Required: true, Unique |
| adminPassword | String | Stores admin password | Required: true, Unique |

### 4.4.2 TABLE: collectionCase

This table contains the one of the purpose details which is Computer Case.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId (auto) | Stores Case id  Auto-generated by MongoDB | PRIMARY KEY |
| caseName | String | Stores name of Case | Required:true |
| caseImg | String | Stores image of Case | Required:true |
| caseDetails | String | Stores Case Details | Required:true |

### 4.4.3 TABLE: collectionCompany

This table contains Complaint details**.**

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId (auto) | Stores Company id Auto-generated by MongoDB | PRIMARY KEY |
| companyName | String | Stores Name of Companys | Required:true |

#### 4.4.4 TABLE: collectionComplaint

This table contains the category details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Complanit id Auto-generated by MongoDB | PRIMARY KEY |
| complaintTitle | String | Stores complaint title | Required: true |
| complaintContent | String | Stores complaint  Content | Required: true |
| userId | ObjectId | Stores userId from collectionUser | FOREIGN KEY |
| complaintReplay | String | Stores complaint  Replay | Required: true |
| pcbuliderId | ObjectId | Stores pcbuliderId  From collectionPcBulider | FOREIGN KEY |
| complaintStatus | String | Stores complaint  Status | Required:true |

#### 4.4.5 TABLE: collectionCooler

This table stores the details of Coolers

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Cooler id  Auto-generated by MongoDB | PRIMARY KEY |
| coolerName | String | Stores Cooler Name | Required:true |
| coolerImg | String | Stores image of Cooler | Required:true |
| coolerDetails | String | Stores Case Details | Required:true |

#### 4.4.6 TABLE: collectioncustom

#### This table stores details of Custom Pc

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Custom id  Auto-generated by MongoDB | PRIMARY KEY |
| motherboardId | ObjectId | Stores motherboard Id | FOREIGN KEY |
| storageId | ObjectId | Stores storage Id | FOREIGN KEY |
| ramId | ObjectId | Stores ram Id | FOREIGN KEY |
| companyId | ObjectId | Stores company Id | FOREIGN KEY |
| graphiccardId | ObjectId | Stores graphiccard Id | FOREIGN KEY |
| cpuId | ObjectId | Stores cpu Id | FOREIGN KEY |
| coolerId | ObjectId | Stores cooler Id | FOREIGN KEY |
| caseId | ObjectId | Stores Case Id | FOREIGN KEY |
| userId | ObjectId | Stores User Id | FOREIGN KEY |
| customStatus | String | Stores Status of the Custom Pc | Required: true |
| pcbuliderId | ObjectId | Stores pcbulider Id | FOREIGN KEY |

#### 4.4.7 TABLE: collectioncpus

#### This table stores details of chat

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Cpu id  Auto-generated by MongoDB | PRIMARY KEY |
| cpuName | String | Stores Cpu name | Required: true |
| companyId | ObjectId | Stores company Id | FOREIGN KEY |
| cpuImg | String | Stores cpu image | Required: true |
| cpuDetails | String | Stores cpu Details | Required: true |

#### 4.4.8 TABLE: collectiondistrict

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores District id  Auto-generated by MongoDB | PRIMARY KEY |
| districtName | String | Stores District Name | Required:true |

#### 4.4.9 TABLE: collectionfeedback

This table stores FeedBack details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Feedback id Auto-generated by MongoDB | PRIMARY KEY |
| FeedbackRecation | String | Stores Recation of Feedback | NOT NULL |
| FeedbackContent | String | Stores Content  of Feedback | Required: true |
| userId | ObjectId | Stores User Id | FOREIGN KEY |
| pcbuliddevId | ObjectId | Stores pcbulider Id | FOREIGN KEY |

**4.4.10 TABLE: collectiongraphiccard**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Graphic card id Auto-generated by MongoDB | PRIMARY KEY |
| graphiccardName | String | Stores graphiccard Name | Required:true |
| graphiccardImg | String | Stores image of Graphic Card | Required:true |
| graphiccardDetails | String | Stores Graphic Card Details | Required:true |

**4.4.11 TABLE: collectionmotherboard**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Motherboars id  Auto-generated by MongoDB | PRIMARY KEY |
| motherboardName | String | Stores Motherboard Name | Required:true |
| typeId | ObjectId | Stores type Id | FOREIGN KEY |
| companyId | ObjectId | Stores company Id | FOREIGN KEY |
| motherboardImg | String | Stores image of motherboard | Required:true |
| motherboardDetails | String | Stores motherboard Details | Required:true |

**4.4.12 TABLE: collectionpcbulider**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Pc Bulider id Auto-generated by MongoDB | PRIMARY KEY |
| pcBuliderName | String | Stores Motherboard Name | Required: true |
| pcBuliderEmail | String | Stores Pc Bulider Email | Required: true,  Unique |
| pcBuliderAddress | String | Stores Pc Bulider Address | Required: true |
| pcBuliderContact | String | Stores Pc Bulider Contact | Required: true |
| pcBuliderProof | String | Stores Pc Bulider Proof | Required: true |
| pcBuliderPhoto | String | Stores Pc Bulider Photo | Required: true |
| placeId | ObjectId | Stores placeId | FOREIGN KEY |
| pcBuliderPassword | String | Stores Pc Bulider Password | Required: true |

**4.4.13 TABLE: collectionplace**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Place id  Auto-generated by MongoDB | PRIMARY KEY |
| placeName | String | Stores Motherboard Name | Required: true |
| districtId | ObjectId | Stores District Id | FOREIGN KEY |

**4.4.14 TABLE: collectionprebulid**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Pre-Bulid id Auto-generated by MongoDB | PRIMARY KEY |
| motherboardId | String | Stores Motherboard Name | FOREIGN KEY |
| storageId | ObjectId | Stores storage Id | FOREIGN KEY |
| ramId | ObjectId | Stores Ram Id | FOREIGN KEY |
| companyId | ObjectId | Stores Company Id | FOREIGN KEY |
| graphiccardId | ObjectId | Stores Graphic Card Id | FOREIGN KEY |
| coolerId | ObjectId | Stores Cooler Id | FOREIGN KEY |
| CaseId | ObjectId | Stores Case Id | FOREIGN KEY |
| userId | ObjectId | Stores User Id | FOREIGN KEY |
| customStatus | String | Stores Custom Pc Status | Required: true |
| pcbuliderId | ObjectId | Stores Pc Bulider Id | FOREIGN KEY |

**4.4.15 TABLE: collectionram**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Ram id  Auto-generated by MongoDB | PRIMARY KEY |
| ramName | String | Stores Motherboard Name | Required:true |

**4.4.16 TABLE: CollectionSmps**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Smps id  Auto-generated by MongoDB | PRIMARY KEY |
| smpsName | String | Stores Motherboard Name | Required:true |
| smpsImg | String | Stores image of SMPS | Required:true |
| smpsDetails | String | Stores SMPS Details | Required:true |

**4.4.17 TABLE: collectionstorage**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Storage id  Auto-generated by MongoDB | PRIMARY KEY |
| storageName | String | Stores storage Name | Required: true |
| storageImg | String | Stores image of storage | Required: true |
| storageDetails | String | Stores storage Details | Required: true |

**4.4.18 TABLE: collectiontype**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores Type id  Auto-generated by MongoDB | PRIMARY KEY |
| typeName | String | Stores Type  Name | Required:true |

**4.4.19 TABLE: collectionUser**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores User id  Auto-generated by MongoDB | PRIMARY KEY |
| userName | String | Stores User Name | Required:true |
| userEmail | String | Stores User Email | Required: true,  Unique |
| userAddress | String | Stores User Address | Required: true |
| userPhoto | String | Stores User Photo | Required: true |
| userProof | String | Stores User Proof | Required: true |
| userContact | String | Stores User Contact | Required: true |
| placeId | ObjectId | Stores Place Id | FOREIGN KEY |
| userPassword | String | Stores User Password | Required: true |

**4.4.20 TABLE: collectionenquiry**

This table stores district details.

|  |  |  |  |
| --- | --- | --- | --- |
| **ATTRIBUTES** | **DATATYPE** | **DESCRIPTION** | **CONSTRIANT** |
| \_id | ObjectId(auto) | Stores enquiry id  Auto-generated by MongoDB | PRIMARY KEY |
| enquiryQs | String | Stores enquiry question | Required:true |
| enquiryAns | String | Stores enquiry Answers | Required: true |
| userId | ObjectId | Stores User Id | FOREIGN KEY |

## 4.5 DATA FLOW DIAGRAM

DFD (Data Flow Diagram) which is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. A data flow diagram is a network that describes the flow of data and processes that change, or transform, data throughout the system.

### 4.5.1 RULES FOR DRAWING DATA FLOW DIAGRAM

There are several rules for Data Flow Diagram and they are: -

* Process should be named and numbered for easy references.
* The direction of flow is from top to bottom and from left to right.
* When a process is imported in the lower levels details, they must be numbered.
* Process and data flow names have the 1st letter of the word must be capital letter.

### 4.5.2 BASIC DFD SYMBOLS

The primitive symbols used in the DFD are:

#### 4.5.2.1 EXTERNAL ENTITY

The external entities are essentially those physical entities external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system. For example, user of a system. Entities supplying data are known as sources and those that consume data are sinks.

#### 4.5.2.2 PROCESS

The functions are represented using circles. Bubbles are annotated with the names of the corresponding functions. They convert data into information.

#### 4.5.2.3 Data flow

A directed arrow or an arc is used as a dataflow symbol that represents the data flow occurring between two processes, or between an external entity and a process.

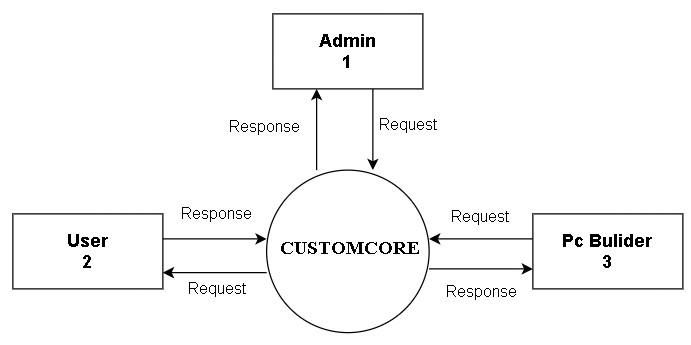
#### 4.5.2.4 DATA STORE

A data store represents logical files. Each data store is connected to a process by means of a dataflow symbol.

#### 4.5.2.5 OUTPUT

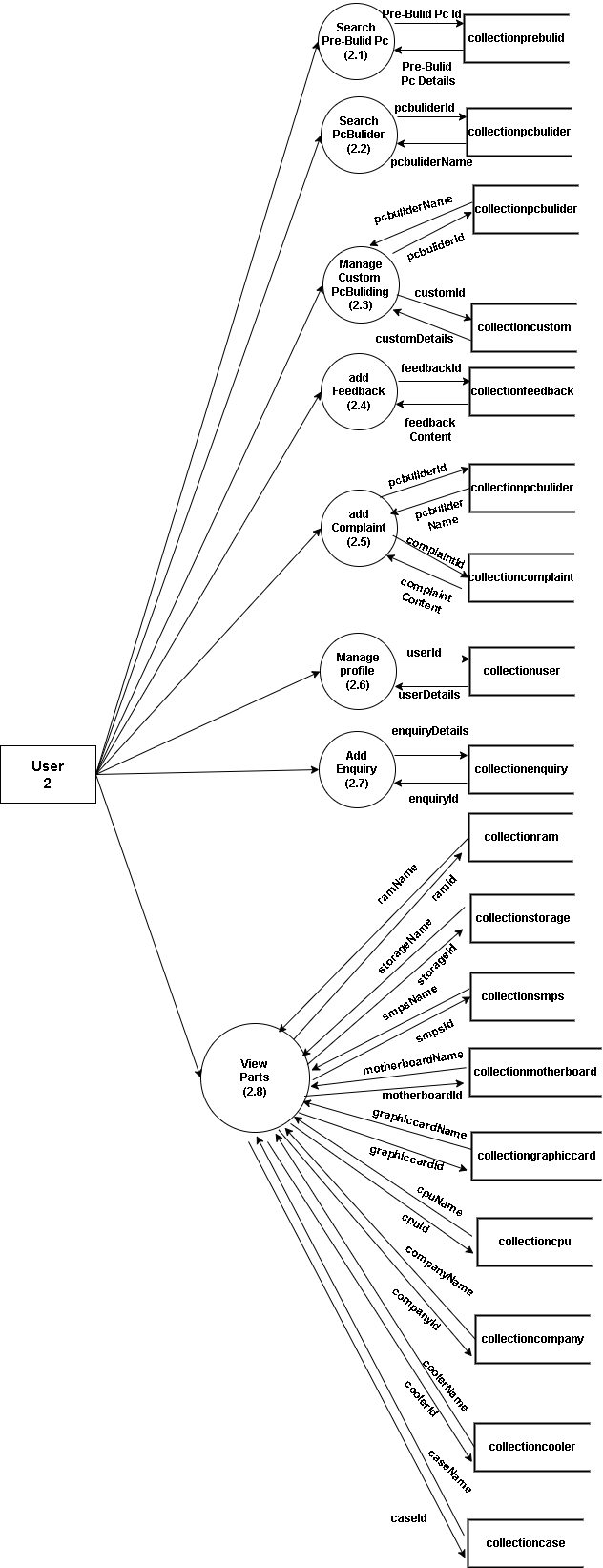
The output symbol is used when a hard copy is produced.

**4.5.3 Level-0 <CONTEXT LEVEL>**

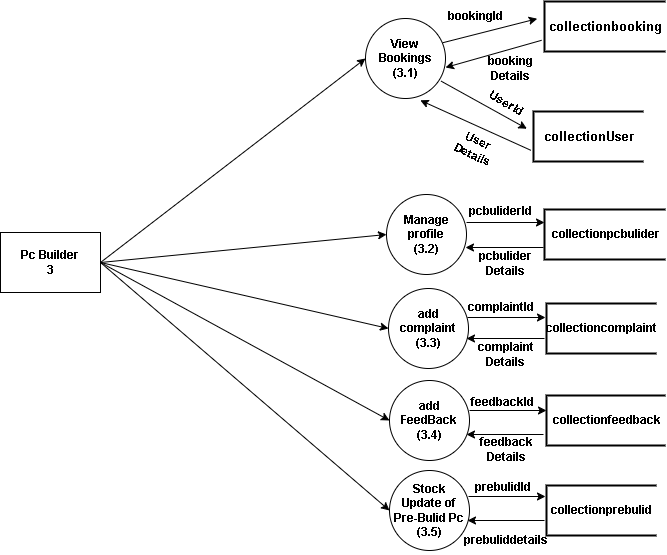


##### 4.5.4 Level 1 <ADMIN MODULE >

**4.5.5 LEVEL 1 <USER MODULE>**



##### 4.5.6 Level -1 < PC BULIDER MODULE>



# CHAPTER 5

# CODING & TESTING

## 5.1 CODING

Coding is the process of translation of the design specification into a programming language that is ultimately transformed into machine executable instructions. Coding phase begin after the design phase is completed. The output of the design phase is the detailed design document. It is the input of the coding phase.

The aim of the coding phase is to produce simple and clean programs. All the syntax and format errors of the program must be detected and resolved. Coding produces modules that must be unit tested. Modules are written in programming language. The following are some of the general coding principles are keep the coding simple, Keep the module short, Code should be legible, and Add comments

In the coding phase, the output document is the code itself after the source code is generated. For the project “CUSTOMCORE”, MERN Stack is used as the programming language. Using appropriate comments did internal documentation of programs. Meaningful names were given to all the variables used in the program

## 5.2 TESTING

System testing is a critical aspect of Software Quality Assurance and represents the ultimate review of specification, design and coding. Testing is a process of executing a program with the intent of finding an error. A good test is one that has a probability of finding an as yet undiscovered error. The purpose of testing is to identify and correct bugs in the developed system. Nothing is complete without testing. Testing is the vital to the success of the system in the code testing the logic of the developed system is tested. For every module of the program is executed to find an error. To perform specification test, the examination of the specifications starting what the program should do and how it should perform under various conditions. Unit testing focuses first on the modules in the proposed system to locate errors. This enables from the interaction between modules are initially avoided. In unit testing step each module has to be checked separately.

Testing and validation are the most important steps after the implementation of the developed system. The system testing is performed to ensure that there are no errors in the implemented system. The software must be executed several times in order to find out the errors in the different modules of the system. Validation refers to the process of using the new software for the developed system in a live environment i.e., new software inside the organization, in order to find out the errors. The validation phase reveals the failures and the bugs in the developed system. It will become to know about the practical difficulties the system faces when operated in the true environment. By testing the code of the implemented software, the logic of the program can be examined.

## 5.3 TESTING METHODS

### 5.3.1 UNITTESTING

A Unit corresponds to a screen/form in the package. Unit testing focuses on the verification of the corresponding class/screen. This testing includes testing of control paths, interfaces, local data structures, logical decisions, boundary conditions and error handling.

### 5.3.2 INTEGRATION AND SYSTEMTESTING

Integration testing is used to verify the combining of the software modules. Integration testing addresses the issue associated with the dual problem of verification and program construction. System testing is used to verify, whether the developed system meets the requirements.

### 5.3.3 USER ACCEPTANCE TESTING

User Acceptance Testing (UAT) is a process to obtain confirmation that a system meets mutually agreed-upon requirements. A Subject Matter Expert (SME), preferably the owner or client of the object under test, provides such confirmation after trial or review. In software development, UAT is one of the final stages of a project and often occurs before a client or customer accepts the new system.

Users of the system perform these tests; which developers derive from the client’s contract or the user requirements specification. Test-designers draw up formal tests and devise a range of security levels. Ideally the designer of the user acceptance tests should not be the creator of the formal integration and system test cases for the same system. The UAT acts as a final verification of the required business function and proper functioning of the system, emulating real-world usage conditions on behalf of the paying client or a specific large customer. If the software works as intended and without issues during normal use, one can reasonably extrapolate the same level of stability in production.

User tests, which are usually performed by clients or end-users, do not normally focus on identifying simple problems such as spelling errors and cosmetic problems, nor showstopper defects, such as software crashes; testers and developers previously identify and fix these issues during earlier unit testing, integration testing, and system testing phases.

The results of these tests give confidence to the clients as to how the system will perform in production. There may also be legal or contractual requirements for acceptance of the system.

### 5.3.3 VALIDATION TESTING

Quality assurance process carried out before the software is ready for release is known as validation testing. The validation testing goal is to validate and be confident about the software product or system, that it full fills the requirements given by the customer. The acceptance of the software from the end customer is also a part of validation testing.

Validation testing answers the question, “Are you building the right software system”. Another question, which the entire process of validation testing in software engineering answers is, “Is the deliverable fit for purpose”. In other words, “does the software system provide the right solution to the problem?” Therefore, often the testing activities are introduced early in the software development life cycle. The two major areas, when validation testing should take place are in the early stages of software development and towards the end, when the product is ready for release. In other words, it is acceptance testing which is a part of validation testing.

### 5.3.5 BLACK BOX TESTING

The test cases are generated based upon the requirements or specification of the program or module. This is also called functional testing. The internals of the module are not considered for selection of test cases.

### 5.3.6 WHITE BOX TESTING

Here, the test cases are generated based on the actual code of the program or module to be tested. This testing is also called glass box testing and structural testing. Three methods used for this are equivalence class partitioning, boundary value analysis and cause effect graphing.

## 5.4 SAMPLE TEST CASES

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Screen** | **Test Field** | **Test Data** | **Expected**  **Result** | **Actual Result** | **Accept/ Reject** | |
| 1 | Login | Email | Ajin | Enter to the homepage | Do not enter into the homepage | Reject | |
| 2 | Login | Email | ajin@g mail.com | Enter to the homepage | Enter to the homepage | Accept | |
| 3 | user reg | Name | Anjana12 | Expect to enter the name | Name field contains only characters | Reject | |
| 4 | user reg | Name | Anjana | Expect to enter the name | The name can be entered | Accept | |
| 5 | user reg | Contact number | 994705554 | Display the msg box for 10 digits | Allow digit less than 10 | Reject | |
| 6 | user reg | Contact number | 9947055541 | No error message | Allow to enter value to the next  field | Accept | |
| 7 | user reg | Email | Abx.@g mail.co m | Expect to enter the email | Email have a definite format | Reject | |
| 8 | user reg | Email | abc@gm ail.com | Expect to enter the email | Email format is correct | Accept |
| 11 | user reg | password | Bessy | Except to make password for the parent | Password contain  Characters& numbers | Reject |
| 12 | user reg | password | Password1234 | Except to make password for the parent | Password is accepted and is unique | Accept |
| 13 | Pc builder reg | Name | AMDS345 | Expect to enter the name | Name contains only characters | Reject |
| 14 | Pc builder reg | Name | AMDS | Expect to enter the name | Name is entered | Accept |
| 15 | Pc builder reg | Contact number | 9446139  1041 | Expect to enter the contact number | Phone number has at most 10 characters | Reject |
| 16 | Pc builder reg | Contact number | 9446139  104 | Expect to enter the contact number | Phone number is valid and can be entered | Accept |

# CHAPTER 6

# IMPLEMENTATION & MAINTENANCE

## 6.1 INTRODUCTION

The implementation is the final state and it is an important phase. It involves the individual programming; system testing, user training and the operational running of developed proposed system that constitutes the application subsystems. A major task of preparing for implementation is education of users, which should really have been taken place much earlier in the project when they were being involved in the investigation and design work. During the implementation phase system actually takes physical shape. In order to develop a system implemented planning is very essential.

## 6.2 IMPLEMENTATION

The implementation phase of the software development is concerned with translating design specification into source code. The user tests the developed system and changes are made according to their needs. Our system has been successfully implemented. Before implementation several tests have been conducted to ensure that no errors are encountered during the operation. The implementation phase ends with an evaluation of the system after placing into the operation for a period of time.

The process of putting the developed system in actual use is called system implementation. This includes all those activities that take place to convert from old system to new system. The system can be implemented only after testing is done and is found to be working to specifications. The implementation stage is a systems project in its own right. The implementation stage involves following tasks:

* Careful planning.
* Investigation of system and constraints.
* Design of method to achieve change over.
* Evaluation of the changeover method.

## 6.3 MAINTENANCE

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes. Maintenance is the ease with which a program can be corrected if any error is encountered, adapted if its environment changes or enhanced if the customer desires a change in requirement. Maintenance follows conversation to extend that changes are necessary to maintain satisfactory operations relative to changes in the user’s environment. Maintenance often includes minor enhancements or corrections to problems that surface in the system’s operation. Maintenance is also done based on fixing the problems reported, changing the interface with other software or hardware enhancing the software.

# CHAPTER 7

# CONCLUSION

Almost every project is subject to change based on client requirements. Since this system is designed to be adaptable for each client, there is always scope for further enhancement. The system and architecture of CUSTOMCORE are highly compatible, allowing new modules to be added without much difficulty.

The software is developed using the MERN stack, which ensures high reliability and compatibility with various environments. The application provides better extensibility and flexibility for future enhancements. Any additional requirements can be incorporated without affecting the system's core functionality. It is a user-friendly platform that is easy and convenient to use.The system is fully operational and has been tested to ensure accurate data entry and report generation. During development, coding standards were followed to facilitate easy maintainability and extensibility.Although the new system enhances operational efficiency, there is room for further improvements. The system has been designed with provisions for future modifications and updates. Any future enhancements will not disrupt the normal functioning of the platform.The developed system is highly interactive and coded to maximize user-friendliness while allowing flexibility for future developments. CUSTOMCORE automates essential tasks in PC building, making the process seamless for users. The system can be integrated with existing solutions, and additional advanced features can be implemented as needed. This adaptability is one of the key strengths of the proposed system.

# CHAPTER 8

# FUTURE ENHANCEMENT

Due to time constraints, the design aspect has not been made highly attractive. Future enhancements can focus on improving the user interface to create a more visually appealing experience. Additional forms can be introduced to facilitate better retrieval of feedback details. Various other features can also be incorporated to enhance the overall usability and functionality of the project, ensuring a more seamless and efficient user experience.

Moreover, cloud-based storage can be introduced to allow users to save and access their builds across multiple devices. Integration with leading e-commerce platforms can enable seamless purchasing and price comparisons. Enhanced security measures, such as two-factor authentication and encrypted user data storage, can also be incorporated to ensure data privacy and system integrity.

Additionally, community engagement features can be expanded to include live chat support, expert consultations, and virtual PC-building simulations. Regular software updates and user feedback mechanisms can further refine the platform, ensuring it evolves to meet user needs and technological advancements over time.

**CHAPTER 9**

# BIBLIOGRAPHY

**Book References**:

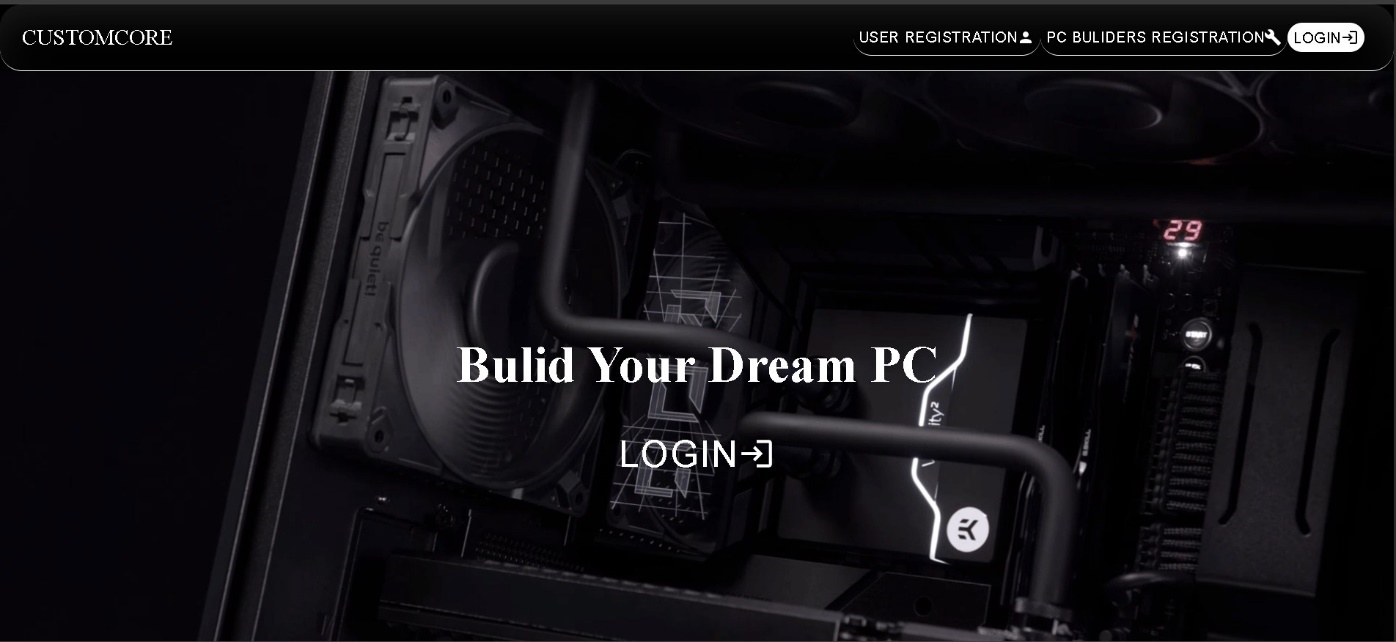
1. JefrryLwhiten Lonnie D Dently, “System Analysis and Design”, Tata mcGraw publication Fourth Edition, 2002.
2. Korth F Henry, Silberschatz, Abraham,” Database System Concepts”, McGraw Hill.
3. Rajib Mall, “Fundamentals of software engineering”, Prentice, Hall of India Limited, 7th Edition, 2000.
4. Michael Bleha, James Rambaugh, “Object-Oriented Modelling& Design with UML”, Pearson, Second Edition, 2005.
5. James R Groff and Paul N Weinberg- “The Complete Reference SQL”, Tata McGraw Hill, Second Edition, 2003.
6. S A Kelkar, “Software Project Management”, Prentice Hall of India.
7. https://www.geeksforgeeks.org/mern-stack/
8. http://www.slideshare.net/DivyaJain31/public-distribution-system2019

**CHAPTER 10**

**APPENDIX**

# 10.1 SCREENSHOTS

### 10.1.1 Landing page



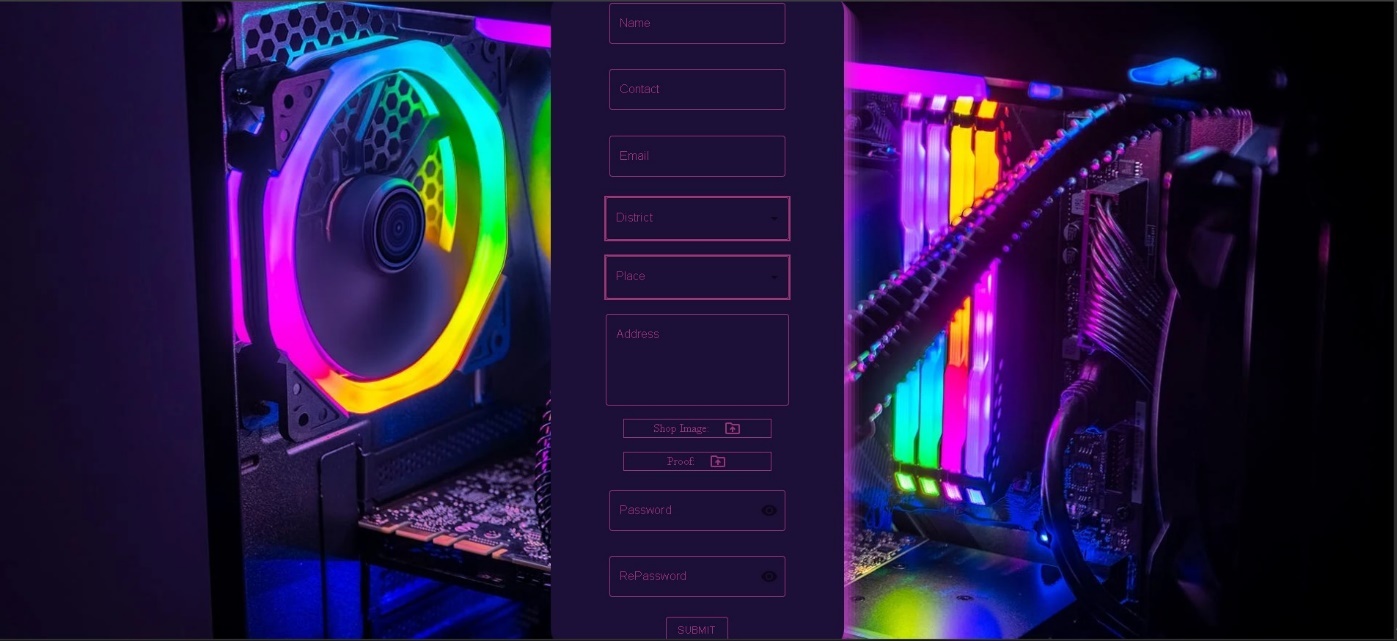
### 10.1.2 User Registration page

A computer with a keyboard and jars of paint

AI-generated content may be incorrect.

## 

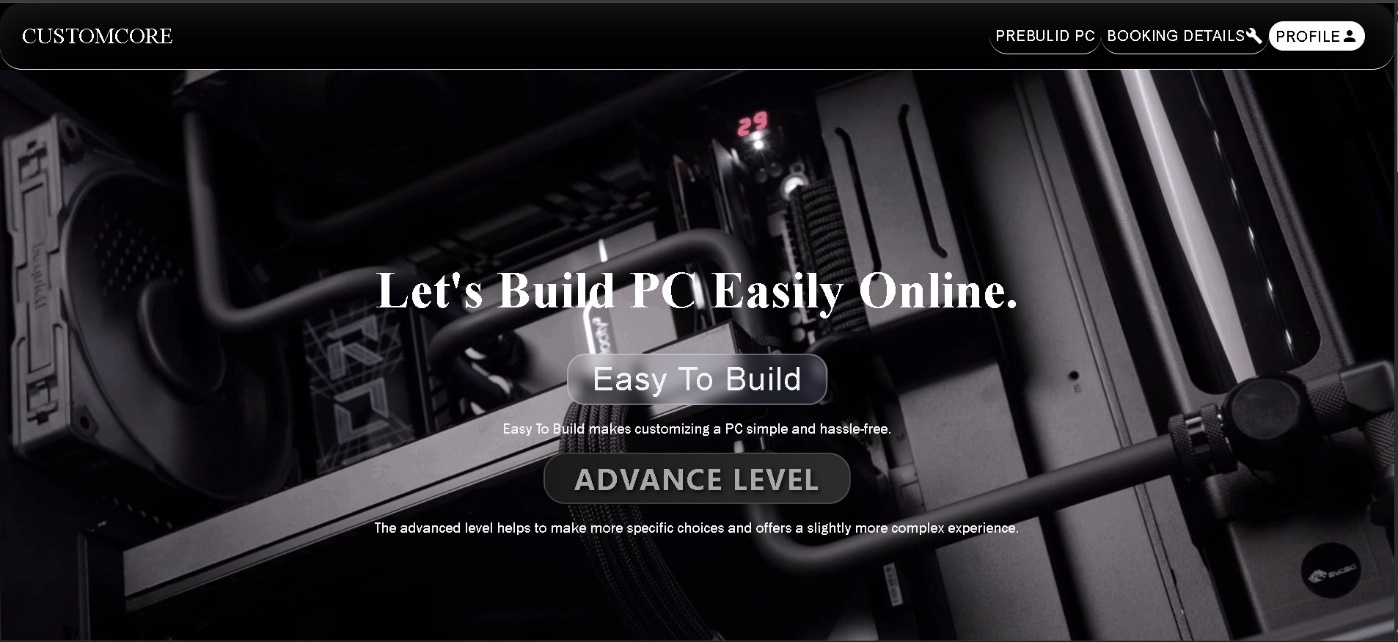
## 10.1.3 Pc Bulider Registration page



## 10.1.4 Login page

## A computer with a screen on it AI-generated content may be incorrect.

### 10.1.5 User Home Page



### 10.1.6 Pc Bulider Home Page

**A person working on a computer

AI-generated content may be incorrect.**

**10.2 SAMPLE CODE**

import React, { useEffect, useState } from 'react'

import Styles from './BuliderReg.module.css'

import { Box, Button, FormControl, IconButton, InputAdornment, InputLabel, MenuItem, OutlinedInput, Select, TextField } from '@mui/material'

import { Place, Visibility, VisibilityOff } from '@mui/icons-material';

import { DriveFolderUploadOutlined } from '@mui/icons-material'

import axios from 'axios';

const BuliderReg = () => {

  const [showPassword, setShowPassword] = React.useState(false);

  const handleClickShowPassword = () => setShowPassword((show) => !show);

  const handleMouseDownPassword = (event) => {

    event.preventDefault();

  };

  const handleMouseUpPassword = (event) => {

    event.preventDefault();

  };

  const [district, setdistrict] = React.useState('');

  const handleChange = (event) => {

    setPlace(event.target.value);

  };

  const fetchDistrict = () => {

    axios.get("http://localhost:5000/collectionDistrict").then((response) => {

      console.log(response.data);

      setDistrictArray(response.data.district)

    })

  }

  const fetchPlace = (e) => {

    const district = e.target.value

    axios.get(`http://localhost:5000/collectionPlaceByIdAll/${district}`).then((response) => {

      console.log(response.data);

      setPlaceArray(response.data)

    })

  }

    const [DistrictArray, setDistrictArray] = useState([])

    const [PlaceArray, setPlaceArray] = useState([])

    const [buliderName, setbuliderName] = useState("")

      const [buliderContact, setbuliderContact] = useState("")

      const [buliderEmail, setbuliderEmail] = useState("")

      const [buliderAddress, setbuliderAddress] = useState("")

      const [buliderPassword, setbuliderPassword] = useState("")

      const [Place, setPlace] = useState("")

      const [image, setimage] = useState([])

      const [proof, setproof] = useState([])

      const [Message, setMessage] = useState("")

  const handleSubmit = async () => {

    const formDataToSend = new FormData();

    formDataToSend.append('name', buliderName);

    formDataToSend.append('email', buliderEmail);

    formDataToSend.append('address', buliderAddress);

    formDataToSend.append('contact', buliderContact);

    formDataToSend.append('place', Place);

    formDataToSend.append('password', buliderPassword);

    console.log(proof)

    console.log(image)

    if (image) formDataToSend.append('image', image);

    if (proof) formDataToSend.append('proof', proof);

    try {

      axios.post('http://localhost:5000/collectionBuliderreg', formDataToSend).then((response) => {

        setMessage('Registration successful!');

        console.log('Registration successful:', response.data);

        alert("Registration successful");

      })

    } catch (error) {

      // setMessage('Error registering. Please try again.');

      console.error('Error registering:', error);

    }

  };

 useEffect(() => {

    fetchDistrict();

  }, [])

  return (

    <div className={Styles.body}>

      <div className={Styles.Card}>

        <div className={Styles.username}>

          <FormControl sx={{ m: 1, width: '30ch' }} variant="outlined">

            <InputLabel htmlFor="outlined-adornment-password" sx={{ color: '#9d3579' }}>Name</InputLabel>

            <OutlinedInput

              id="outlined-adornment-password"

              label="User Name" sx={{

                '& .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&:hover .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&.Mui-focused .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                input: {

                  color: '#9d3579',

                },

              }} onChange={(e) => setbuliderName(e.target.value)}/>

          </FormControl>

        </div>

        <div className={Styles.contact}>

          <FormControl sx={{ m: 1, width: '30ch' }} variant="outlined">

            <InputLabel htmlFor="outlined-adornment-password" sx={{ color: '#9d3579' }}>Contact</InputLabel>

            <OutlinedInput

              id="outlined-adornment-password"

              label="contact" sx={{

                '& .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&:hover .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&.Mui-focused .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                input: {

                  color: '#9d3579',

                },

              }} onChange={(e) => setbuliderContact(e.target.value)}/>

          </FormControl>

        </div>

        <div className={Styles.email}>

          <FormControl sx={{ m: 1, width: '30ch' }} variant="outlined">

            <InputLabel htmlFor="outlined-adornment-password" sx={{ color: '#9d3579' }}>Email</InputLabel>

            <OutlinedInput

              id="outlined-adornment-password"

              label="email" sx={{

                '& .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&:hover .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&.Mui-focused .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                input: {

                  color: '#9d3579',

                },

              }} onChange={(e) => setbuliderEmail(e.target.value)}/>

          </FormControl>

        </div>

        <div className={Styles.district}>

          <Box sx={{ minWidth: 250, color: '#9d3579', backgroundColor: '#1c0f38', border: '2px solid #9d3579' }}>

            <FormControl fullWidth>

              <InputLabel id="demo-simple-select-label" sx={{ color: '#9d3579' }}>District</InputLabel>

              <Select

                labelId="demo-simple-select-label"

                id="demo-simple-select"

                // value={district}

                label="District"

                onChange={fetchPlace}

                sx={{ color: '#9d3579', backgroundColor: '#1c0f38', border: '0.1px solid #9d3579' }}

              >

                {DistrictArray && DistrictArray.map((District, index) => (

                  <MenuItem key={index} value={District.\_id} sx={{ color: 'black' }}>

                    {District.districtName}

                  </MenuItem>

                ))}

              </Select>

            </FormControl>

          </Box>

        </div>

        <div className={Styles.district}>

          <Box sx={{ minWidth: 250, color: '#9d3579', backgroundColor: '#1c0f38', border: '2px solid #9d3579' }}>

            <FormControl fullWidth>

              <InputLabel id="demo-simple-select-label" sx={{ color: '#9d3579' }}>Place</InputLabel>

              <Select

                labelId="demo-simple-select-label"

                id="demo-simple-select"

                value={Place}

                label="District"

                onChange={handleChange}

                sx={{ color: '#9d3579', backgroundColor: '#1c0f38', border: '1px solid #9d3579' }}

              >

                {PlaceArray?.map((Place, index) => (

                  <MenuItem key={index} value={Place.\_id} sx={{ color: 'black' }}>

                    {Place.placeName}

                  </MenuItem>

                ))}

              </Select>

            </FormControl>

          </Box>

        </div>

        <div className={Styles.address}>

          <TextField

            id="outlined-multiline-static"

            label="Address"

            multiline

            rows={4}

            sx={{

              width: '250px',

              '& .MuiOutlinedInput-root': {

                '& fieldset': {

                  borderColor: '#9d3579',

                },

                '&:hover fieldset': {

                  borderColor: '#9d3579',

                },

                '&.Mui-focused fieldset': {

                  borderColor: '#9d3579',

                },

                color: '#9d3579'

              },

              '& .MuiInputLabel-root': {

                color: '#9d3579',

              },

              '& .MuiInputLabel-root.Mui-focused': {

                color: '#9d3579',

              },

            }}

            onChange={(e) => setbuliderAddress(e.target.value)}

          />

        </div>

        <div className={Styles.forminput}>

          <label className={Styles.label} htmlFor='fileimage'>Shop Image: <DriveFolderUploadOutlined className={Styles.icon} /></label>

          <input type="file" id='fileimage' onChange={e => setimage(e.target.files[0])} style={{ display: "none" }} />

        </div>

        <div className={Styles.forminput}>

          <label className={Styles.label} htmlFor='fileproof'>Proof: <DriveFolderUploadOutlined className={Styles.icon} /></label>

          <input type="file" id='fileproof' onChange={e => setproof(e.target.files[0])} style={{ display: "none" }} />

        </div>

        <div className={Styles.password}>

          <FormControl sx={{ m: 1, width: '30ch' }} variant="outlined">

            <InputLabel htmlFor="outlined-adornment-password" sx={{ color: '#9d3579' }}>Password</InputLabel>

            <OutlinedInput

              id="outlined-adornment-password"

              sx={{

                '& .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&:hover .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&.Mui-focused .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                input: {

                  color: '#9d3579',

                },

              }}

              type={showPassword ? 'text' : 'password'}

              endAdornment={

                <InputAdornment position="end">

                  <IconButton

                    aria-label={

                      showPassword ? 'hide the password' : 'display the password'

                    }

                    onClick={handleClickShowPassword}

                    onMouseDown={handleMouseDownPassword}

                    onMouseUp={handleMouseUpPassword}

                    edge="end"

                  >

                    {showPassword ? <VisibilityOff /> : <Visibility />}

                  </IconButton>

                </InputAdornment>

              }

              label="Password"

              onChange={(e) => setbuliderPassword(e.target.value)}

            />

          </FormControl>

        </div>

        <div className={Styles.Repassword}>

          <FormControl sx={{ m: 1, width: '30ch' }} variant="outlined">

            <InputLabel htmlFor="outlined-adornment-password" sx={{ color: '#9d3579' }}>RePassword</InputLabel>

            <OutlinedInput

              id="outlined-adornment-password"

              sx={{

                '& .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&:hover .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                '&.Mui-focused .MuiOutlinedInput-notchedOutline': { borderColor: '#9d3579', },

                input: {

                  color: '#9d3579',

                },

              }}

              type={showPassword ? 'text' : 'password'}

              endAdornment={

                <InputAdornment position="end">

                  <IconButton

                    aria-label={

                      showPassword ? 'hide the password' : 'display the password'

                    }

                    onClick={handleClickShowPassword}

                    onMouseDown={handleMouseDownPassword}

                    onMouseUp={handleMouseUpPassword}

                    edge="end"

                  >

                    {showPassword ? <VisibilityOff /> : <Visibility />}

                  </IconButton>

                </InputAdornment>

              }

              label="Password"

            />

          </FormControl>

        </div>

        <div className={Styles.button}>

          <Button variant="outlined" sx={{ color: '#9d3579', borderColor: '#9d3579' }} onClick={handleSubmit}>Submit</Button>

        </div>

      </div>

    </div>

  )

}

export default BuliderReg