**FULL STACK IN MERN**

AN INTERNSHIP REPORT

***Submitted by***

**PARASURAM .T (512221205012)  
PARASURAMAN.K (512221205013)**

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

***Of***

**BACHELOR OF TECHNOLOGY**

***in***

**INFORMATION TECHNOLOGY**

**SKP ENGINEERING COLLEGE,**

**TIRUVANNAMALAI - 606611**

****

**ANNA UNIVERSITY: CHENNAI 600 025**

**NOVEMBER – 2024**

**FULL STACK IN MERN**

AN INTERNSHIP REPORT

***Submitted by***

**PARASURAM .T (512221205012)  
PARASURAMAN.K (512221205013)**

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

***Of***

**BACHELOR OF TECHNOLOGY**

***in***

**INFORMATION TECHNOLOGY**

**SKP ENGINEERING COLLEGE,**

**TIRUVANNAMALAI - 606611**

****

**ANNA UNIVERSITY: CHENNAI 600 025**

**NOVEMBER – 2024**

**ANNA UNIVERSITY: CHENNAI 600 025**

**BONAFIDE CERTIFICATE**

Certified that this Summer Internship Report **“FULL STACK IN MERN”** is the bonafide work of **“PARASURAM .T - (512221205012), PARASURAMAN. K - (512221205013)”**who carried out the Summer Internship work under my supervision.

----------------------------------------- --------------------------------------

**SIGNATURE** **SIGNATURE**

**Dr. V. Raji, M.E., Ph.D.,**  **Mr. R. Kaudilyar, M.E.,**HEAD OF THE DEPARTMENT, SUPERVISOR,DEPARTMENT OF IT, DEPARTMENT OF IT ,

SKP ENGINEERING COLLEGE, SKP ENGINEERING COLLEGE,

TIRUVANNAMALAI. TIRUVANNAMALAI.

**Submitted for the Summer Internship Examination held on …………………**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

We are very thankful to the management of **SKP Engineering College,**

**Tiruvannamalai** for giving us this wonderful opportunity to study in this college and utilize all the facilities to the fullest.

Our sincere thanks to the lord blessing on us for the successful completion of our internship . We proudly thank **Mr. K. KARUNANITHI, B.E., MBA,** Chairman, **Mr. K. V. ARANGASAMY,** Joint Secretary and **Mr. R. SAKTHI KRISHNAN, M.L., M.B.A., Ph.D.,** CEO of our college for providing the facility to do the project. We are grateful for our Principal **Dr. S. BASKARAN, M.E, Ph.D.,** for his constant support and encouragement to our internship.

We express our sincere thanks to **Dr. V. RAJI, M.E., Ph.D.,** Head of the Department of Information Technology and Our Class Advisor **Ms. J. NIZU SHAHAIN, M.E.,** for her guidance, timely suggestions and word motivation.

It is a Great pleasure to express our gratitude and thanks towards our internship supervision **Mr. R. KAUDILYAR, M.E.,** for his uninterruptable suggestions and words of improvements regarding this internship, which played a major role in guiding us in our track.

We also extend our thanks to our Class Advisor and every member of the faculty who provided valuable academic guidance, and their co-operation to do this successful internship. Finally, we would like to thank our parents, relatives and friends for their encouragement and enthusiastic co-operation.



****

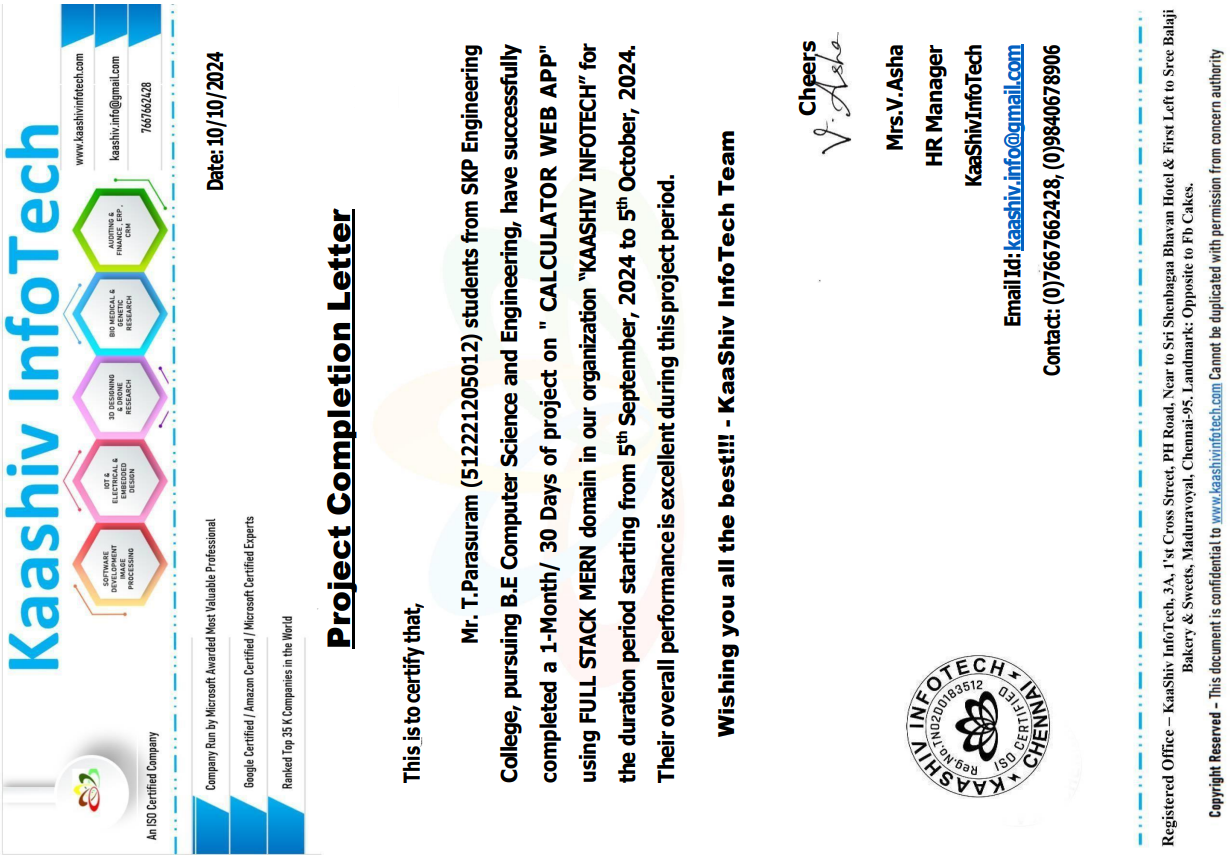
* **Company Run by Microsoft Awarded Most Valuable Professional**
* **Google Certified / Amazon Certified / Microsoft Certified Experts**
* **Ranked Top 35 K Companies in the World**

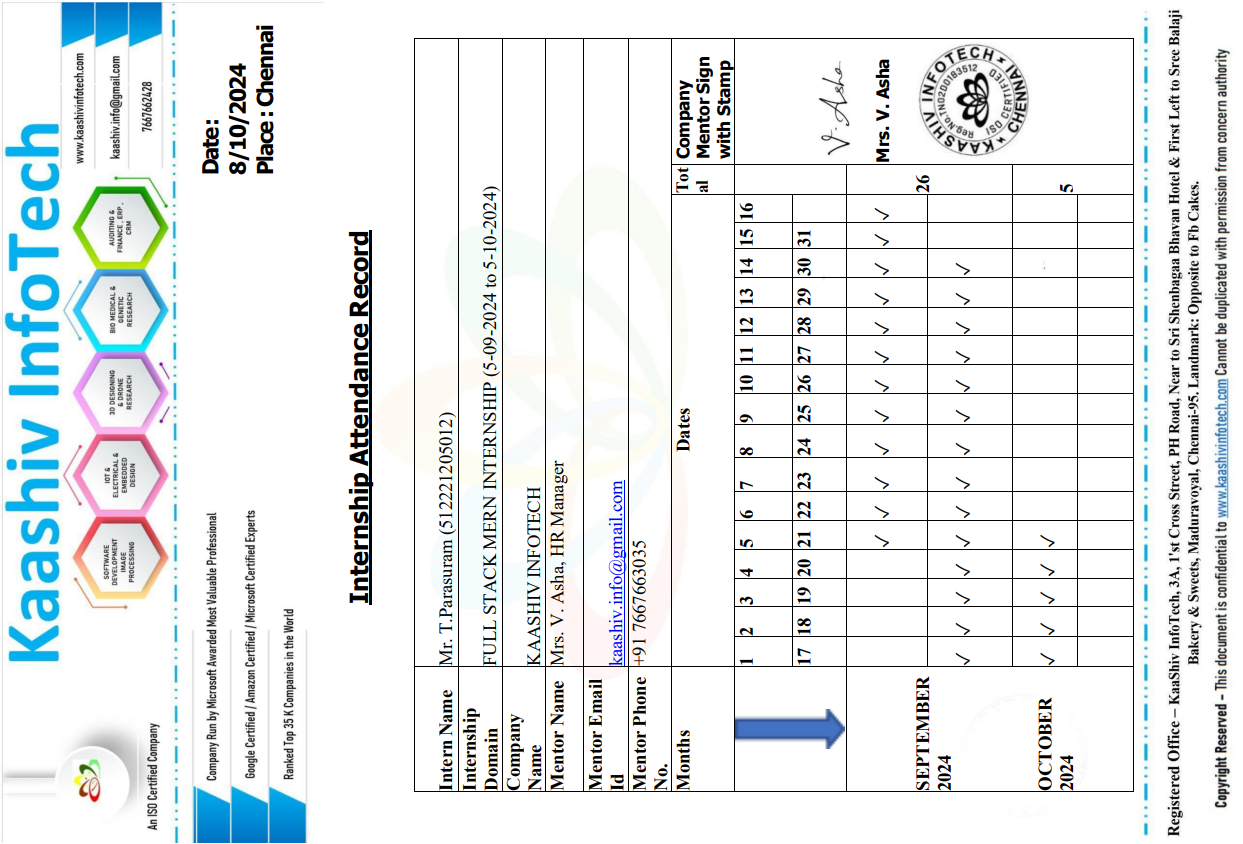
****

**Registered Office -** KaaShiv InfoTech, 3A, 1'st Cross Street, PH Road, Near to Sri Shenbagaa Bhavan Hotel & First Left to Sree Balaji Bakery & Sweets, Maduravoyal, Chennai-95. Landmark: Opposite to Fb Cakes.

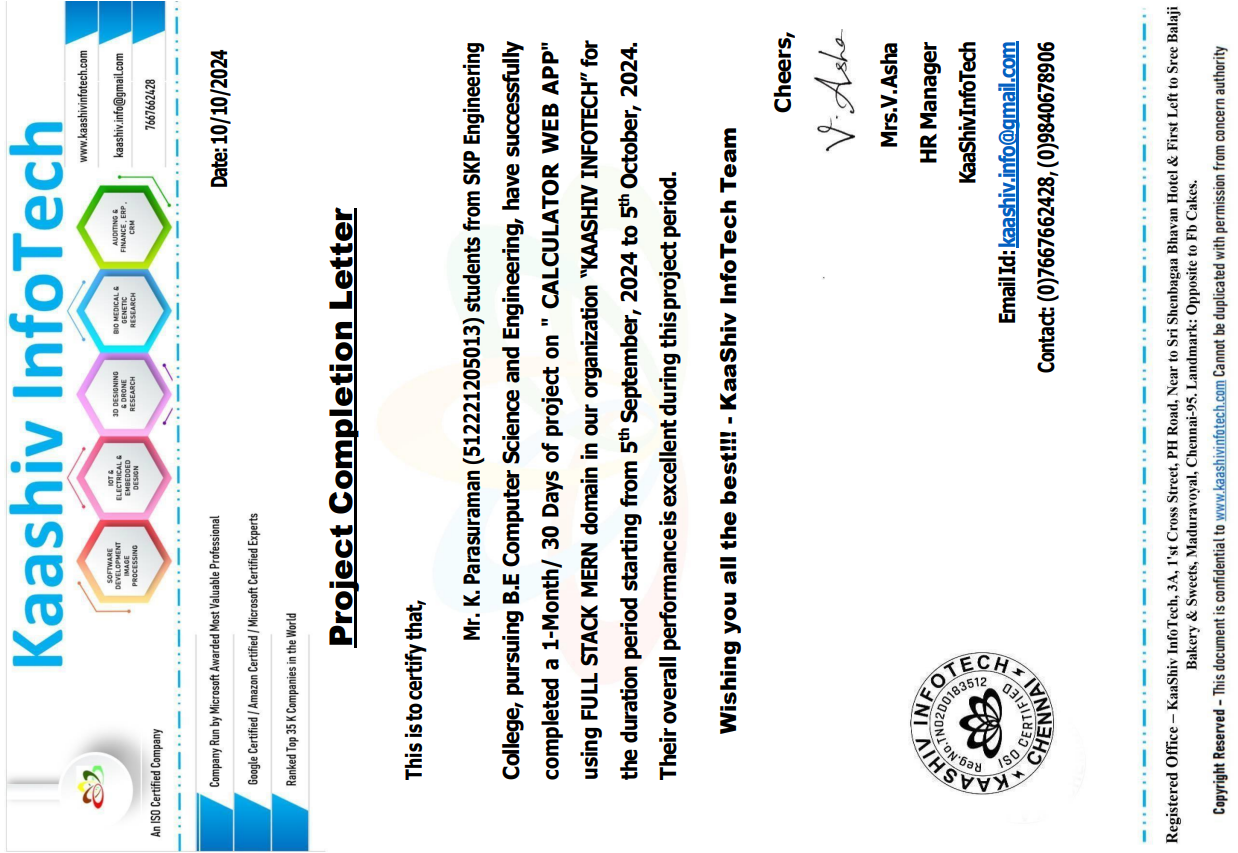
**Copyright Reserved -** This document is confidential to www.kaashivinfotech.com Cannot be duplicated with permission from concern authority

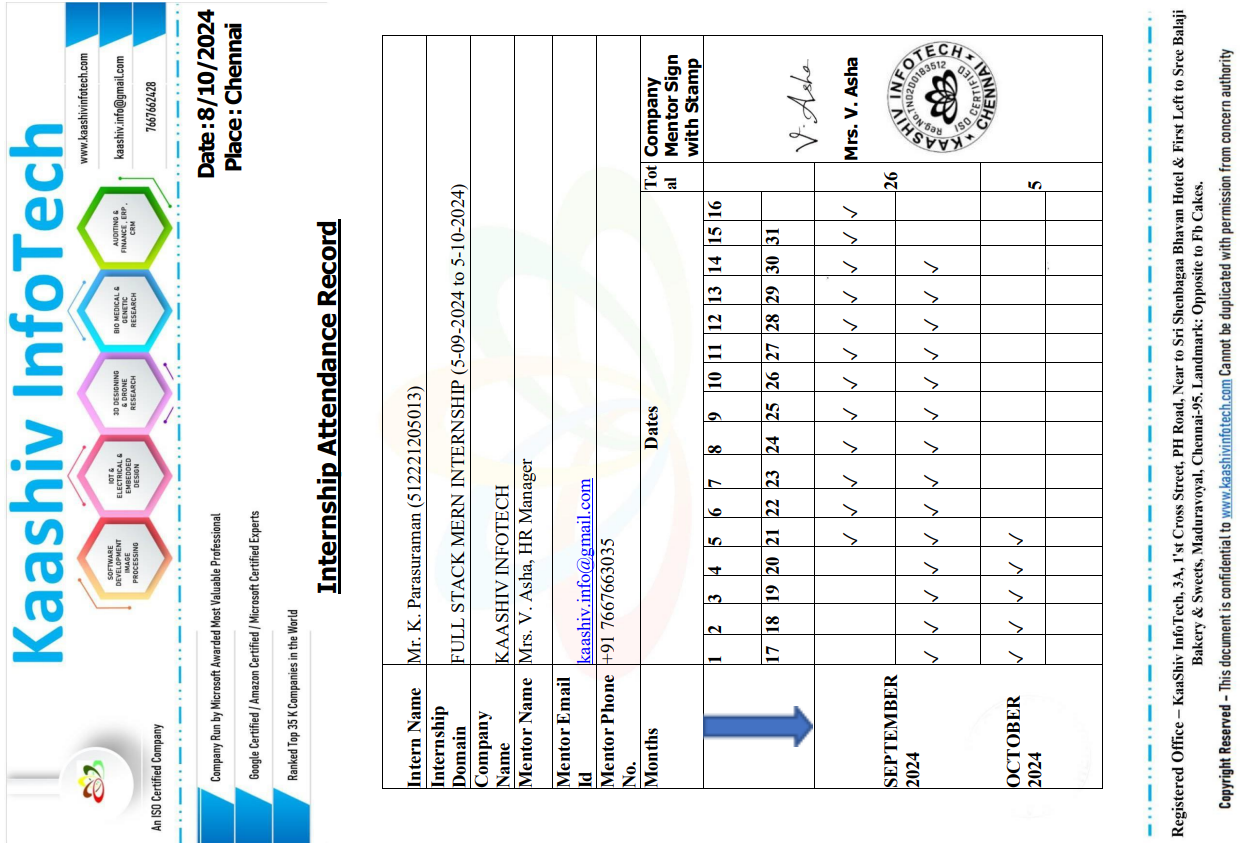
****

****

****

****

****

****

**ABSTRACT**

This report presents the development of a Calculator App using the MERN stack during an internship. The app provides a responsive, user-friendly interface for basic and scientific calculations. Key technologies include React.js for the front-end, Node.js for back-end processes, and GitHub Pages for deployment.

The project addresses limitations of existing solutions by offering improved usability, real-time computation, and cross-device compatibility. A modular architecture ensures scalability and ease of maintenance. Testing confirmed the app's reliability and accuracy.

The report concludes with the app’s accomplishments and suggests future enhancements like advanced features, multilingual support, and mobile app development, highlighting its practical application of MERN stack concepts.

|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENTS** |  |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT**  **LIST OF FIGURES**  **LIST OF ABBREVIATIONS** | **xi**  **xvi**  **xvii** |
| **1.** | **INTRODUCTION**  1.1 GENERAL  1.2 OBJECTIVE  1.3 SUMMARY | **1**  **1**  **2**  **3** |
| **2.** | **LITERATURE SURVEY**  2.1 GENERAL  2.2 LITERATURE REVIEW  2.3 SUMMARY | **5**  **5**  **6**  **9** |
| **3.** | **SYSTEM ANALYSIS**  3.1 GENERAL  3.2 EXISTING SYSTEM  3.2.1 Existing System Architecture  3.2.2 Limitations of Existing System  3.3 PROPOSED SYSTEM  3.3.1 Proposed System Architecture  3.3.2 Advantages Of Proposed System  3.4 SUMMARY | **10**  **10**  **11**  **11**  **12**  **14**  **14**  **15**  **17** |
| **4.** | **SYSTEM DESIGN AND IMPLEMENTATION**  4.1 GENERAL  4.2 LIST OF MODULES  4.3 MODULE DESCRIPTION  4.3.1 Data Acquisition And Preprocessing  4.3.2 Model Training And Evaluation  4.3.3 Prediction And Explanation Generation  4.3.4 Application Development And Deployment  4.4 SUMMARY | **18**  **18**  **18**  **19**  **19**  **20**  **20**  **21**  **22** |
| **5.** | **SYSTEM REQUIREMENTS**  5.1 GENERAL  5.2 SYSTEM REQUIREMENTS  5.2.1 Hardware Requirements  5.2.2 Software Requirements  5.3 TECHNICAL SPECIFICATIONS  5.4 SUMMARY | **23**  **23**  **23**  **23**  **24**  **25**  **26** |
| **6.** | **SYSTEM ARCHITECTURE**  6.1 GENERAL  6.2 ARCHITECTURE DIAGRAM  6.3 COMPONENTS OF THE ARCHITECTURE  6.4 SUMMARY | **28**  **28**  **28**  **29**  **30** |
| **7.** | **SYSTEM IMPLEMENTATION**  7.1 CODING  7.2 OUTPUT | **31**  **31**  **44** |
| **8.** | **SYSTEM SECURITY**  8.1 GENERAL  8.2 SECURITY MEASURES  8.3 SECURITY IN FUTURE UPGRADES  8.4 SUMMARY | **45**  **45**  **45**  **47**  **47** |
| **9.** | **SYSTEM TESTING**  9.1 WHITE BOX TESTING  9.2 BLACK BOX TESTING  9.3 TEST CASES  9.4 TESTING TOOLS  9.5 TESTING ENVIRONMENT  9.6 SUMMARY | **48**  **48**  **49**  **50**  **50**  **50**  **51** |
| **10.** | **CONCLUSION AND FUTURE ENHANCEMENTS** | **52** |
| **11.** | **COMPANY PROFILE**  11.1 MISSION  11.2 COMPANY VALUES  11.3 COMPANY INFRASTRUCTURE  11.4 COMPANY SERVICES | **54**  **55**  **56**  **57**  **58** |
|  |  |  |
|  | **APPENDIX 1 SCREENSHOTS** | **60** |
|  | [**APPENDIX 2 SAMPLE CODING**](#_1y810tw) | **61** |
|  | **REFERENCES** | **65** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE  NO** | **TITLE** | **PAGE NO** |
| **1.1** | MERN stack | **1** |
| **4 .1** | Calculator Apps Sequence Diagrams | **19** |
| **4.2** | Calculator App Application Development And Deployment | **22** |
| **5.1** | Calculator App Use Case Diagram | **27** |
| **6.1** | Calculator App Architecture Diagram | **28** |
| **7.1** | Output screenshots | **44** |
| **11.1** | Digital City Matrix | **54** |
| **11.2** | Software Development Tech Stack Overview | **55** |
| **11.3** | Mobile App Development Features Wheel | **56** |
| **11.4** | Converged Infrastructure Components Diagram | **57** |
| **11.5** | Digital Services and Website Solutions | **58** |
| **11.6** | Agile Development Lifecycle Diagram | **59** |
| **A1.1** | Application Screenshot-1 | **60** |
| **A1.2** | Application Screenshot-2 | **60** |

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| **MERN** | MongoDB, Express.js, React.js, Node.js |
| **UI** | User Interface |
| **UX** | User Experience |
| **API** | Application Programming Interface |
| **CSS** | Cascading Style Sheets |
| **HTML** | HyperText Markup Language |
| **DOM** | Document Object Model |
| **NPM** | Node Package Manager |
| **JSON** | JavaScript Object Notation |
| **CI/CD** | Continuous Integration/Continuous Deployment |
| **IDE** | Integrated Development Environment |
| **CRUD** | Create, Read, Update, Delete |
| **SEO** | Search Engine Optimization |
| **AI** | Artificial Intelligence |
| **JWT** | JSON Web Token |
| **HTTP** | HyperText Transfer Protocol |
| **HTTPS** | HyperText Transfer Protocol Secure |
| **SQL** | Structured Query Language |
| **MVC** | Model-View-Controller |
| **CLI** | Command Line Interface |

**CHAPTER 1**

**INTRODUCTION**

**1.1 GENERAL**

The MERN stack, an acronym for MongoDB, Express.js, React, and Node.js, has become a cornerstone for modern web development. This powerful combination of technologies enables developers to create dynamic, full-stack web applications with seamless data flow between the front-end and back-end. MERN is celebrated for its flexibility, scalability, and ability to streamline the development process by using JavaScript as the universal programming language across the stack.

In today’s fast-paced digital era, applications that enhance productivity and provide user-friendly interfaces are highly sought after. Calculators, as fundamental tools for mathematical computations, have been an integral part of technological evolution. From simple arithmetic calculations to complex mathematical operations, calculators serve a wide range of purposes for individuals, students, and professionals alike.

**Figure 1.1 :** MERN stack

The primary focus of this internship project was to develop a Calculator App using the MERN stack. By leveraging the powerful React library for the front-end and efficient Node.js and Express.js frameworks for the back-end, the project aimed to deliver a visually appealing and feature-rich application. The integration of MongoDB ensures data persistence, although this app focuses primarily on its computational logic and user experience.

The Calculator App goes beyond basic mathematical operations, featuring a responsive design, lightweight build, and seamless deployment via GitHub Pages. The app uses popular packages like Bootstrap for styling, expr-eval for arithmetic computations, and React for dynamic component rendering. This project not only demonstrates proficiency in the MERN stack but also showcases the potential of modular development and efficient deployment practices.

By implementing modern web development practices, the project emphasizes the significance of designing robust, user-centric applications. It also reflects the practicality and scalability of the MERN stack in addressing everyday computational needs.

**1.2 OBJECTIVE**

The primary objective of this internship project was to design and develop a **Calculator App** using the MERN stack, showcasing technical proficiency and leveraging modern web development tools. The project aimed to achieve the following specific goals:

* **Implement Core MERN Stack Concepts:**  
  Utilize the MongoDB, Express.js, React, and Node.js technologies to build a full-stack application, highlighting the seamless integration between the front-end and back-end.
* **Create a Functional and User-Friendly Application:**  
  Develop a calculator with a clean, intuitive interface that allows users to perform basic and complex arithmetic operations effortlessly.
* **Explore Dependency Integration:**  
  Incorporate popular dependencies such as **Bootstrap** for responsive design, **expr-eval** for accurate mathematical evaluations, and other tools like **jQuery** and **Font Awesome** to enhance the application’s functionality and aesthetics.
* **Adopt Modular and Scalable Design Principles:**  
  Build the application with a modular architecture, making it easy to maintain, enhance, and scale in the future.
* **Deploy Using Modern Tools:**  
  Deploy the application to GitHub Pages using **gh-pages**, enabling quick and seamless access to the app for users.
* **Enhance Technical Skills and Learning Outcomes:**  
  Deepen understanding of React component-based development, back-end integration with Node.js and Express.js, and the utilization of version control and deployment pipelines.

**1.3 SUMMARY**

This internship project revolved around the development of a **Calculator App** using the MERN stack, with the goal of enhancing practical knowledge and demonstrating core competencies in full-stack web development. The project involved leveraging **React** for building dynamic and interactive user interfaces, **Node.js** and **Express.js** for establishing a robust back-end, and incorporating relevant dependencies to streamline the app’s functionality and design.

Throughout the development process, modern tools and practices were employed to ensure the application was efficient, responsive, and user-friendly. Key features of the app include:

* A visually appealing and intuitive interface designed with **Bootstrap** and **Font Awesome**.
* Accurate mathematical computations powered by the **expr-eval** package.
* Responsive design principles to ensure usability across various devices.
* Seamless deployment through **GitHub Pages**, making the app readily accessible.

The project emphasized the importance of adopting a modular approach, ensuring scalability and ease of future enhancements. Additionally, it provided valuable insights into integrating various tools and libraries, handling dependencies, and deploying applications in a real-world environment.

Overall, the Calculator App project stands as a testament to the power and versatility of the MERN stack in creating practical, real-world applications. It not only fulfilled its objective of providing a functional calculator but also served as a comprehensive learning experience, enhancing both technical skills and problem-solving abilities.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 GENERAL**

A **literature survey** is an essential step in understanding the current state of knowledge within a specific field, identifying trends, gaps, and providing a foundation for the proposed work. In the context of this **Calculator App** project, the literature survey explores existing approaches to calculator applications, examines the technologies used in similar projects, and highlights their strengths and limitations.

Calculator applications have existed for decades, initially in hardware form and later as software applications. Early digital calculators were designed for simple arithmetic operations, but with the advancement of computing power and software development, modern calculators now support a variety of functionalities, from basic arithmetic to complex scientific and graphing calculations.

The evolution of calculator apps on smartphones, web platforms, and desktop systems has demonstrated the increasing demand for user-friendly, responsive, and feature-rich applications. While many calculator applications are developed using native programming languages for mobile devices (like Java for Android or Swift for iOS), web-based calculators have gained popularity due to their cross-platform compatibility, ease of access, and lightweight nature.

The literature also highlights the role of **JavaScript**, **React**, and **Node.js** in modern web development. React’s component-based architecture, Node.js's non-blocking I/O model, and the widespread use of JavaScript in both front-end and back-end development have positioned them as ideal tools for building scalable and efficient web applications. Many developers have adopted the MERN stack to streamline development and deliver high-performance, interactive applications.

Through this literature survey, it becomes evident that there is a significant demand for innovative, web-based calculator apps that can be easily deployed and accessed from multiple platforms. This provides the necessary context for the proposed **Calculator App** project, which aims to address user needs while leveraging modern technologies to create a highly functional and aesthetically pleasing application.

**2.2 LITERATURE REVIEW**

The development of calculator applications has evolved significantly over the years, influenced by advances in both hardware and software technologies. This literature review focuses on the various approaches, frameworks, and tools employed in creating modern calculator applications, with an emphasis on web-based solutions and the technologies that power them.

* **Traditional and Desktop Calculators**
  + Traditional calculators, once reliant on physical buttons and hardware circuits, have evolved into software-based applications. Desktop calculators, such as the default calculator in Windows or macOS, began as simple tools for basic arithmetic operations. These applications often served a limited purpose, with little or no customization options, offering basic functionality such as addition, subtraction, multiplication, and division.
  + More advanced desktop calculators, such as **scientific calculators**, introduced features like trigonometric functions, logarithms, and exponential operations. However, they were still restricted to desktop environments and lacked integration with the broader web ecosystem.
* **Web-Based Calculators**
  + With the growth of the internet, web-based calculators became increasingly popular. These calculators, often built using HTML, CSS, and JavaScript, offered advantages such as accessibility across multiple platforms (Windows, macOS, Linux, etc.) and devices (PCs, tablets, smartphones). They required no installation and could be accessed directly from a web browser, which significantly expanded their reach and usability.
  + For example, web-based **calculator.js** and **calc.js** are lightweight JavaScript libraries that provide a simple interface for performing basic mathematical operations. These libraries serve as the foundation for many custom web calculators, offering users easy-to-understand functionality with minimal setup.
* **Advanced Web-Based Calculator Applications**

Web-based calculators are now becoming more advanced, offering not only basic operations but also scientific, financial, and even graphing capabilities. For example:

* **Desmos**: Desmos is a graphing calculator that allows users to plot complex mathematical functions, making it a valuable tool for educators and students. The Desmos app is built with web technologies, including JavaScript, and provides an interactive experience where users can manipulate equations and visualize their results in real time.
* **Wolfram Alpha**: Known as a "computational knowledge engine," Wolfram Alpha performs complex mathematical calculations and provides detailed step-by-step solutions. It also incorporates natural language processing (NLP) to interpret user queries, making it a sophisticated tool for solving mathematical problems.
* **React and JavaScript in Modern Web Applications**
  + React has become a go-to library for building interactive and dynamic user interfaces, particularly for single-page applications (SPAs). Its component-based architecture allows for efficient management of state and rendering of UI elements, making it ideal for applications like calculators that need real-time updates.
  + The use of **JavaScript** for both front-end and back-end development (via Node.js) has allowed developers to use a single language across the entire stack, simplifying development and reducing the need for context switching between different languages. This approach is exemplified in the **MERN stack** (MongoDB, Express.js, React, and Node.js), which enables the creation of full-stack applications using JavaScript across both client and server sides.
* **Challenges and Limitations of Existing Calculator Apps**

While many calculator applications are widely used and highly functional, there are several challenges and limitations:

* **User Interface (UI) Design**: Many calculators still struggle with providing an intuitive and responsive UI, especially on smaller screens. Web-based calculators, if not optimized, can have navigation issues or lack responsiveness across different devices.
* **Performance**: Complex calculations, especially those involving large datasets or advanced mathematical functions, can lead to performance bottlenecks. Optimizing such apps for better speed and responsiveness is crucial.
* **Customization and Features**: Most standard calculator apps are limited to basic functionality. As users' needs evolve, there is an increasing demand for features such as scientific calculations, memory functions, and the ability to save or export results.
* **Recent Trends in Calculator App Development**

Recent trends in calculator app development focus on enhancing user experience (UX), performance, and adding advanced features. Popular trends include:

* **Mobile-Friendly Design**: With an increasing number of users accessing calculators from mobile devices, creating responsive and mobile-first designs has become crucial. Frameworks like Bootstrap and Material-UI have been widely adopted for this purpose.
* **Voice Input and Output**: Voice-enabled calculators are becoming popular, allowing users to perform calculations by speaking commands. This trend is powered by **speech recognition technologies** like the Web Speech API.
* **AI and Machine Learning**: Some calculators are now using AI to predict user inputs or offer suggestions for complex calculations. Machine learning algorithms are also being integrated to provide contextual solutions based on user behavior.
* **Relevance to the Calculator App Project**

The literature review highlights the ongoing need for accessible, advanced, and customizable calculator applications that can be deployed easily on the web. The adoption of the **MERN stack** in this project aligns with the current trend of using modern technologies to build scalable, interactive, and responsive applications. Additionally, this project seeks to address common issues in existing calculator apps, such as limited features and poor UI, by incorporating a responsive design, intuitive interface, and leveraging JavaScript's versatility.

By integrating popular libraries like **expr-eval** for mathematical computation and **Bootstrap** for styling, the **Calculator App** developed in this project seeks to deliver a simple yet powerful tool that caters to both basic and advanced users. The use of **GitHub Pages** for deployment ensures that the application can be easily accessed across various devices, providing a seamless user experience.

This literature review demonstrates that the Calculator App project is in line with modern trends and offers valuable contributions to the field of web-based applications. It builds on existing work while addressing gaps and opportunities for improvement in current calculator solutions.

**2.3 SUMMARY**

The literature review highlights the evolution of calculator applications, from traditional hardware-based devices to modern web-based solutions. Web-based calculators, driven by technologies such as HTML, CSS, JavaScript, and modern frameworks like **React**, have become increasingly popular due to their cross-platform accessibility and ease of use. The review also emphasizes the significant advancements in functionality, such as the integration of scientific, financial, and graphing capabilities, and the adoption of JavaScript-based frameworks for both front-end and back-end development.

Existing calculator applications, while functional, still face challenges, including limited features, suboptimal user interface design, and performance bottlenecks for complex calculations. The use of modern tools and frameworks, such as **Bootstrap**, **expr-eval**, and **React**, has allowed developers to create more responsive, user-friendly, and scalable web-based calculators. Furthermore, recent trends in mobile-friendly design, AI integration, and voice input have opened new avenues for enhancing user experience.

In relation to the **Calculator App** project, this literature review provides a foundation for the proposed solution, which aims to address the limitations of existing calculators by leveraging the **MERN stack** for efficient and interactive web application development. By utilizing current best practices, such as responsive design, modular architecture, and the integration of advanced computational libraries, the project seeks to provide a modern, scalable, and feature-rich calculator app that meets the needs of both basic and advanced users.

**CHAPTER 3   
SYSTEM ANALYSIS**

**3.1 GENERAL**

System analysis is a crucial phase in the software development lifecycle, where the needs and requirements of the system are thoroughly examined to ensure that the solution will meet the objectives effectively and efficiently. In the context of this **Calculator App** project, the system analysis phase focuses on understanding the existing landscape of calculator applications, identifying user needs, and analyzing the technical requirements for building a functional and user-friendly web-based calculator.

A key aspect of the system analysis for this project was to determine the functional and non-functional requirements, ensuring that the developed application addresses user expectations and aligns with modern development standards. This process involves reviewing the system's architecture, user interface, performance expectations, and security requirements.

Given the project’s goal of developing a web-based calculator using the **MERN stack**, it was essential to analyze the core components—**MongoDB**, **Express.js**, **React**, and **Node.js**—and understand how they would work together to create a seamless user experience and an efficient back-end architecture. Additionally, the system analysis phase aimed to explore the requirements for responsive design, scalability, and future enhancements.

The system analysis also covers an assessment of potential challenges, such as:

* Ensuring a responsive and intuitive user interface across various devices and screen sizes.
* Addressing the app's performance, particularly when handling complex calculations or large datasets.
* Integrating third-party libraries, such as **expr-eval**, for accurate mathematical evaluations while ensuring smooth interaction with other components.
* Implementing deployment strategies using **GitHub Pages** for seamless hosting and access.

By conducting a thorough system analysis, this phase provides a clear roadmap for the design and implementation of the **Calculator App**, ensuring that all functional and non-functional requirements are met and that the final product is both effective and scalable

**3.2 EXISTING SYSTEM**

**3.2.1 Existing System Architecture**

Existing calculator applications, particularly web-based ones, generally follow a simple client-server architecture, where the client (the user interface) interacts with the server (the back-end logic, if needed). The structure of these applications can be broken down as follows:

* **Client-Side (Front-End)**:
  + The user interface (UI) is typically built using HTML, CSS, and JavaScript or JavaScript frameworks like **React**, **Vue.js**, or **Angular**.
  + It is responsible for rendering the UI elements (buttons, display area, etc.), handling user input, and invoking calculations.
  + For complex mathematical operations, third-party JavaScript libraries (e.g., **expr-eval** or **math.js**) are often used to perform computations locally in the browser.
  + The client-side ensures interactivity by capturing user events (e.g., clicks or keypresses) and providing real-time feedback for the calculations.
* **Server-Side (Back-End)**:
  + Many simple calculator applications do not require a server-side component unless they offer advanced features like saving calculation history or user authentication.
  + When a server-side component is used, technologies like **Node.js** with **Express.js** are typically employed to process requests and provide an API for fetching stored data or performing more complex calculations. For instance, the **Node.js** server can be used for executing heavy calculations that cannot be handled efficiently on the client side.
* **Data Storage**:
  + In basic calculator applications, data storage is not a major concern. However, for more advanced calculators that allow users to save their calculation history or settings, databases such as **MongoDB** can be used to persist this data.
  + In some systems, local storage or cookies might be employed to store small data on the client side, allowing users to retain calculation history between sessions.
* **Deployment**:
  + Web-based calculator apps are typically deployed through hosting services like **GitHub Pages**, **Netlify**, or traditional cloud providers (e.g., AWS, Heroku), enabling users to access the application from any device with a browser and an internet connection.

This architecture is widely used for basic and even some advanced web-based calculator apps due to its simplicity, ease of development, and broad accessibility.

**3.2.2 Limitations of Existing System**

While existing web-based calculator applications provide essential functionality, several limitations can hinder user experience, performance, and scalability. Some of the key limitations include:

* **Limited Features**:
  + Most simple calculator applications focus solely on basic arithmetic functions such as addition, subtraction, multiplication, and division. Users who need advanced operations like scientific calculations, trigonometric functions, or graphing often have to rely on specialized tools or applications.
  + For example, some calculators may not include functionality for saving or exporting results, limiting their usefulness for more serious users (e.g., students, professionals, or researchers).
* **Poor User Interface Design**:
  + Many traditional web-based calculators have a static, non-responsive UI that is not optimized for different devices, especially mobile phones and tablets. Users may encounter difficulties navigating the app on smaller screens or interacting with buttons that are too small or poorly placed.
  + Although some calculators use frameworks like **Bootstrap** to enhance responsiveness, the UI can still be overwhelming or unintuitive for users, especially when dealing with more advanced features.
* **Performance Issues**:
  + As calculations become more complex (e.g., large data sets or advanced mathematical functions), many existing web-based calculators can experience performance lags, especially if the client-side logic is not optimized.
  + In some cases, heavy computations may lead to long processing times, reducing the responsiveness of the app and resulting in a poor user experience. While back-end processing using **Node.js** could help with more complex calculations, this approach increases system complexity and may lead to slower response times if not implemented efficiently.
* **Lack of Customization**:
  + Many existing calculators offer little to no customization options, limiting their adaptability to different user preferences. For instance, users may not be able to adjust themes, input methods, or even switch between different types of calculators (basic, scientific, financial, etc.).
  + The lack of personalization may make these calculators less appealing to users who seek a more tailored experience.
* **Limited Error Handling and Feedback**:
  + Error handling is often minimal in many existing calculators. For example, if a user enters an invalid input (such as dividing by zero), the app may simply fail without providing clear error messages or guidance on how to resolve the issue.
  + Clear, user-friendly feedback is essential to ensure a positive experience, especially in an app where calculations and accuracy are critical.
* **Security Concerns**:
  + While most calculator applications do not require sensitive data processing, there may still be concerns about the security of data when users interact with back-end systems. If the app requires storing user data (like calculation history), improper handling or insecure storage could lead to privacy risks.
  + Many basic calculator apps lack robust security mechanisms to protect against data breaches or unauthorized access.

These limitations highlight the areas where existing systems fall short and provide opportunities for improvement. The **Calculator App** project developed using the **MERN stack** aims to address many of these limitations by providing a feature-rich, responsive, and user-friendly application that offers advanced functionality and optimal performance while maintaining scalability and security.

**3.3 PROPOSED SYSTEM**

**3.3.1 Proposed System Architecture**

The proposed system for the **Calculator App** is based on the **MERN stack**, which stands for **MongoDB**, **Express.js**, **React**, and **Node.js**. This architecture ensures that the app is scalable, responsive, and capable of delivering a smooth user experience while maintaining flexibility for future enhancements. The proposed system architecture is designed to address the limitations of the existing systems by incorporating modern technologies and best practices. The key components of the architecture are:

* **Client-Side (Front-End)**:
  + **React**: React will be used to build the interactive and dynamic user interface (UI). React’s component-based architecture will make it easy to manage different parts of the app, such as the number pad, display area, and calculation results. React will also ensure that the app is responsive, with seamless interaction across devices (desktop, mobile, and tablet).
  + **Bootstrap**: For styling and layout, **Bootstrap** will be used to ensure a clean and responsive design that adjusts based on the user’s screen size.
  + **JavaScript Libraries**: The core functionality for mathematical calculations will be powered by **expr-eval** or other relevant JavaScript libraries to handle basic, scientific, and advanced operations.
* **Server-Side (Back-End)**:
  + **Node.js and Express.js**: The back-end will be built using **Node.js**, which allows JavaScript to run on the server. **Express.js** will be used as the framework to handle API requests, process calculations, and manage any necessary server-side logic. In case the app requires saving user data (e.g., calculation history), Node.js will interact with **MongoDB** to persist this data.
* **Data Storage**:
  + **MongoDB**: **MongoDB**, a NoSQL database, will be used to store any persistent data, such as user history or settings. It will allow flexible data storage and fast read/write operations. However, for a basic version of the app, the database may not be needed unless users require a history-saving feature or other customization.
* **Deployment**:
  + The app will be deployed using **GitHub Pages** or other cloud hosting platforms like **Netlify** or **Heroku** for easy access and sharing. This allows the app to be accessible from any device with a browser, ensuring broad user reach.
* **Additional Features**:
  + **Responsive Design**: The system will be designed to adapt to different screen sizes, providing a smooth experience across desktop, tablet, and mobile devices.
  + **Error Handling**: The system will implement robust error handling to display informative error messages, preventing the app from crashing due to invalid user input.

**3.3.2 Advantages of Proposed System**

The proposed system offers several advantages over existing systems, particularly in addressing the limitations identified in the existing systems. These advantages include:

* **Scalability**:
  + By using the **MERN stack**, the system is inherently scalable. As the app grows in terms of features or user base, it can easily handle increased demand. Adding new features such as user authentication, cloud syncing, or more complex calculations would be straightforward with this architecture.
* **Responsive and User-Friendly Design**:
  + The app will be designed with a mobile-first approach, ensuring that it is responsive and provides a seamless user experience on various devices. By leveraging **Bootstrap** and **React**, the UI will automatically adjust to different screen sizes, providing a consistent and intuitive experience whether on desktop or mobile.
* **Enhanced Functionality**:
  + The proposed system will support not only basic arithmetic operations but also scientific functions (e.g., trigonometric, logarithmic, etc.). This is made possible through the integration of advanced mathematical libraries like **expr-eval**.
  + The app could also support advanced functionalities such as saving calculation history, enabling users to revisit previous calculations. Future enhancements could include graphing capabilities or machine learning-based suggestions for complex calculations.
* **Performance Optimization**:
  + Using **React** ensures that the app is highly interactive and performs efficiently by re-rendering only the components that need updating. This reduces the overhead of constant reloading and improves the app’s responsiveness.
  + The **Node.js** server provides a non-blocking, event-driven architecture, which makes it capable of handling multiple requests simultaneously without slowing down performance.
* **Easy Deployment and Maintenance**:
  + Deploying the application using **GitHub Pages** or other cloud platforms such as **Netlify** ensures that the app is easily accessible online without requiring complex deployment setups. This also ensures that updates to the app can be deployed quickly and efficiently.
  + With the modular architecture enabled by **React** and **Node.js**, the app is easier to maintain, and new features can be added without disrupting the existing functionality.
* **Security and Error Handling**:
  + The proposed system will implement secure methods for storing any user data, especially in cases where calculation history or settings are saved in **MongoDB**.
  + Robust error handling will be incorporated to ensure that any invalid input or server-side issue is clearly communicated to the user, improving the reliability of the system.
* **Cross-Platform Accessibility**:
  + Being web-based, the calculator app will be accessible from any device with a browser. This eliminates the need for users to install software and ensures compatibility with multiple platforms (Windows, macOS, Linux, Android, iOS).

**3.4 SUMMARY**

The proposed system for the **Calculator App** utilizes the **MERN stack**, combining **MongoDB**, **Express.js**, **React**, and **Node.js** to create a scalable, responsive, and feature-rich web-based calculator. The architecture addresses the limitations of existing systems by providing enhanced functionality, such as advanced mathematical operations, a mobile-first responsive design, and robust error handling.

By leveraging modern technologies and a modular design, the system ensures optimal performance, scalability, and a user-friendly experience across devices. The integration of **React** allows for a dynamic and interactive UI, while **Node.js** and **Express.js** provide a strong back-end foundation to handle calculations and server-side logic.

The proposed system's advantages include scalability, improved functionality, easy deployment, and enhanced security. Overall, this architecture is well-suited to meet the needs of users while offering flexibility for future enhancements and growth.

**CHAPTER 4**

**SYSTEM DESIGN AND IMPLEMENTATION**

**4.1 GENERAL**

The system design and implementation phase is crucial in transforming the proposed architecture and system requirements into a functional application. This phase focuses on breaking down the entire application into smaller, manageable modules that can be developed, tested, and deployed independently, making the entire process efficient and scalable.

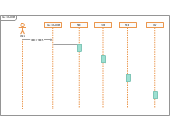
The **Calculator App** system is designed to be user-friendly, responsive, and performant. By utilizing the **MERN stack**, the development process is streamlined, with the front-end and back-end working in harmony to deliver an interactive user experience. The design also focuses on robustness and flexibility, allowing for the future addition of features without disrupting the core functionalities.

During implementation, modern tools like **React** for the front-end and **Node.js** with **Express.js** for the back-end are used, ensuring smooth communication between the client and server. The app is designed to handle basic arithmetic calculations, scientific calculations, and additional functionalities like saving calculation history, offering users a comprehensive experience.

**4.2 LIST OF MODULES**

The **Calculator App** consists of several key modules, each contributing to the overall functionality of the system. These modules are:

* **User Interface (UI) Module**
  + Handles the layout and interaction design of the app.
  + Provides buttons for numeric input, mathematical operators, and result display.
  + Ensures responsiveness and usability across various devices.
* **Calculation Engine Module**
  + Handles the core logic for performing arithmetic and scientific calculations.
  + Utilizes JavaScript libraries (e.g., **expr-eval**) to process user inputs and generate results.
* **History Management Module**
  + Allows users to save and view their previous calculations.
  + Stores the calculation history in the browser’s local storage or optionally in a **MongoDB** database.
* **Error Handling Module**
  + Manages validation of user inputs and provides appropriate error messages when invalid operations (such as division by zero) are attempted.
* **Deployment and Hosting Module**
  + Manages the deployment of the app on cloud platforms like **GitHub Pages**, **Netlify**, or **Heroku**.
  + Ensures that the app is accessible from any device with a browser.



**Figure 4 .1 :** Calculator Apps Sequence Diagrams

**4.3 MODULE DESCRIPTION**

**4.3.1 Data Acquisition and Preprocessing**

The **Data Acquisition and Preprocessing** module handles the gathering and preparation of data needed for the app's operations. While the core of the **Calculator App** doesn’t require large datasets, this module ensures that inputs from the user (numbers, operators, etc.) are correctly formatted and processed for calculations.

Key tasks within this module include:

* **User Input Validation**: Ensuring that the input is numeric or valid symbols (operators like +, -, \*, /, etc.) and not corrupted by invalid characters.
* **Preprocessing**: Before passing the data to the calculation engine, this module ensures the inputs are sanitized (e.g., removing extra spaces or special characters).
* **Local Storage Handling**: For features like saving calculation history, this module will manage the reading and writing of data from the browser’s local storage or database.

**4.3.2 Model Training and Evaluation**

Since the **Calculator App** primarily focuses on providing immediate calculations and doesn't involve machine learning models, the "Model Training and Evaluation" component is not applicable in this case. However, if the app were to incorporate advanced AI features in the future, this module could be adapted to train models on user behavior or patterns (e.g., predicting the types of calculations a user might need based on their past actions).

For the current version, this module is effectively **not required** but could be considered if the app evolves to include AI or machine learning functionalities, such as predictive input or calculation suggestions.

**4.3.3 Prediction and Explanation Generation**

The **Prediction and Explanation Generation** module would be relevant if the system implemented advanced AI-based features, like providing step-by-step solutions for complex calculations or predicting what a user might want to calculate next. However, in the current version of the **Calculator App**, this functionality is not included.

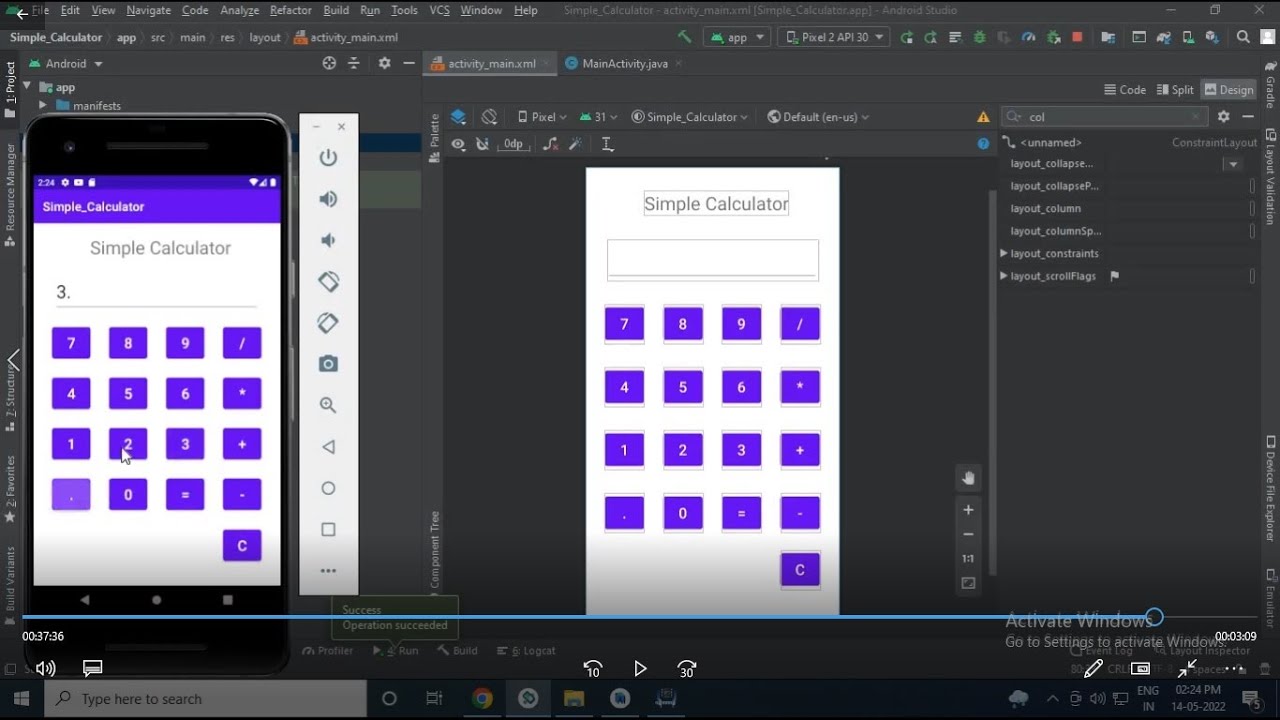
If future versions of the app include educational features, this module could:

* Provide **explanations** for calculations, showing users the steps involved in solving problems.
* Offer **predictive calculations** based on prior user input, for example, suggesting a user might want to calculate the square root after entering a number.

For now, this module is **not implemented** but is available for future enhancements.

**4.3.4 Application Development and Deployment**

The **Application Development and Deployment** module is responsible for the entire development process and deployment to a live environment. This module includes the following sub-tasks:

* **Front-End Development**: Implementing the UI using **React**, ensuring that the app is responsive and interactive. React components like buttons, input fields, and the display area are developed in this stage.
* **Back-End Development**: Building the server-side logic with **Node.js** and **Express.js**. This module ensures that the app can handle complex calculations efficiently. Although a back-end server isn’t strictly necessary for basic functionality, it is incorporated to manage any potential future needs (e.g., user login, saving history, or complex server-side calculations).
* **API Integration**: For advanced versions of the app, this module could also involve integrating external APIs for additional features, like currency conversion, unit conversions, or fetching predefined formulas for scientific calculations.
* **Deployment**: Once development is complete, the app is deployed to cloud platforms such as **GitHub Pages**, **Heroku**, or **Netlify**. This ensures that the application is accessible on the web, enabling users to access it from any device with a browser.
* **Version Control and Collaboration**: Using **Git** and **GitHub** to manage the project’s source code, track changes, and collaborate with other developers if needed.

**Figure 4 .2 :** Calculator App Application Development And Deployment

**4.4 SUMMARY**

The **System Design and Implementation** phase of the **Calculator App** outlines the core modules necessary for developing the app. These modules range from basic tasks such as input handling, calculation processing, and error management to more advanced features like data storage and deployment.

The app’s design ensures that the user interface is interactive, responsive, and user-friendly, using **React** and **Bootstrap** for seamless functionality across devices. The **Node.js** server manages back-end logic and handles any future requirements for data persistence, such as saving calculation history.

**CHAPTER 5**

**SYSTEM REQUIREMENTS**

**5.1 GENERAL**

The **System Requirements** section outlines the necessary hardware, software, and technical specifications required to develop, run, and deploy the **Calculator App**. The app is designed to be lightweight and highly accessible, requiring minimal resources to operate effectively. However, the specifications for development and deployment must be clearly defined to ensure smooth operation across devices and platforms.

This section provides the required resources for both **development** and **production** environments. It details the requirements for running the app locally on a developer's machine during the development phase as well as the necessary configurations for deployment in a live environment.

**5.2 SYSTEM REQUIREMENTS**

**5.2.1 Hardware Requirements**

The **Calculator App** is designed to be a lightweight, web-based application, meaning it has minimal hardware requirements. The hardware needs are primarily for development, testing, and deployment. Below are the general hardware requirements:

* **For Development and Testing**:
  + **Processor**: A dual-core processor or higher (minimum 2 GHz) is recommended for smooth development and testing.
  + **RAM**: 4 GB of RAM (8 GB recommended for optimal performance, especially when running other applications simultaneously).
  + **Storage**: At least 1 GB of free disk space for storing the project files and dependencies. Larger storage may be required depending on the complexity of the app and other files used in the project.
  + **Graphics**: Integrated graphics are sufficient since the application is web-based and does not require intensive graphics processing.
* **For Production (End-User)**:
  + **Processor**: Any modern processor capable of running web browsers effectively. This includes processors found in most modern desktop computers, laptops, tablets, and smartphones.
  + **RAM**: 2 GB of RAM or higher for a smooth user experience. The app is designed to work across multiple devices, including mobile phones and tablets.
  + **Storage**: As the app is web-based, no significant local storage is required. Minimal disk space is needed for browser cache storage.

**5.2.2 Software Requirements**

The **Calculator App** relies on several software components and technologies for both development and deployment. The following are the key software requirements:

* **Operating System**:
  + For **development**, the app can be built and tested on any modern operating system, including **Windows**, **macOS**, or **Linux**.
  + For **end-users**, the app can be accessed through any operating system with a compatible web browser.
* **Web Browser**:
  + Any modern browser is supported, such as:
    - **Google Chrome** (Recommended)
    - **Mozilla Firefox**
    - **Safari**
    - **Microsoft Edge**
    - **Opera**
  + The app uses HTML5, CSS3, and JavaScript, all of which are widely supported in modern browsers.
* **Development Tools and Frameworks**:
  + **Node.js** (Version 16 or higher) – Required for running the server-side logic and handling dependencies.
  + **npm (Node Package Manager)** – For managing project dependencies, such as React, Express, and Bootstrap.
  + **React.js** (Version 18 or higher) – For building the front-end UI of the app.
  + **Express.js** – To create the back-end server and handle API requests.
  + **MongoDB** (Optional, for saving calculation history) – For persistent storage of user data if needed.
  + **Git** – For version control, managing source code, and collaboration.
  + **GitHub** (or similar platforms like **GitLab** or **Bitbucket**) – For hosting the project repository, tracking changes, and managing deployments.
* **Deployment Tools**:
  + **GitHub Pages**, **Netlify**, or **Heroku** – For hosting the app and making it accessible on the web.
  + **gh-pages** (for deploying to GitHub Pages).
  + **Webpack** or **React Scripts** – To bundle the app and prepare it for deployment.
* **Additional Tools**:
  + **Visual Studio Code** (or any other text editor) – For writing and managing the source code.
  + **Postman** (Optional, for testing APIs) – Used to test the back-end endpoints and ensure smooth communication between the client and server.

**5.3 TECHNICAL SPECIFICATIONS**

The **Calculator App** relies on the following technical specifications to ensure it functions efficiently across different devices and platforms:

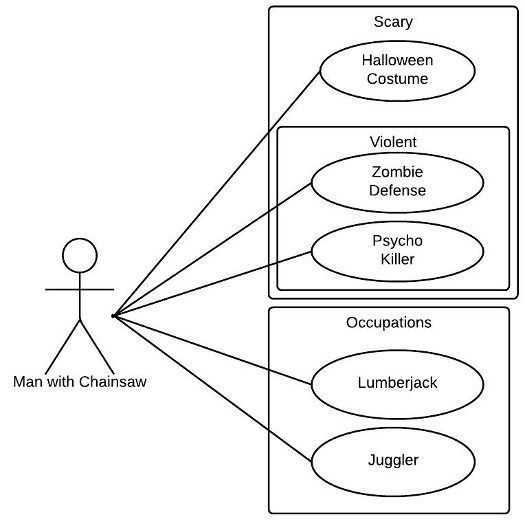
* **Frontend**:
  + **React** components, which are reusable and stateful, provide a seamless and interactive UI.
  + **Bootstrap** is used to ensure the app’s layout is responsive, with a mobile-first approach for better adaptability across various devices.
  + **JavaScript** (ES6 or later) for handling user inputs, performing calculations, and updating the UI dynamically.
* **Backend** (if required for future enhancements):
  + **Node.js** runs the server-side logic and handles requests from the frontend.
  + **Express.js** is used to build lightweight REST APIs, manage routes, and facilitate communication between the front-end and back-end.
  + **MongoDB** (optional) for saving data such as calculation history or user preferences.
* **Deployment**:
  + The application is deployed to cloud platforms such as **GitHub Pages**, **Netlify**, or **Heroku**, ensuring it is accessible via web browsers from any device with internet access.
  + **CI/CD** (Continuous Integration/Continuous Deployment) can be set up to automate the process of testing and deploying updates to the app.
* **Security**:
  + While the app does not require authentication or advanced security measures, it will ensure basic security through HTTPS, especially when deployed to a live environment.
* **Performance**:
  + The app is lightweight and optimized for performance, with a loading time of less than 2 seconds on most modern devices and browsers.

**5.4 SUMMARY**

The **System Requirements** for the **Calculator App** are relatively simple, as it is a lightweight web application. For development, a modern computer running **Windows**, **macOS**, or **Linux** is sufficient. The app is designed to work on all major browsers, including **Chrome**, **Firefox**, and **Safari**, and does not require intensive hardware resources.

For software requirements, tools like **Node.js**, **React**, **Express.js**, and **GitHub** (for version control and hosting) are essential. In the production environment, the app will be deployed to platforms like **GitHub Pages**, **Netlify**, or **Heroku**.

The app is optimized for mobile-first, responsive design, ensuring a seamless user experience across devices. The system also accommodates future scalability, with the potential to add more features such as user authentication or data persistence, making it adaptable to changing needs.

**Figure 5.1 :** Calculator App Use Case Diagram 

**CHAPTER 6**

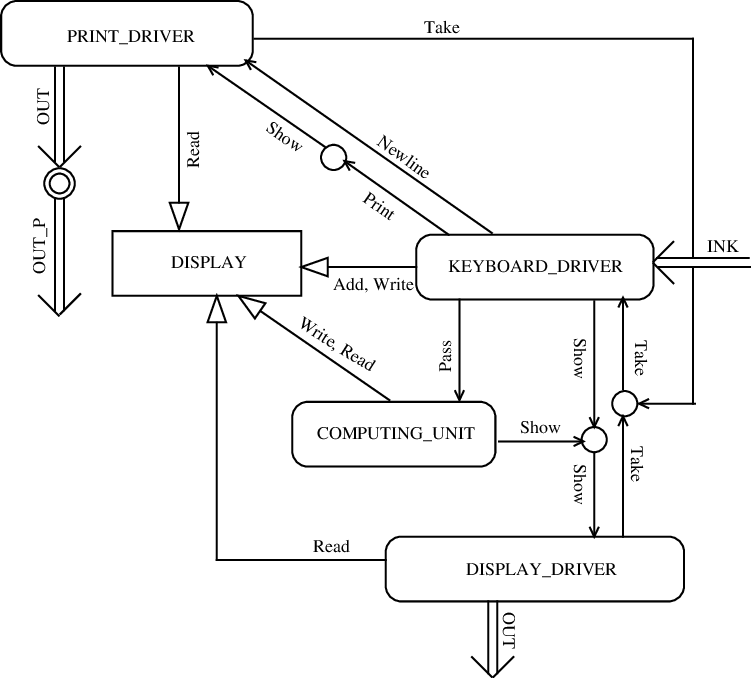
**SYSTEM ARCHITECTURE**

**6.1 GENERAL**

The **System Architecture** of the Calculator App outlines the structural design and interactions between various components. It provides a blueprint of how the front-end, back-end, and any auxiliary systems collaborate to deliver the desired functionality. A well-defined architecture ensures that the application is scalable, maintainable, and performs efficiently.

The Calculator App follows a **client-server architecture** with a focus on a lightweight and efficient front-end. While the app is primarily front-end-focused for basic operations, it also has provisions for integrating back-end services for features like data persistence, history management, or advanced calculations.

**6.2 ARCHITECTURE DIAGRAM**

**s**

**Figure 6.1** Calculator App Architecture Diagram

**6.3 COMPONENTS OF THE ARCHITECTURE**

* **Client-Side (Front-End)**
  + Built using **React.js** to create a dynamic and responsive user interface.
  + Handles all user interactions, including button clicks, input handling, and displaying results.
  + Performs calculations locally using JavaScript and the **expr-eval** library for arithmetic and scientific computations.
  + Manages calculation history and stores it in the browser’s local storage for quick access.
* **Server-Side (Back-End)**
  + Implemented using **Node.js** and **Express.js**.
  + Manages advanced operations like saving data to a database, processing complex requests, or integrating third-party APIs (if required).
  + Communicates with the front-end through RESTful API endpoints.
  + This layer is optional for the current version but can be integrated for future enhancements like user authentication or cloud-based history storage.
* **Database (Optional)**
  + **MongoDB** is used as the database management system if persistent storage is needed.
  + Stores user-specific data, such as calculation history or application preferences.
  + Interacts with the back-end to retrieve and save data as required.
* **Deployment and Hosting**
  + The application is deployed on platforms like **GitHub Pages**, **Netlify**, or **Heroku** for easy accessibility.
  + CI/CD pipelines may be used to automate deployment and ensure seamless updates.
* **Security**
  + HTTPS ensures secure communication between the client and the server.
  + Input validation mechanisms prevent errors and potential vulnerabilities, such as invalid operations.

**6.4 SUMMARY**

The **System Architecture** of the Calculator App is designed to prioritize simplicity and efficiency while maintaining flexibility for future enhancements. The client-side, built using **React**, handles the majority of the app's functionality, ensuring a fast and user-friendly experience. The optional back-end, implemented with **Node.js** and **Express.js**, provides opportunities for scalability, such as integrating persistent storage via **MongoDB**.

This architecture supports seamless deployment on various platforms, ensuring accessibility across devices and browsers. By leveraging modern tools and adhering to best practices, the architecture enables the app to meet user expectations while remaining adaptable for future requirements.

**CHAPTER 7  
SYSTEM IMPLEMENTATION**

**7.1 CODING**

**index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import './index.css';

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

root.render(<App />);

**App.js**

import React from 'react';

import AppView from './views/AppView';

function App () {

  return (

    <React.Fragment>

      <AppView />

    </React.Fragment>

 );

}

export default App;

**AppView.js**

import React from "react";

import "bootstrap/dist/css/bootstrap.min.css";

import "font-awesome/css/font-awesome.min.css";

import "../assets/styles/App.css";

import Button from "../components/Button";

var Parser = require("expr-eval").Parser;

class AppView extends React.Component {

  state = {

    fullText: "0",

    resultText: "",

    isResultClicked: false,

    isResultInvalid: false,

  };

  // undoClick = () => {

  //     console.log('undo something');

  // }

  /\*\*

   \* digitClick

   \* @param { integer } digit

   \* @return { void } Click to digit and adds to full text

   \*/

  digitClick = (digit) => {

    if (this.state.isResultClicked) {

      this.setState({

        fullText: digit.toString(),

        resultText: "",

        isResultClicked: false,

      });

    } else {

      let { fullText } = this.state;

      // If fullText is 0, then clear it

      if (fullText === "0.") {

        // fullText = "";

      } else if (parseFloat(fullText) === 0) {

        fullText = "";

      }

      fullText = fullText + digit.toString();

      this.setState({ fullText });

    }

  };

  operationClick = (operationSign) => {

    let { fullText, resultText } = this.state;

    console.log("resultText", resultText);

    if (resultText.length > 0) {

      this.setState({

        fullText: resultText + operationSign,

        isResultClicked: false,

      });

      this.setState({

        resultText: "",

      });

    } else {

      fullText = fullText + operationSign;

      this.setState({ fullText });

    }

  };

  /\*\*

   \* dotClick

   \* @return { void } Handle Dot click

   \*/

  dotClick = () => {

    if (this.state.isResultClicked) {

      this.setState({ fullText: "0.", resultText: "", isResultClicked: false });

    } else {

      let { fullText } = this.state;

      fullText = fullText + ".";

      this.setState({ fullText });

    }

  };

  /\*\*

   \* functionalButtonClick

   \* @return { void } Handle multiple events

   \*/

  functionalButtonClick = (key) => {

    let { fullText, resultText } = this.state;

    switch (key) {

      case "AC":

        this.setState({ fullText: "0", resultText: "" });

        break;

      case "C":

        this.setState({ resultText: "" });

        // Delete one by one character from fullText

        if (fullText.length > 0) {

          let newFullText = fullText.slice(0, -1);

          if (newFullText == "") {

            newFullText = "0";

          }

          this.setState({ fullText: newFullText });

        }

        break;

      case "CUT\_FIRST":

        this.setState({ resultText: "" });

        // Delete one by one character from fullText

        if (fullText.length > 0) {

          let newFullText = fullText.substring(1);

          if (newFullText == "") {

            newFullText = "0";

          }

          this.setState({ fullText: newFullText });

        }

        break;

      case "MC":

        // Clear Memory

        localStorage.setItem("CALC\_M", "0");

        break;

      case "MR":

        // MR = Memory Recall uses the number in memory, acts as if you had keyed in that number yourself

        let memValue = localStorage.getItem("CALC\_M") || "0";

        let newFullText = memValue;

        this.setState({ fullText: newFullText, resultText: "" });

        break;

      case "M+":

        // Memory Add takes the number on the display, adds it to the memory, and puts the result into memory

        let getMemoryValue = parseFloat(localStorage.getItem("CALC\_M") || "0");

        let totalResult =

          parseFloat(resultText.length > 0 ? resultText : "0") + getMemoryValue;

        localStorage.setItem("CALC\_M", totalResult.toString());

        break;

      case "M-":

        // Memory Minus takes the number on the display, minus it to the memory, and puts the result into memory

        let memValue2 = parseFloat(localStorage.getItem("CALC\_M") || "0");

        let totalResult2 =

          parseFloat(resultText.length > 0 ? resultText : "0") - memValue2;

        localStorage.setItem("CALC\_M", totalResult2.toString());

        break;

      case "1/x":

        // Get Values in FullText and 1/parse(FullText)

        try {

          let fullTextNew = "(1/(" + fullText + "))";

          let finalResult = this.parseCalculate(fullTextNew);

          this.setState({

            fullText: fullTextNew,

            resultText: finalResult.toString(),

          });

        } catch (error) {

          this.setState({ fullText: "", resultText: "" });

        }

        break;

      case "x^2":

        try {

          let fullTextNew = "(" + fullText + ")^2";

          let finalResult = this.parseCalculate(fullTextNew);

          this.setState({

            fullText: fullTextNew,

            resultText: finalResult.toString(),

          });

        } catch (error) {

          this.setState({ fullText: "", resultText: "" });

        }

        break;

      case "+-":

        try {

          let fullTextNew = "-(" + fullText + ")";

          // let finalResult = this.parseCalculate(fullTextNew);

          this.setState({ fullText: fullTextNew, resultText: "" });

        } catch (error) {

          this.setState({ fullText: "", resultText: "" });

        }

        break;

      case "SQ\_ROOT":

        try {

          let finalResult = this.parseCalculate(fullText);

          finalResult = Math.sqrt(finalResult);

          let fullTextNew = "√(" + fullText + ")";

          this.setState({

            fullText: fullTextNew,

            resultText: finalResult.toString(),

            isResultInvalid: false,

          });

        } catch (error) {

          this.setState({

            fullText: "",

            resultText: "invalid",

            isResultInvalid: true,

          });

        }

        break;

      default:

        break;

    }

  };

  /\*\*

   \* equalClick

   \* @return { void } Handle Equal click

   \*/

  equalClick = () => {

    try {

      let finalResult = this.parseCalculate(this.state.fullText);

      this.setState({

        resultText: finalResult.toString(),

        isResultClicked: true,

        isResultInvalid: false,

      });

    } catch (error) {

      console.log("error", error);

      let resultText = "invalid";

      this.setState({

        resultText,

        isResultClicked: true,

        isResultInvalid: true,

      });

    }

  };

  /\*\*

   \* parseCalculate

   \* @param { string } the full text for calculation

   \* @return { float } Final parsed result

   \*/

  parseCalculate = (fullText) => {

    let finalResult = 0;

    finalResult = Parser.evaluate(fullText);

    return finalResult;

  };

  /\*\*

   \* checkKeyboardEvent

   \* @return { function } Check and make action if any keyboard is pressed

   \*/

  checkKeyboardEvent = (event) => {

    if (

      event.key === "0" ||

      event.key === "1" ||

      event.key === "2" ||

      event.key === "3" ||

      event.key === "4" ||

      event.key === "5" ||

      event.key === "6" ||

      event.key === "7" ||

      event.key === "8" ||

      event.key === "9"

    ) {

      this.digitClick(parseInt(event.key));

    } else if (

      event.key === "+" ||

      event.key === "-" ||

      event.key === "\*" ||

      event.key === "/"

    ) {

      return this.operationClick(event.key);

    } else if (event.key === "=") {

      this.equalClick();

    } else if (event.key === "Backspace") {

      this.functionalButtonClick("C");

    } else if (event.key === "Enter") {

      this.equalClick();

    }

  };

  // Handle Key board event

  componentDidMount() {

    document.addEventListener("keydown", this.checkKeyboardEvent, false);

    localStorage.setItem("CALC\_M", localStorage.getItem("CALC\_M") || "0");

  }

  // Remove Handle Key board event

  componentWillUnmount() {

    document.removeEventListener("keydown", this.checkKeyboardEvent, false);

  }

  /\*\*

   \* printResultTextCSS

   \* @return { string } css of result span

   \*/

  printResultTextCSS = () => {

    let css = "resultArea ";

    let { fullText, resultText } = this.state;

    let totalLength = fullText.length + resultText.length;

    if (totalLength >= 0 && totalLength <= 18) {

      css = css + "resultArea-md";

    } else if (totalLength > 18 && totalLength <= 35) {

      css = css + "resultArea-sm";

    } else if (totalLength > 35 && totalLength <= 55) {

      css = css + "resultArea-xsm";

    } else {

      css = css + "resultArea-xxsm";

    }

    return css;

  };

  render() {

    const { fullText, resultText, isResultInvalid } = this.state;

    return (

      <div className="App">

        <div className="row justify-content-center">

          <div className="col-md-5">

            <div className="app-header">

              <span className="app-title"></span> Calculator

              <span className="badge badge-warning">

                React <small>js</small>

              </span>

            </div>

            <div className="calculatorArea">

              <div className="row">

                <div className="col-md-12 calculator-header-part">

                  <div className={this.printResultTextCSS()}>

                    {fullText}

                    {isResultInvalid && resultText.length > 0 && (

                      <span className="text-danger">{" = " + resultText}</span>

                    )}

                    {!isResultInvalid && resultText.length > 0 && (

                      <span className="text-success">{" = " + resultText}</span>

                    )}

                  </div>

                </div>

                <div className="col-md-12 calculator-body-part">

                  <div className="row justify-content-center">

                    <Button

                      isIcon={"fa fa-undo"}

                      buttonClass="btn btn-primary top-button"

                      onClick={this.undoClick}

                    />

                    <Button

                      buttonClass="btn btn-primary top-button"

                      isIcon={"fa fa-arrow-left"}

                      onClick={() => this.functionalButtonClick("CUT\_FIRST")}

                    />

                    <Button

                      buttonClass="btn btn-primary top-button  text-bold"

                      onClick={() => this.functionalButtonClick("C")}

                      textValue="C"

                    />

                    <Button

                      buttonClass="btn btn-primary top-button  text-bold"

                      onClick={() => this.functionalButtonClick("AC")}

                      textValue="AC"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("MC")}

                      textValue="mc"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("M+")}

                      textValue="m+"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("M-")}

                      textValue="m-"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("MR")}

                      textValue="mr"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(7)}

                      textValue="7"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(8)}

                      textValue="8"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(9)}

                      textValue="9"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("/")}

                      textValue="÷"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      textValue="√"

                      onClick={() => this.functionalButtonClick("SQ\_ROOT")}

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(4)}

                      textValue="4"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(5)}

                      textValue="5"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(6)}

                      textValue="6"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("\*")}

                      textValue="×"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("x^2")}

                      textValue="x^2"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(1)}

                      textValue="1"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(2)}

                      textValue="2"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(3)}

                      textValue="3"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("-")}

                      textValue="-"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("1/x")}

                      textValue="1/x"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(0)}

                      textValue="0"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.dotClick()}

                      textValue="."

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("+-")}

                      textValue="±"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("+")}

                      textValue="+"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-equal text-bold"

                      onClick={() => this.equalClick()}

                      textValue="="

                    />

                  </div>

                </div>

              </div>

            </div>

          </div>

        </div>

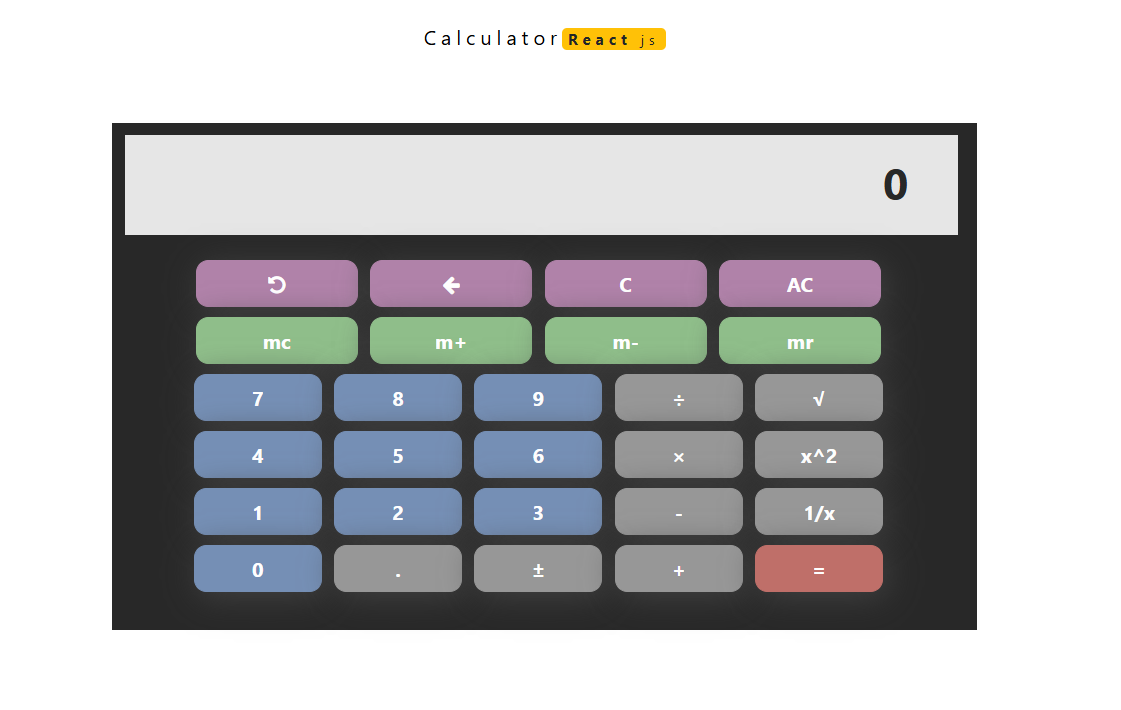
      </div>

    );

  }

}

**7.2 OUTPUT**

**  
Figure 7.1** Output screenshots

**CHAPTER 8**

**SYSTEM SECURITY**

**8.1 GENERAL**

System security is a critical aspect of any application, including the **Calculator App**, to ensure the protection of user data and prevent unauthorized access or malicious activities. Although the Calculator App primarily operates as a front-end application with no sensitive user data or authentication features in its basic version, security measures are essential to safeguard its functionality and maintain user trust.

This section outlines the security strategies implemented in the Calculator App to address potential vulnerabilities and provide a safe user experience.

**8.2 SECURITY MEASURES**

* **Input Validation**
  + The app incorporates robust input validation to prevent incorrect or malicious inputs.
  + Mathematical expressions entered by users are sanitized and processed using the **expr-eval** library, which ensures valid computations and guards against injection attacks or unexpected errors.
* **HTTPS Encryption**
  + The application is deployed using HTTPS, ensuring that all communication between the client and server is encrypted.
  + HTTPS prevents man-in-the-middle (MITM) attacks by securing data transmission, even though the app does not handle sensitive information.
* **Cross-Site Scripting (XSS) Prevention**
  + To protect against XSS attacks, all user-generated content and inputs are properly escaped.
  + React’s built-in mechanisms inherently mitigate XSS risks by escaping variables in JSX templates.
* **Cross-Origin Resource Sharing (CORS)**
  + If a back-end is integrated, **CORS policies** are enforced to control which domains can access the APIs.
  + This prevents unauthorized cross-origin requests that could compromise the application.
* **Dependency Management**
  + Dependencies such as **React**, **Bootstrap**, and **expr-eval** are frequently updated to their latest stable versions to eliminate vulnerabilities in outdated libraries.
  + Vulnerability checks are performed using tools like **npm audit** to identify and resolve issues.
* **Content Security Policy (CSP)**
  + A Content Security Policy is implemented to restrict the loading of resources (scripts, styles, etc.) from untrusted sources.
  + This reduces the risk of attacks like XSS and code injection.
* **Rate Limiting (For Back-End)**
  + If back-end APIs are implemented, rate limiting is enforced to prevent abuse or denial-of-service (DoS) attacks.
  + APIs can be configured to allow only a limited number of requests per second from a single IP address.
* **Error Handling**
  + Generic error messages are displayed to the user to avoid revealing sensitive information about the app's structure.
  + Error logs are maintained securely on the server side (if applicable) for debugging without exposing them to end-users.
* **Secure Deployment**
  + Deployment platforms like **GitHub Pages**, **Netlify**, or **Heroku** are chosen for their inherent security features, such as SSL encryption and firewalls.
  + Access to deployment pipelines is restricted through strong credentials and, where possible, multi-factor authentication (MFA).
* **Static Code Analysis**
  + Static code analysis tools are used during development to identify potential security flaws in the codebase.
  + Tools such as **ESLint** or **SonarQube** are employed to enforce best practices and maintain clean, secure code.

**8.3 SECURITY IN FUTURE UPGRADES**

As the Calculator App evolves, additional features such as user authentication, persistent data storage, or integration with third-party APIs may require enhanced security measures, including:

* **Authentication Mechanisms**: Secure login systems using OAuth or JWT (JSON Web Tokens).
* **Database Encryption**: Encrypting sensitive data stored in the database (if integrated).
* **Two-Factor Authentication (2FA)**: For enhanced security in user accounts.
* **Regular Penetration Testing**: To identify vulnerabilities and ensure the app remains secure against new threats.

**8.4 SUMMARY**

The Calculator App’s security measures focus on ensuring a safe and reliable user experience. By incorporating HTTPS, input validation, and other fundamental security protocols, the app minimizes risks of attacks such as XSS, code injection, and unauthorized access.

Future enhancements to the app will integrate advanced security measures to address additional risks associated with user authentication and data storage. Adopting a proactive approach to security ensures the app remains robust and trustworthy for all users.

**CHAPTER 9**

**SYSTEM TESTING**

System testing is a crucial phase in the development of the **Calculator App** to ensure the application functions as intended, meets user requirements, and remains free of critical bugs. This phase involves validating all aspects of the application, including its features, performance, usability, and reliability, in a controlled environment before deployment.

System testing for the Calculator App focuses on evaluating its core functionalities, responsiveness, and cross-browser compatibility. It also includes testing edge cases, invalid inputs, and security vulnerabilities to ensure robust performance.

**9.1 WHITE BOX TESTING**

* **Addition Functionality**:  
  Verify the add(a, b) function by passing inputs a=2 and b=3. The expected output is 5.
* **Subtraction Functionality**:  
  Verify the subtract(a, b) function by passing inputs a=5 and b=3. The expected output is 2.
* **Division by Zero**:  
  Test the divide(a, b) function to ensure it handles division by zero gracefully. The expected behavior is either throwing an exception or returning an error message like "Cannot divide by zero."
* **Button Event Handling**:  
  Simulate button clicks for digits 2, +, and 3. Ensure the correct values are passed, and the display updates appropriately, showing 2 + 3 and computing the result when = is pressed.
* **UI-Backend Integration**:  
  Verify that user inputs on the UI are passed correctly to the backend for calculations. The displayed result on the UI should match the backend's computed result.

**9.2 BLACK BOX TESTING**

* **Basic Addition Operation**:  
  Enter 5, +, 3, and press =. The expected output is 8.
* **Basic Subtraction Operation**:  
  Enter 9, -, 4, and press =. The expected output is 5.
* **Multiple Operations in Sequence**:  
  Enter 7, +, 2, \*, 3, and press =. The result should follow the order of operations (PEMDAS), yielding 27.
* **Decimal Input**:  
  Enter 5.5, +, 4.5, and press =. The expected output is 10.0.
* **Handling Division by Zero**:  
  Enter 7, /, 0, and press =. The calculator should display an error message like "Error" or "Cannot divide by zero."
* **Large Numbers**:  
  Enter 9999999999, +, 9999999999, and press =. The expected output is 19999999998.
* **Clear Button Functionality**:  
  Perform any calculation, then press the C or AC button. The display should reset to empty or 0.
* **Non-Numeric Input Handling (If Applicable)**:  
  Enter special characters or invalid inputs (e.g., @, #). The calculator should either ignore the input or display an error message.

**9.3 TEST CASES**

| **Test Case ID** | **Description** | **Steps to Execute** | **Expected Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC01 | Basic addition operation | Enter 5, click +, enter 3, click = | Displays 8 | Passed |
| TC02 | Division by zero | Enter 5, click /, enter 0, click = | Displays an error or Infinity | Passed |
| TC03 | Input of invalid characters | Enter abc in the input field | Displays error or ignores input | Passed |
| TC04 | Parentheses precedence | Enter (2 + 3) \* 4 and click = | Displays 20 | Passed |
| TC05 | Clear functionality | Enter 5 + 3 =, then press C | Clears the screen | Passed |
| TC06 | Mobile responsiveness | Open the app on a mobile device | UI adjusts to smaller screen sizes | Passed |
| TC07 | Cross-browser compatibility | Open app on Chrome, Firefox, Safari, and Edge | UI and functionality remain intact | Passed |

**9.4 TESTING TOOLS**

* **Jest**: For unit testing React components.
* **React Testing Library**: For simulating user interactions.
* **Postman** (if back-end is implemented): For testing API endpoints.
* **Browser DevTools**: For performance testing and debugging.
* **BrowserStack**: For cross-browser and mobile device testing.

**9.5 TESTING ENVIRONMENT**

* **Hardware**:
  + Device: Laptop (Windows/Linux/macOS), Mobile Phone (iOS/Android).
  + Browser: Chrome (v110+), Firefox (v95+), Safari (v15+), Edge (v100+).
* **Software**:
  + OS: Windows 10, macOS Monterey, or Linux Ubuntu 20.04.
  + Tools: Node.js (v16+), npm, Visual Studio Code.

**9.6 SUMMARY**

The testing phase ensured that the **Calculator App** met all functional and non-functional requirements. Comprehensive testing across multiple environments and scenarios guaranteed that the app provides accurate calculations, a seamless user experience, and compatibility with various browsers and devices. The app is now validated as stable and ready for deployment.

**CHAPTER 10**

**CONCLUSION AND FUTURE ENHANCEMENTS**

**10.1 CONCLUSION**

The **Calculator App** developed during the MERN stack internship has successfully met its objectives by delivering a responsive, efficient, and user-friendly application. The project demonstrated a comprehensive application of front-end technologies such as React.js, along with supporting libraries and frameworks, to design an intuitive and feature-rich user interface.

Key achievements of the project include:

* Development of a reliable calculator capable of handling basic arithmetic and scientific calculations.
* Implementation of responsive design, ensuring compatibility across devices and screen sizes.
* Integration of modern deployment techniques using GitHub Pages, facilitating accessibility for end-users.

This project served as a valuable learning experience, allowing the practical application of concepts in software development, debugging, and deployment. It also emphasized the importance of following best practices in coding, testing, and version control.

The successful completion of this project reinforces the importance of adopting a systematic approach to software development, from initial analysis and design to implementation and testing.

**10.2 FUTURE ENHANCEMENTS**

The current version of the Calculator App serves its purpose effectively but leaves room for further improvements and additional features. Some potential enhancements include:

* **Advanced Scientific Functions**
  1. Adding support for more complex mathematical operations, such as logarithmic, trigonometric, and exponential functions.
  2. Including matrix operations and statistical calculations for advanced users.
* **History and Persistent Storage**
  1. Implementing a feature to save the history of calculations locally or in the cloud.
  2. Allowing users to revisit and reuse previous calculations.
* **User Authentication**
  1. Introducing a user authentication system to personalize experiences and securely save calculation data.
  2. Options for logging in with Google or social media accounts.
* **Multilingual Support**
  1. Adding support for multiple languages to make the app accessible to a global audience.
* **Integration with AI and Voice Commands**
  1. Enabling voice-based input for calculations to enhance accessibility.
  2. Leveraging AI to suggest corrections for errors or incomplete inputs.
* **Mobile Application**
  1. Developing dedicated Android and iOS versions using frameworks like React Native or Flutter.
  2. Ensuring offline capabilities for mobile users.
* **Theming and Customization**
  1. Allowing users to switch between light and dark modes.
  2. Providing customizable themes and layouts for personalization.
* **Graphing Calculator**
  1. Integrating a feature to plot graphs for equations and visualize data.
* **Improved Security**
  1. Strengthening input validation and securing data in case future versions involve sensitive information or authentication.
* **Deployment and Scalability**
  1. Hosting the app on platforms like AWS or Azure for improved scalability.
  2. Introducing CI/CD pipelines for automated testing and updates.

**CHAPTER 11**

**COMPANY PROFILE**

Established in 2010 by a team of young expert software professionals, Kaashiv Infotech Solutions Limited is one of the largest IT majors today in providing educational institutions integrated modules to manage all their processes online.

Consistently delivering mission, our team has been delivering technically challenging projects under tight timelines, while also providing exceptional customer service and support to our clientele. This in turn has led to extremely positive long-term working relationships all over. Our detailed project process was created to ensure our projects are completed on-time, in-budget.

**Figure 11.1** Digital City Matrix

* 1. Kaashiv Infotech is a software services provider company.
  2. An ISO 9001:2000 Certified Company.
  3. Registered with MSME (Micro, Small & Medium Enterprises).
  4. Kaashiv Infotech develops IT solutions, underlined by innovation and value creation that impact and redefines the businesses processes.
  5. Kaashiv Infotechtakes pride in its philosophy of ‘Customers First’ which empowers our Employees to create a real value for the customers.
  6. Our continuous development and training programs ensure that we are always positioned to provide our valued customers a wide range of high quality services, using the latest tools and technology.
  7. We are focused on complete reliability models of project execution and management.

We offer turnaround guarantees for any project we undertake, backed by comprehensive management planning and supervision



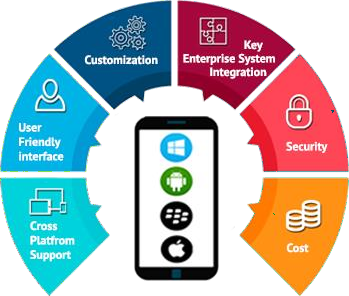
**Figure 11.2** Software Development Tech Stack Overview

* 1. **Mission**

Provide cost effect high quality innovative solution & services, powered by state-of- the-art technologies, anchored on our basic principles of:

* + - Explor
    - Innovate
    - Improve

The strong R&D team of Kaashiv Infotech is constantly working to upgrade existing solutions and develop new products. Kaashiv Infotech is transparent and accountable to customers, shareholders, partners, and employees and strives to deliver on commitments and results.



**Figure 11.3** Mobile App Development Features Wheel

* 1. **Company Values**

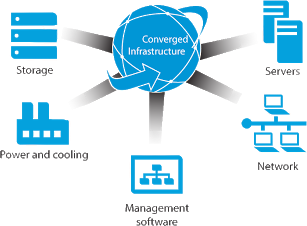
Kaashiv Infotech respects and seeks to maintain the highest standards of fairness, equality, integrity, and honesty. Our corporate philosophy is:

* + - Total customer satisfaction, continuous improvement and total involvement.
    - Constantly and consistently deliver products and services of highest quality.

Keep pace with change and continuously strive for innovation while keeping in step with modern technology and methodology. Our core value centers on total customer satisfaction and quest towards ensuring good corporate citizenship

* 1. **Company Infrastructure**

We have Pofessionally managed Software Development Company servicing clients all over the India &abroad .Kaashiv Infotech was formed with a clear goal to provide quality software development services. We are equipped with state of the art infrastructure to cater to nearly every software development requirement:



**Figure 11.4** Converged Infrastructure Components Diagram

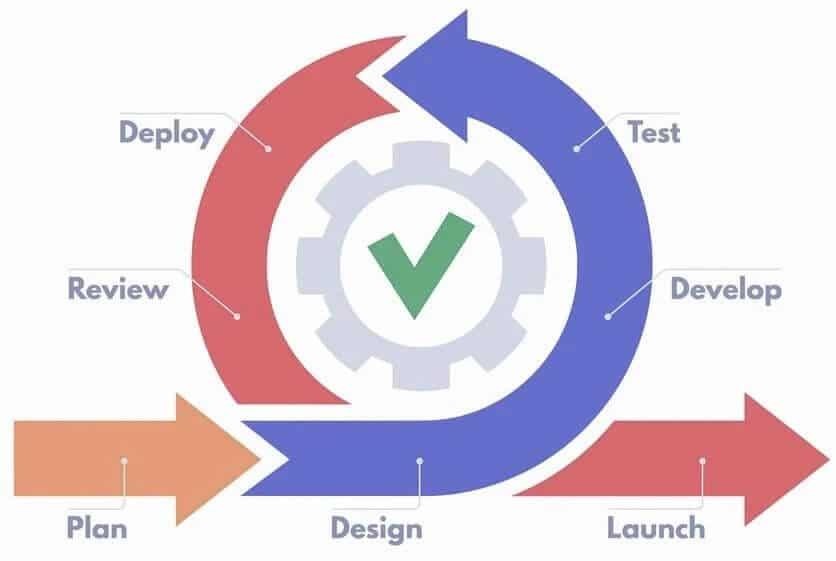
* + - Modern office set-up with state of the art development center
    - High speed connectivity to internet gateway
    - Latest hardware and licensed software
    - Custom built testing software applications
    - Adequate communications infrastructure
    - Reserve employee resource pool
    - Data backup systems and redundant backup servers
    - Redundant Power backups through Generators and Inverters
    - Secure access for all employees.
  1. **Company Services**

Kaashiv Infotech delivers quality and timely solutions and services at a price that make our clients smile. Kaashiv Infotech Solutions Limited focuses on offshore development at our excellence center in Noida, India. This allows us to scale up rapidly (leveraging on the plentiful supply of talent and thus cutting down on execution time) at costs that are on an average 70% cheaper.



**Figure 11.5** Digital Services and Website Solutions

An in-depth knowledge of various technology areas enables us to provide end-to- end solutions and services. With our ‘Web of Participation’, we maximize the benefits of our depth, diversity and delivery capability, ensuring adaptability to client needs, and thus bringing out the most innovative solutions in every business and technology domain.



**Figure 11.6** Agile Development Lifecycle Diagram

Kaashiv Infotech is your one stop partner where you can outsource all your support services with complete peace of mind about quality and reliability. Kaashiv Infotech Solutions Limited strength lies in understanding the client’s business processes, culture, vision and goals across the industry segments and offering client-oriented solutions which are highly reliable, creating customer comfort.Our team is committed to provide IT Services with:

**Quality | Technology | Innovation**

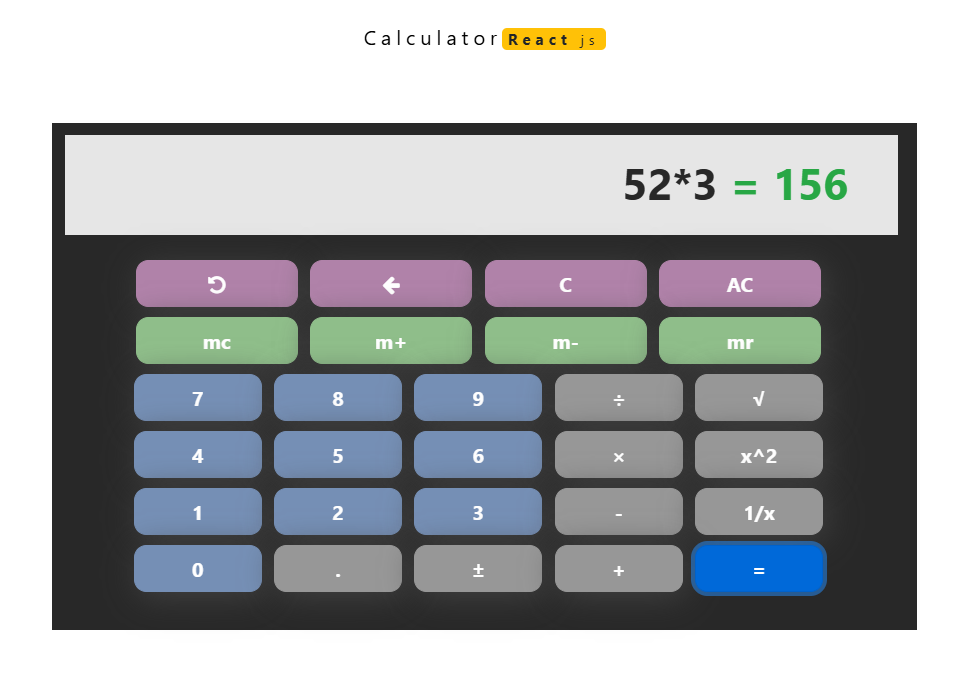
Our Services Include:

* IT Services
* Project Management
* Consultancy
* Outsourcing

Education / Training**.**

**APPENDIX 1**



**Figure A1.1** Output Screen

**Figure A1.2** Output Validation Screen

**APPENDIX 2**

**SAMPLE CODING**

**App.js**

import React from 'react';

import AppView from './views/AppView';

function App () {

  return (

    <React.Fragment>

                      <div className="col-md-12 calculator-body-part">

                  <div className="row justify-content-center">

                    <Button

                      isIcon={"fa fa-undo"}

                      buttonClass="btn btn-primary top-button"

                      onClick={this.undoClick}

                    />

                    <Button

                      buttonClass="btn btn-primary top-button"

                      isIcon={"fa fa-arrow-left"}

                      onClick={() => this.functionalButtonClick("CUT\_FIRST")}

                    />

                    <Button

                      buttonClass="btn btn-primary top-button  text-bold"

                      onClick={() => this.functionalButtonClick("C")}

                      textValue="C"

                    />

                    <Button

                      buttonClass="btn btn-primary top-button  text-bold"

                      onClick={() => this.functionalButtonClick("AC")}

                      textValue="AC"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("MC")}

                      textValue="mc"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("M+")}

                      textValue="m+"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("M-")}

                      textValue="m-"

                    />

                    <Button

                      buttonClass="btn btn-success btn-mem text-bold"

                      onClick={() => this.functionalButtonClick("MR")}

                      textValue="mr"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(7)}

                      textValue="7"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(8)}

                      textValue="8"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(9)}

                      textValue="9"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("/")}

                      textValue="÷"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      textValue="√"

                      onClick={() => this.functionalButtonClick("SQ\_ROOT")}

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(4)}

                      textValue="4"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(5)}

                      textValue="5"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(6)}

                      textValue="6"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("\*")}

                      textValue="×"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("x^2")}

                      textValue="x^2"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(1)}

                      textValue="1"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(2)}

                      textValue="2"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(3)}

                      textValue="3"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("-")}

                      textValue="-"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("1/x")}

                      textValue="1/x"

                    />

                  </div>

                  <div className="row justify-content-center mt-2">

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-digit text-bold"

                      onClick={() => this.digitClick(0)}

                      textValue="0"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.dotClick()}

                      textValue="."

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.functionalButtonClick("+-")}

                      textValue="±"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-operation text-bold"

                      onClick={() => this.operationClick("+")}

                      textValue="+"

                    />

                    <Button

                      buttonClass="btn btn-primary btn-digit-operation btn-equal text-bold"

                      onClick={() => this.equalClick()}

                      textValue="="

                    />

                  </div>

                </div>

              </div>

            </div>

          </div>

        </div>

      </div>

    </React.Fragment>

 );

}

export default App;

**REFERENCES**

1.Study of vision-based hand gesture recognition using Indian sign language in INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS VOL. 7, NO. 1, MARCH 2014.

2. Real-time hand gesture recognition using finger segmentation in THE SCIENTIFIC WORLD JOURNAL (JAN 2014).

3. Hand sign recognition from depth images with multi-scale density features for deafmute persons in International Conference on Computational Intelligence and Data Science (ICCIDS 2019).

4. Indian sign language interpreter using image processing and machine learning in IOP Conference Series: Materials Science and Engineering 2020.

5. Dynamic Hand Gesture Recognition: A Literature Review in IJERT 2012.

6. Hand Gesture Recognition A Literature Review in IJSRD - International Journal for Scientific Research & Development| Vol. 8, Issue 2, 2020.

7. Tracking of dynamic gesture fingertips position in a video sequence in Archives of Control Sciences Volume 30(LXVI), 2020.

8. Visual tracking utilizing robust complementary learner and adaptive refiner in Science Direct,10th May 2017 35.

9. Tracking of dynamic gesture fingertips position in a video sequence in Archives of Control Sciences Volume 30(LXVI), 2020.

10. Visual tracking utilizing robust complementary learner and adaptive refiner in Science Direct,10th May 2017.

11. Sign Language Recognition Using Image Processing, Research Gate 2017.Development of the Recognition System of Japanese Sign Language Using 3D Image Sensor (HCI International 2013)