**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 project. 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. 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 Guide **Ms. J. NIZU SHAHAIN, 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 design and development of the "Calculator Web App," a responsive, web-based calculator built using React and various supporting libraries. The primary goal of the project was to create a user-friendly calculator capable of performing basic arithmetic operations and evaluating complex mathematical expressions. The app leverages the expr-eval library to handle expression parsing, allowing users to input calculations in a more flexible, natural format.

To ensure a modern and appealing user interface, the project incorporates Bootstrap for layout and styling, providing a responsive design that adapts seamlessly to various screen sizes. Font Awesome icons enhance the visual appeal, making the interface intuitive and easy to navigate. JQuery is used to simplify DOM manipulation for dynamic elements, while React Testing Library and Jest DOM are employed to ensure robust and reliable user interactions through thorough testing.

Additionally, the application monitors and optimizes performance using Web Vitals, allowing for continuous tracking of key metrics like load time and responsiveness, which contribute to an efficient user experience. Challenges encountered during the project included implementing accurate expression evaluation, handling asynchronous updates in React, and ensuring the app’s performance across different devices.

The "Calculator Web App" demonstrates the effective use of modern JavaScript libraries and frameworks to create a high-quality, interactive web application. This project serves as a practical example of building and testing single-page applications with React, showcasing the combination of visual design, performance optimization, and mathematical functionality. Future improvements could include adding more advanced mathematical functions, customizable themes, and support for saving calculation history.

|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENTS** |  |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | ABSTRACT LIST OF FIGURES LIST OF ABBREVIATIONS | XI XV  XVII |
| **1.** | **INTRODUCTION OF WEB DESIGNING** | 1 |
| **2.** | **HISTORY OF WEB DESIGNING**   * 1. The start of the web and web design   2. Evolution of web design   3. End of the first browser wars | 2  2 2 3 |
| **3.** | **HTML**   * 1. Web page structure is based on HTML   2. Doctype lets the web browser know what to expect   3. HTML structure | 5 6  7  8 |
| **4.** | **CSS**   * 1. How we refer to CSS syntax   2. Styling a heading   3. Hexadecimal colors   4. Three ways to use styles   5. Internal versus external style sheets   6. Creating an external style sheet   7. What makes styles cascading | 14  15 16 20 21 22 23 27 |
| **5.** | **JAVASCRIPT**   * 1. Client-Side JavaScript   2. Advantage of Javascript   3. Limitations of Javascript   4. Javascript Development Tools   5. Javascript in Internet Explorer   6. JavaScript in Chrome | 28  29 30 30 31 32 33 |
| **6.** | **NODE JS**   * 1. Event-Driven Architecture | 36  36 |
| **7.** | **MONGODB**   * 1. Document-Oriented   2. Collections   3. Databases   4. Indexes | 40  41 41 41 41 |
| **8.** | **REACT.JS**   * 1. Components   2. JSX (JavaScript XML)   3. State and Props   4. Lifecycle Methods (Class Components)   5. Hooks (Functional Components)   6. Event Handling   7. Conditional Rendering   8. Lists and Keys   9. Form Handling   10. Routing | 43  43 44 44 45 45 45 45 45 45 45 |
| **9.** | **PROJECT OVERVIEW: CALCULATOR WEB APP**   * 1. Objective Material UI Components   2. Core Features API Integration   **9.3** Technology Stack  **9.4** User Experience and Design  **9.5** Future Enhancements | 46  46 46 46 46 47 |
| **10.** | **SOURCE CODE FOR THE CALCULATOR WEB APP** | 48 |
| **11.** | **COMPANY PROFILE**   * 1. Mission   2. Company Values   3. Company Infrastructure   4. Company Services | 62  63 64 65 66 |
| **12.** | **CONCLUSION  12.1** Conclusion  **12.2** References | 68 68 68 |

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| **FIGURE  NO** | **TITLE** | **PAGE NO** |
| **1.1** | Web Design Tools and Platforms Overview | 1 |
| **2.1** | History of Web Design Trends | 4 |
| **3.1** | HTML5 Basic Structure with Logo | 5 |
| **3.2** | HTML Code with Browser Representation | 6 |
| **4.1** | CSS Code with Color Transition Representation | 15 |
| **4.2** | CSS Syntax Components | 16 |
| **4.3** | Smoothie World Webpage Example | 19 |
| **5.1** | JavaScript Logo | 29 |
| **5.2** | HTML CSS JavaScript Differences | 35 |
| **6.1** | NodeJS Logo | 37 |
| **7.1** | MongoDB Data Exploration Process | 40 |
| **8.1** | React js logo | 43 |
| **10.1** | Output Screen | 63 |
| **11.1** | Digital City Matrix | 62 |
| **11.2** | Software Development Tech Stack Overview | 63 |
| **11.3** | Mobile App Development Features Wheel | 64 |
| **11.4** | Converged Infrastructure Components Diagram | 65 |
| **11.5** | Digital Services and Website Solutions | 66 |
| **11.6** | Agile Development Lifecycle Diagram | 67 |

LIST OF ABBREVIATIONS

API Application Programming Interface

UI User Interface

XAI Explainable AI

LIME Local Interpretable Model-agnostic Explanation

ML Machine Learning

HTML Hyper Text Markup Language

PCA Principle Component Analysis

**CHAPTER 1**

**INTRODUCTION OF WEB DESIGNING**

**Web design** refers to the **design** of websites that are displayed on the internet. It usually refers to the user experience aspects of **website** development rather than software development. W**eb design** encompasses many different skills and disciplines in the production and maintenance of websites. A **web designer** works on the appearance, layout, and, in some cases, content of a **website**.



**Figure 1.1** Web Design Tools and Platforms Overview

**CHAPTER 2**

**HISTORY OF WEB DESIGNING (1988–2001)**

Although web design has a fairly recent history. It can be linked to other areas such

as graphic design, user experience, and multimedia arts, but is more aptly seen from a technological standpoint. It has become a large part of people's everyday lives. It

is hard to imagine the Internet without animated graphics, different styles of typography, background, videos and music.

# The start of the web and web design

In 1989, whilst working at CERN Tim Berners-Lee proposed to create a global hypertext project, which later became known as the World Wide Web. During 1991 to 1993 the World Wide Web was born. Text-only pages could be viewed using a simple line-mode browser.In 1993 Marc Andreessen and Eric Bina, created the Mosaic browser. At the time there were multiple browsers, however the majority of them were Unix-based and naturally text heavy.

In 1994 Andreessen formed Mosaic Communications Corp. that later became known as Netscape Communications, the Netscape 0.9 browser. Netscape created its own HTML tags without regard to the traditional standards process.

# Evolution of web design

In 1996, Microsoft released its first competitive browser, which was complete with its own features and HTML tags. It was also the first browser to support style sheets, which at the time was seen as an obscure authoring technique and is today an important aspect of web design.

The HTML markup for tables was originally intended for displaying tabular data. However designers quickly realized the potential of using HTML tables for creating the complex, multi-column layouts that were otherwise not possible.

To create complex designs, many web designers had to use complicated table structures or even use blank spacer .GIF images to stop empty table cells from collapsing.CSS was introduced in December 1996 by the W3C to support presentation and layout. This allowed HTML code to be semantic rather than both semantic and presentational, and improved web accessibility, see tableless web design. In 1996, Flash (originally known as FutureSplash) was developed. At the time, the Flash content development tool was relatively simple compared to now, using basic layout and drawing tools, a limited precursor to ActionScript, and a timeline, but it enabled web designers to go beyond the point of HTML, animated GIFs and JavaScript

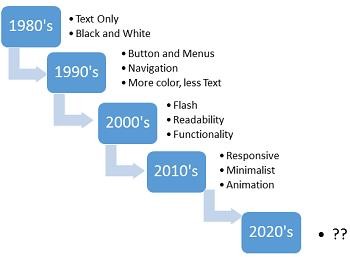
# End of the first browser wars

Further information: Browser wars § First Browser War (1995–2001) In 1998, Netscape released Netscape Communicator code under an open source licence, enabling thousands of developers to participate in improving the software. However, these developers decided to start a standard for the web from scratch, which guided the development of the open source browser and soon expanded to acomplete application platform.The Web Standards Project was formed and promoted browser compliance with HTML and CSS standards. It was also the first browser to fully support the PNG image format. By 2001, after a campaign by Microsoft to popularize Internet Explorer, Internet Explorer had reached 96% of web browser usage share, which signified the end of the first browsers wars as Internet Explorer had no real competition.

Since the start of the 21st century the web has become more and more integrated into people lives. As this has happened the technology of the web has also moved on. There have also been significant changes in the way people use and access the web, and this has changed how sites are designed.

Since the end of the browsers wars new browsers have been released. Many of these are open source meaning that they tend to have faster development and are more supportive of new standards. The new options are considered by many to be better than Microsoft's Internet Explorer.

The W3C has released new standards for HTML (HTML5) and CSS (CSS3), as well as new JavaScript API's, each as a new but individual standard. While the term HTML5 is only used to refer to the new version of HTML and some of the JavaScript API's, it has become common to use it to refer to the entire suite of new standards (HTML5, CSS3 and JavaScript).



**Figure 2.1** History of Web Design Trends

**CHAPTER 3**

**HTML**

Hypertext Markup Language, or HTML, is a programming language used to describe the structure of information on a web page. Together, HTML, CSS, and JavaScript make up the essential building blocks of websites, with CSS controlling a page’s appearance, and JavaScript programming its functionality. You can think of HTML as providing the bones of a web page, while CSS provides the skin, and JavaScript provides the brains.

A web page can contain headings, paragraphs, images, videos, and many other types of data. Front-end developers use HTML elements to specify what kind of information each item on a web page contains — for instance, the “p”



**Figure 3.1** HTML5 Basic Structure with Logo

element indicates a paragraph. Developers also write HTML code to specify how different items relate to one another in the overall structure of the page.

Every website you open in your browser, from social networks to music services, uses HTML. A look under the hood of any website would reveal HTML code providing structure for all the page’s components.



**Figure 3.2** HTML Code with Browser Representation

# 3.1 Web page structure is based on HTML

Hypertext Markup Language (HTML) documents use the .html or .htm extension. This extension allows a web browser or device such as a smartphone to understand that HTML content is on the page, and the content of the page is then rendered by the browser or device according to the rules of HTML.

Markup tags are used to defi ne the content on an HTML page. Markup tags are contained between greater than (<) and less than (>) symbols, and they are placed at the start and end of an object or text that is used in an HTML page. Here is an example of two heading 1 tags for text. The tags are not seen by the viewer of the web page, but every web browser knows that the text between the tags is a heading1.

# <h1>New Smoothie Recipe!</h1>

In this example, the <h1> is the opening tag and the </h1> is the closing tag. So this entire line of code is an element. More specifi cally, it is referred to as the heading 1 element. HTML and XHTML are closely related. There is a list of rules defined by the World Wide Web Consortium, or W3C that specify the perimeters of HTML and XHTML.

# Doctype lets the web browser know what to expect

The start of every web page should include a Doctype declaration, telling the Doctype declaration tells the web browser a little bit of information about what it is going to see on the page. Because there are different specifications for XHTML and HTML the web browser knows which language it’s about to see and render. Because a browser renders the page starting at the top line and then moves down, placing your doctype on the first line makes a lot of sense. While it’s not required, it’s good

form to always use Doctype at the start of your HTML pages. The doctype for HTML 4.0.1 looks like this:

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "**[**http://www.w3.org/TR/html4/loose.dtd**](http://www.w3.org/TR/html4/loose.dtd)**">**

When a web browser sees a doctype declaration, the browser expects that everything on the page that follows will use that language. If the page adheres to the specifications perfectly, it is considered valid.

# HTML structure

One of the most important concepts to understand when designing web content is the nested structure of HTML documents. Elements are often nested within each other. You will often start with the HTML structure first and then begin to style it with CSS.

As an example, let’s look at the basic elements that are in virtually every web page:

# <html>

**<body>**

# </body>

**</html>**

In this example, the <body> element is nested within the <html> element. In other words, <body> is placed between the opening <html> tag and the closing </html> tag, so nested tags are those that are placed between other opening and closing tags. These two elements <body> and <html> form the structure of all web pages; when a browser opens an HTML document, it looks for this structure. Content within the body tag is visible on the page as it is displayed within the web browser.

<html>

<body>

Nobody knows who invented smoothies, but the world wouldn’t be the same without them!

</body>

</html>

In HTML documents, some of the content is displayed to the viewer in their browser, but

there is also other code on the page that is hidden from view, but useful for the

search engine, or site developer. Examples of this hidden code include scripts to add

interactivity, code to help search engines categorize the document, and the styles that defi ne

the appearance of the page. This code is often found inside of the <head> element, and the

<head> element is nested within the <html> tags. An example of this is:

<html>

<head>

</head>

<body>

Nobody knows who invented smoothies, but the world wouldn’t be the same without them!

</body>

</html>

In the above example, there is no content in the <head> element just yet. Notice that the <head> element is nested within <html> but is not nested within <body>. The

<head> element opens and closes before the <body> element starts.

The <body> element contains text but it is lacking context so neither you nor a search engine can determine if it is a heading, list, quotation, or some other type of content. To define the text as a paragraph the <p> tag is used:

<html>

<head>

</head>

<body>

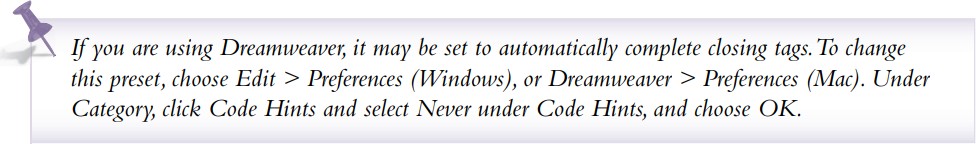
<p>Nobody knows who invented smoothies, but the world wouldn’t be the same without them!</p>

</body>

</html>

The paragraph element is now nested within the <body> element, which, in turn, is now nested within the <html> element. You will now open this document in a text editor and add to the fi le:

1. Choose File > Open and navigate to your web04lessons folder. Depending on which text editor you are using, you may need to select “All Files” instead of “Text Documents” in order to see the fi le. Choose the index.html fi le and then click Open. To get a better understanding of the structure of HTML and nesting of tags, you will add a hyperlink to this document linking the word SmoothieWorld to an external website.
2. In the last paragraph that reads “All content on this site is the copyright of SmoothieWorld,” click once before the word SmoothieWorld and then type the following code: <a>. This <a> is the opening for the anchor element, which you use to link to other pages in your site or elsewhere on the Internet.
3. Click to the right of the word SmoothieWorld and type </a>. This is the closing tag for the anchor tag and is required in XHTML.



To finish the job of creating a link, you need to add the destination of the link with the href attribute.

1. Click between the letter a and the closing bracket (>) in the opening tag. Press thespacebar once to add a space and type href="". The complete code should now read <a href="">. You now have an anchor tag and the href attribute. To finish the job of creating a hyperlink, you need to add the value of the attribute. In this case the value will be a URL — a web address.

1. Choose File > Save and then preview the page in your web browser by either opening your browser and choosing File > Open and navigating to the fi le you just saved, or by Ctrl + clicking (Mac OS) or right-clicking (Windows) the fi le and directing your operating system to open the fi le with a web browser. The link has the standard blue underlined appearance of a hyperlink that you have not yet visited.
2. Close your browser and return to your text editor.

**CHAPTER 4**

**CSS**

**Cascading Style Sheets** (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties.

A style sheet consists of a list of rules. Each rule or rule-set consists of one or more selectors, and a declaration block.

In CSS, selectors declare which part of the markup a style applies to by matching tags and attributes in the markup itself.

Selectors may apply to the following:

* all elements of a specific type, e.g. the second-level headers h2
* elements specified by attribute, in particular:
  + id: an identifier unique within the document
  + class: an identifier that can annotate multiple elements in a document
* elements depending on how they are placed relative to others in the document tree.

# How we refer to CSS syntax

**Figure 4.1** CSS Code with Color Transition Representation

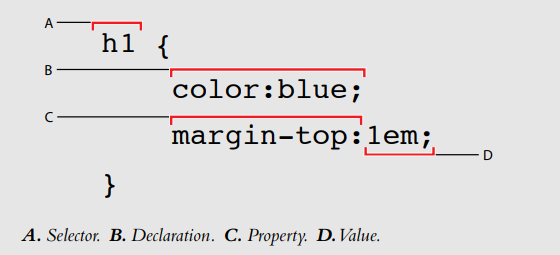
Before you begin to work with CSS, we need to explain how we will refer to the various parts of CSS syntax throughout this book. This is not as easy as it sounds because there is a gap between the official specification of the CSS language and the way designers often refer to CSS in the “real world.” Nevertheless, here are the fundamentals: all the following code is what we refer to as a rule in CSS:

h1 {

color:blue;margin-top:1em;

}

There are various components to this rule, as follows:

**Figure** **4.2** CSS Syntax Components

# Styling a heading

To get a sense of how CSS works, you’ll create a simple CSS rule that changes the style of a heading in your page. In your index.html page, you already have the content “SmoothieWorld” nested inside an <h1> tag. Perhaps one of the best ways to begin thinking about how CSS works is to consider how the default style of this heading is rendered in the browser.

1. Examine the heading of the file you previewed in the last step of the previous exercise. The style and formatting instructions are being provided by the browser. The size, color, and font are provided by the browser because exact formatting instructions are not specified. The browser only knows that this is a headline. You will redefine this style with a CSS rule.
2. In your code, locate the <title> tag on line 5, click once at the end of the line, following the closing tag, then press return to add a new line of code. Type the following:

<style type="text/css">

1. Press Return three times and then type </style>, This is a style element which you will use to place your rule for the style of the <h1> element.



The <style> element is nested within the <head> tags of your page. In HTML, everything nested inside the <head> section is not rendered by the browser on the page. For example, there is also a <title> element inside this section; this title appears at the top of the web browser, not on the actual page.

1. In the empty line below the opening <style> tag, type the following:

# h1 {

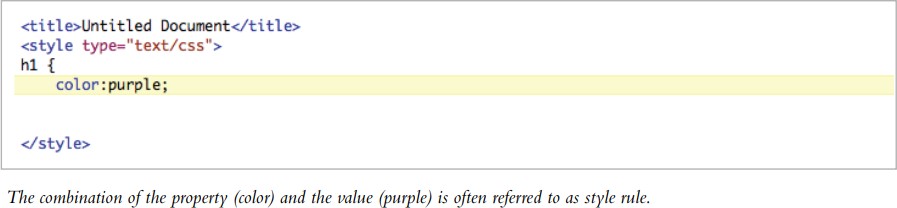
}

This is your selector. The selector is the HTML element you want to style, in this case the Heading 1 element.

1. Press Return and then press Tab to place your cursor below the curley bracket. The tab is optional but it helps make your CSS more readable. As you will soon see, the number of lines in this rule will grow and it’s worthwhile keeping the code easy to read.
2. Type the following code below the h1 {:

# color:purple;

The word color is referred to as a property in CSS syntax and the word purple is a value. The combined pair of a property and a value is called a declaration.



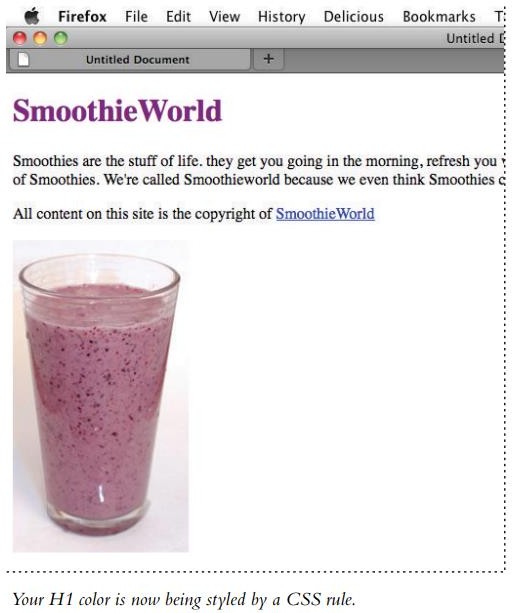
1. Press Return again and on the next line, type } which is the right curley bracket character. This closes the curley bracket you added in step 4. You now have three lines in this rule:

h1 {

color:purple;

}

1. Choose File > Save and then preview your page in the browser. The head is now a light purple color and you have successfully created your fi rst CSS rule. Close your browser and return to your text editor.



**Figure 4.3** Smoothie World Webpage Example

1. In the HTML fi le, select the word purple and type the following for the color value: #800080. This hexadecimal color is the equivalent of purple. You can use either named colors or hexadecimal colors when defi ning colors using CSS. Save your file and then preview it in the browser. The color remains the same. Hexadecimal colors are a more common method for describing colors.

# Hexadecimal colors

Color in both HTML and CSS color is referred to by a six-character code preceded by a pound sign. This code is called hexadecimal code, and is the system used to identify apply color to elements. You can reproduce almost any color using a unique hexadecimal code. For example, the following code is dark-red: a#CC0000.

The first, middle, and last pair of digits in the hexadecimal code correspond to values in the RGB spectrum. For instance, white, which is represented in RGB as R:255 G:255 B:255, is represented in HTML as #FFFFFF (255|255|255). A program like Photoshop will allow you to choose a specifi c RGB color in the Color Editor and give you the equivalent hexadecimal color for use in your code.

The rule you just created uses what is offi cially known as a “type selector” since it targets every instance of the h1 element type in your document. Type selectors assign CSS properties to an existing HTML tag. In this case the <h1> tag. All <h1> tags on this page will be displayed as purple. Type selectors are more commonly known as tag selectors. It is rare that you will actually hear someone use the phrase “type selector” but that is the official name for it, so we mention it here. You will now get an introduction to another category of CSS styles known as a class. You will also work with the <span> element which separates and controls inline content, such as a sentence.

# Three ways to use styles

In this exercise, your styles were located within the head section of the page. This type of style is called an internal style sheet. In addition to internal (or embedded) style sheets, there are external style sheets and inline styles.

An external style sheet is a separate document with the fi le extension .css. When using an external style sheet, all styles reside inside the style sheet document and you link it to your HTML pages. While internal style sheets aff ect only the page on which they exist, external styles can be applied to multiple pages.

Inline styles are used infrequently. With inline styles, the style rules are nested inside the HTML tags. An example of an inline style that colors a heading purple would look like this:

# <h1 style="color:purple">Smoothies</h1>

Inline styles are powerful because they override both internal and external styles, although they only apply to a single tag at a time. This embedded nature of inline styles means they are not easily re-used. In the simple example illustrated above you can see the style for the color purple is nested inside the <h1> tag. If you had 50

<h1> elements throughout your website and were using inline styles, you would add this style code 50 times. If you decided to change the color to green, you would need to locate and modify all 50 uses of the style. Inline styles are useful for single overrides or when an internal or external style sheet may not be available; a good example of this is HTML-based e-mail.

# Internal versus external style sheets

Internal style sheets are CSS rules contained directly within a document, using the <style> tag. The entire style sheet is contained within the opening and closing <style> tags. External style sheets are CSS rules saved in a separate document with the fi le extension .css. With internal style sheets, CSS rules apply only to the HTML in the current document. For example, if you had a 20 page website and were using internal style sheets, you would need to create a separate style sheet in each of the pages. A change to the style would require you to update the internal styles in each of the 20 separate pages.



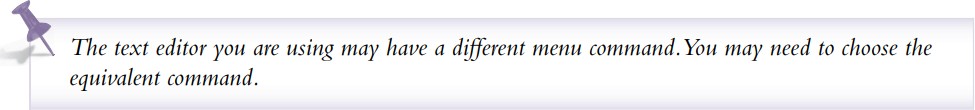
External style sheets place all the CSS rules for a site in a single document. You can attach the .css file to an unlimited number of HTML pages. This provides more fl exibility. If a style rule is changed in the external style sheet, all paragraphs across the site are modifi ed with a single step. You will make an external style sheet and then attach it to a new HTML page.

# Creating an external style sheet

An HTML page does not have to be limited to just one style sheet, and many large websites will break-up their styles into separate pages, making them easier to organize and maintain. You can even use style sheets for specific functions such as printing a page or for displaying a site on mobile devices. In Lesson 9, you will see that specific style sheets can even be used to make sites compatible with older web browsers when they are used to visit sites you create.

In this exercise, you will create a new external style sheet, move the style rules from your current document to the external style sheet, and then attach the style sheet to a new HTML page.

1. Choose File > New Text Document.



1. Choose File > Save. Name the document styles.css and save the styles.css fi le into the same folder as the HTML document you worked on in the previous exercise. An external cascading style sheet has a specific .css fi le extension but it is simply a text file.
2. Switch to the HTML document from the last exercise, but keep the style sheet open as well.
3. In the HTML document locate the rules you created within the <style> tags and then select them. Do not select the style tags themselves, just the rules that start with h1 and end with the closing bracket }.



1. Choose Edit > Cut, then switch to the styles.css fi le and choose Edit > Paste to paste the rules into the external style sheet document. Choose File > Save to save your style sheet. The entire external style sheet acts as a substitute for the <style> tags in the HTML

document. Now that you have moved the rules to this document you need to link it to your HTML page so that a web browser knows where to fi nd the style rules that apply to the HTML.

1. Switch back to the index.html page and choose File > Save. You will add the

<link> tag, pointing to the styles.css document. If you do not link to the external styles, the HTML page will have no styles.

1. Place your cursor after the closing style tag </ style> then press return to start a new line. Now type the following:

# <link rel="stylesheet" type="text/css" href="styles.css" />

You have added the rel, type, and href attributes. You may recall the href attribute from when you added the hyperlink in an earlier exercise. In order for your external style sheet to work properly, the name of the file, and the path to where it is located must both be accurate.

1. Choose File > Save and then preview the HTML page in your browser. The page should not change, as the same style is being used; it is simply being applied from outside the document.
2. Close the browser and return to your text editor. You’ll now create a new HTML document, and add the same link to the external CSS fi le, seeing how the rules are applied.
3. Choose File > Open and locate the fi le test.html in the web04lessons folder. This is an empty HTML document.
4. Continuing to work in your text editor, switch back to the index.html fi le and select the entire <link> element you typed in step 7:

# <link rel="stylesheet" type="text/css" href="styles.css" /> and then choose Edit > Copy

1. Switch back to the test.html document and then click below the <title> element and Choose Edit > Paste to place the <link> element, then save the the fi le by choosing File > Save.

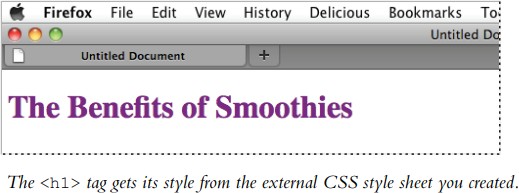


The external style sheet is now attached to this HTML document. Any HTML tags you add to this new document will be styled if there is a corresponding rule in the CSS file.

For example, the <h1> tag has a style of the color purple. 13 Click inside the <body> element and type:

# <h1>The Benefits of Smoothies </h1>

Save the file and preview it in your web browser



The heading is purple because the style rule for the <h1> element is color:purple and because this rule is located in an external sheet and linked in two places: the index.html and test.html pages. Because of this, you can control the style from of both HTML documents a central location.

# What makes styles cascading

You’ve seen three diff erent places where CSS rules are found: inline, internally, and externally. If there are confl icting defi nitions of styles between inline, internal, and external styles, the inline style will be used because it is closer to the HTML source. The internal style sheet takes precedence over an external style sheet, and defi nitions used in an external style sheet are used only if they don’t conflict with either inline or internal styles.

# CHAPTER 5

# JAVASCRIPT

**JavaScript** often abbreviated as **JS**, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output(I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

JavaScript engines were originally used only in web browsers, but they are now core components of other runtime systems, such as Node.js and Deno. These systems are used to build servers and are also integrated into frameworks, such as Electron and Cordova, for creating a variety of applications.

Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

**Figure 5.1** JavaScript Logo

# Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script

should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

# Advantage of Javascript

**Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

**Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.

**Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

**Richer interfaces:** You can use JavaScript to include such items as dragand-drop components and sliders to give a Rich Interface to your site visitors.

# Limitations of Javascript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript cannot be used for networking applications because there is no such support available.
* JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

**5.4 Javascript Development Tools**

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here:

**Microsoft FrontPage:** Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.

**Macromedia Dreamweaver MX:** Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

**Macromedia HomeSite 5:** HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

# Javascript in Internet Explorer

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

* Follow Tools -> Internet Options from the menu.
* Select Security tab from the dialog box.
* Click the Custom Level button.
* Scroll down till you find the Scripting option.
* Select Enable radio button under Active scripting.
* Finally click OK and come out.

To disable JavaScript support in your Internet Explorer, you need to select Disable radio button under Active scripting.

# Javascript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox

* Open a new tab -> type about: config in the address bar.
* Then you will find the warning dialog. Select I’ll be careful, I promise!
* Then you will find the list of configure options in the browser.
* In the search bar, type javascript.enabled.
* There you will find the option to enable or disable javascript by rightclicking on the value of that option -> select toggle

If javascript.enabled is true; it converts to false upon clicking toogle. If javascript is disabled; it gets enabled upon clicking toggle.

# JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome:

* Click the Chrome menu at the top right hand corner of your browser.
* Select Settings.
* Click Show advanced settings at the end of the page.
* Under the Privacy section, click the Content settings button.
* In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

# JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera:

* Follow Tools-> Preferences from the menu.
* Select Advanced option from the dialog box.
* Select Content from the listed items.
* Select Enable JavaScript checkbox.
* Finally click OK and come out.

To disable JavaScript support in Opera, you should not select the Enable JavaScript checkbox.

# Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using <noscript> tags. You can add a noscript block immediately after the script block as follows:

<html>

<body>

<script language="javascript" type="text/javascript">

<!--

document.write ("Hello World!")

//-->

</script>

<noscript>

Sorry...JavaScript is needed to go ahead.

</noscript>

</body>

</html>

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

# Difference Between HTML, CSS and Javascript:

**Figure 5.2** HTML CSS JavaScript Differences

HTML is a structural language that build the structure of a website as the skeletal system form the structure of the human body. Heading, paragraphs, images, text all are the part of HTML that creates the basic structure of a website.

CSS is a styling language that gives styling of a website. With the help of front color, background color and border styling CSS furnishes the look of the site as the skin gives look to the human body.

JavaScript is a programming language that gives motion and logics to the website for example a popup window alert. It is just like the motion of human body.

**CHAPTER 6**

**NODE JS**

Node.js is an open-source, cross-platform runtime environment for executing JavaScript code server-side. It is built on Chrome's V8 JavaScript engine and designed to build scalable network applications.



**Figure 6.1** NodeJS Logo

# Event-Driven Architecture:

Node.js uses an event-driven, non-blocking I/O model, making it efficient and suitable for I/O-heavy tasks.

# Single-Threaded Model:

Despite being single-threaded, Node.js can handle many connections concurrently using asynchronous operations

# Modules:

Node.js uses a modular approach, where functionalities are split into separate modules (e.g., http, fs for file system operations).

# Common Module Types:

**Core Modules:** Built-in (e.g., fs, http).

**Third-Party Modules:** Available via npm (Node Package Manager).

**Custom Modules:** Created by developers.

# npm (Node Package Manager):

npm is a package manager for Node.js that allows you to install, manage, and share modules.

Packages are stored in the node\_modules directory and specified in package.json.

# Basic Node.js Application Example:

* 1. **Creating a Basic HTTP Server:**

|  |
| --- |
| javascript |
| const http = require('http');  const server = http.createServer((req, res) => { res.statusCode = 200;  res.setHeader('Content-Type', 'text/plain');  res.end('Hello, World!\n');  });  server.listen(3000, '127.0.0.1', () => { console.log('Server running at http://127.0.0.1:3000/');  }); |

**http.createServer:** Creates a new HTTP server.

**server.listen:** Binds the server to a port and address.

# Using npm:

* **Initialize a Project:**

**npm init**

# Install a Package:

**npm install express**

* **Use a Package in Code:**

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hello, Express!');

});

app.listen(3000, () => {

console.log('Server running on http://localhost:3000/');

});

# Asynchronous Programming:

* **Callbacks:** Functions passed as arguments to handle results once an operation completes.
* **Promises:** Objects representing the eventual completion or failure of an asynchronous operation.
* **async/await:** Syntactic sugar for working with Promises, making asynchronous code look synchronous.

# Error Handling:

* **Try/Catch:** Used with async/await to handle errors.
* **Event Listeners:** Handle errors in asynchronous operations.

# Common Use Cases:

* **Web Servers:** Serve web pages and handle HTTP requests.
* **APIs:** Build RESTful or GraphQL APIs.
* **Real-Time Applications:** Chat applications, live updates.

# Example Project Structure:

**server.js:** Main application file.

**routes/**: Directory for route modules.

**controllers/**: Directory for business logic

**views/**: Directory for templates (if using a templating engine).

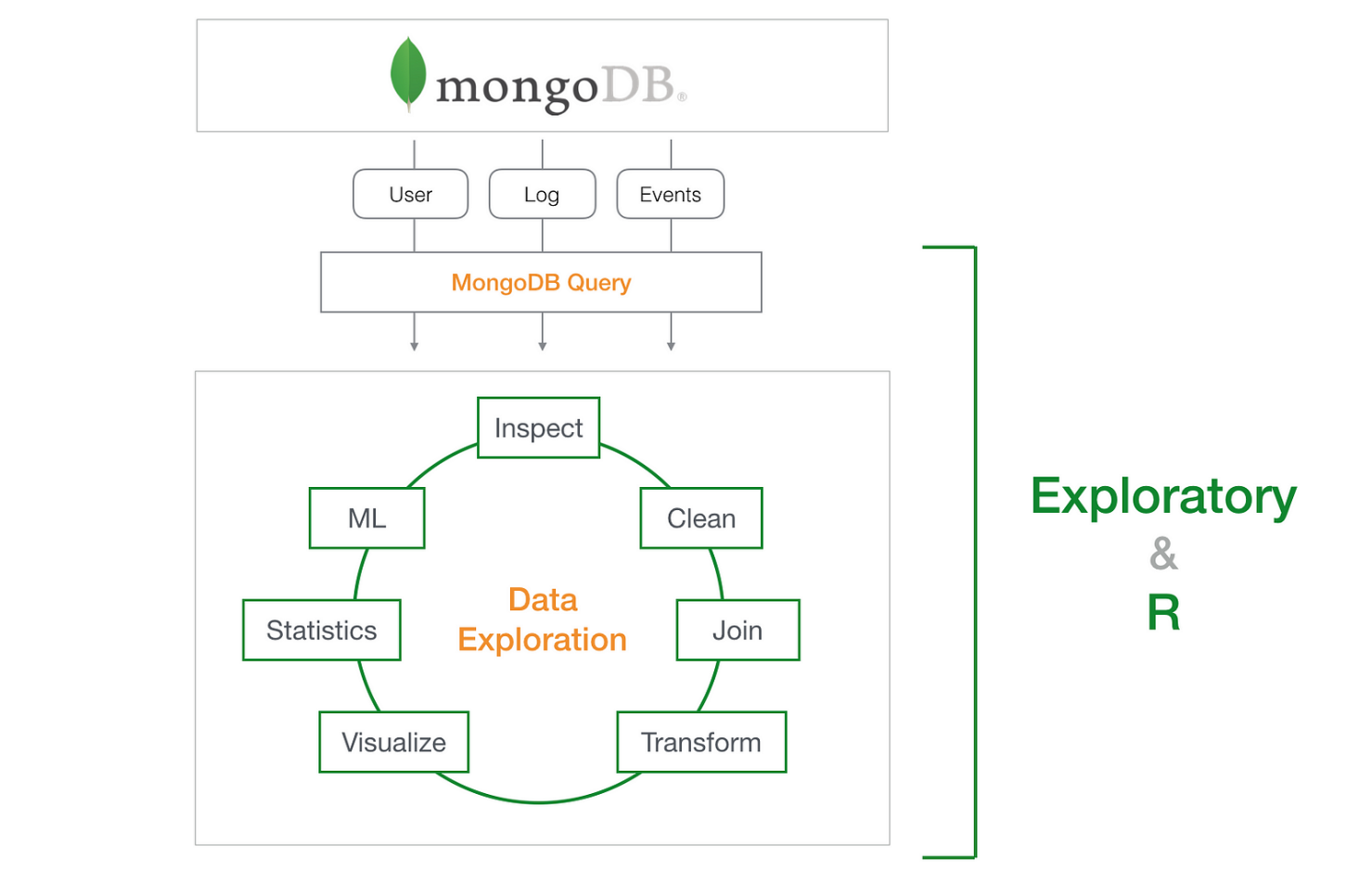
**public/**: Directory for static files (CSS, JavaScript).

# CHAPTER 7

# MONGO DB

MongoDB is a NoSQL, open-source database management system designed for handling large volumes of unstructured data. It stores data in flexible, JSON-like documents, making it suitable for modern web applications and real-time analytics.





**Figure 7.1** MongoDB Data Exploration Process

# Document-Oriented:

* MongoDB stores data in BSON (Binary JSON) format, allowing for flexible and hierarchical data structures**.**
* Documents are similar to JSON objects and can contain nested fields and arrays.

# Collections:

* Data is organized into collections, which are analogous to tables in relational databases.
* Collections hold multiple documents.

# Databases:

* A MongoDB instance can host multiple databases.
* Each database contains collections and their associated documents.

# Indexes:

* MongoDB uses indexes to improve query performance.
* Common index types include single-field, compound, and geospatial indexes.

# Basic MongoDB Operations:

1. Connecting to MongoDB
2. Creating and Using a Database
3. Creating a Collection
4. Inserting Documents
5. Querying Documents
6. Updating Documents
7. Deleting Documents
8. Deleting a Collection
9. Deleting a Database

# Aggregation Framework:

* Used for complex data processing and transformation.
* Examples include $match, $group, $sort, and $project.

# Indexing:

* Improve query performance.
* Example: Create an index on the name field

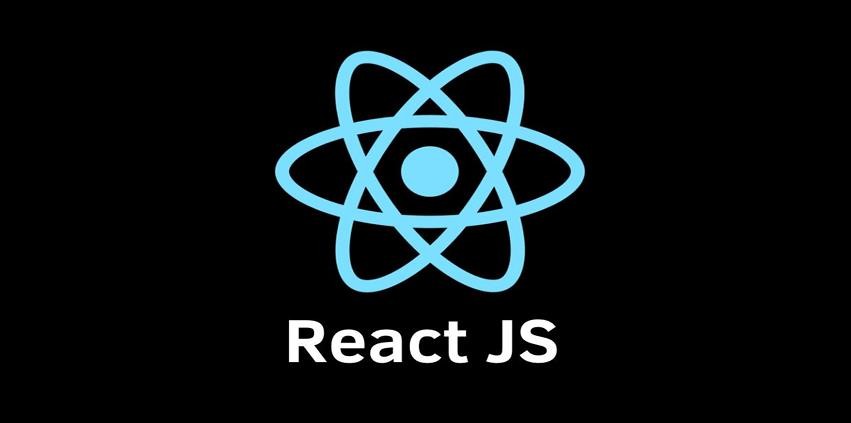
# Replication and Sharding:

* **Replication:** MongoDB supports replica sets, providing high availability by replicating data across multiple servers.
* **Sharding:** Distributes data across multiple servers to handle large datasets and high throughput.

# CHAPTER 8

# REACT JS

React.js is a popular JavaScript library developed by Facebook for building user interfaces, especially for single-page applications. It allows developers to create reusable UI components and manage the state and behavior of complex UIs efficiently.



**Figure 8.1** React js logo

# Components:

* + **Functional Components**

# Class Components

* + Functional Components: JavaScript functions that accept props (properties) and return React elements.
  + Class Components: ES6 classes that extend React.Component and have a render() method that returns React elements

Example:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

# 8.2 JSX (JavaScript XML):

* + A syntax extension for JavaScript that looks similar to HTML.
  + Allows embedding HTML-like elements in JavaScript code.

Example:

const element = <h1>Hello, world!</h1>;

# 8.3 State and Props:

* + **State:** Used to manage data within a component. It can be updated and causes the component to re-render.
  + **Props:** Read-only attributes passed from parent components to child components

# Lifecycle Methods (Class Components):

* + Methods that allow you to run code at specific points in a component’s life cycle(e.g.,componentDidMount,componentDidUpdate, componentWillUnmount).

# 8.5 Hooks (Functional Components):

* + Functions that let you use state and other React features without writing a class.
  + useState Hook
  + **useEffect Hook:** Performs side effects in function components (e.g., data fetching, subscriptions).

# 8.6 Event Handling:

* + React handles events in a way similar to handling events in DOM elements.

# 8.7 Conditional Rendering:

* + Render elements or components conditionally based on certain conditions.

# Lists and Keys:

* + Rendering lists of elements using JavaScript’s map() function.
  + Keys: A unique identifier for each element in a list to help React identify which items have changed.

# 8.9 Form Handling:

* + Managing form state and handling form submissions.

# 8.10 Routing:

* + Use libraries like react-router-dom to handle navigation between different views or pages.

**CHAPTER 9**

**PROJECT OVERVIEW: CALCULATOR WEB APP**

**9.1 Objective**

The Calculator Web App project aims to create a responsive, user-friendly calculator interface that performs basic mathematical operations. Built using React, it provides a fast and intuitive user experience, leveraging modern web development libraries and tools for enhanced interactivity and visual appeal.

**9.2 Core Features**

1. **Mathematical Calculations**
   * Supports fundamental arithmetic operations (addition, subtraction, multiplication, division).
   * Includes an expression evaluator for more complex calculations using expr-eval, allowing users to input and compute entire mathematical expressions in a single step.
2. **Responsive and Styled UI**
   * Uses bootstrap and font-awesome for a visually pleasing and responsive design.
   * Font Awesome icons enhance the UI, while Bootstrap provides a mobile-friendly, adaptive layout.
3. **Interactive Interface**
   * @testing-library/react, @testing-library/user-event, and @testing-library/jest-dom ensure a robust and well-tested interactive experience, offering reliable and predictable user interactions.
   * jquery allows for handling dynamic elements on the page and simplifying DOM manipulation tasks.
4. **Performance and Analytics**
   * web-vitals monitors and analyzes the app’s performance, providing metrics such as load time and responsiveness to enhance user experience.

**9.3 Technology Stack**

* **Frontend**: React, Bootstrap, Font Awesome
* **Logic and Calculation**: expr-eval for expression evaluation
* **Testing**: @testing-library/react, @testing-library/user-event, @testing-library/jest-dom
* **Performance Monitoring**: web-vitals

**9.4 User Experience and Design**

The calculator’s design focuses on clarity and ease of use, with a responsive layout and prominent buttons. The UI allows users to perform calculations smoothly on both desktop and mobile devices, with easy-to-read results and a clean interface.

**9.5 Future Enhancements**

Potential improvements could include additional advanced mathematical functions (trigonometric, exponential), history tracking for past calculations, and theme customization options.

**CHAPTER 10**

**SOURCE CODE FOR THE CALCULATOR WEB APP**

**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>

    );

  }

}

export default AppView;

**OUTPUT :**

****

**Figure 10.1** Output Screen

**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



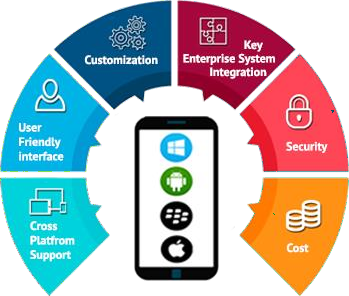
**Figure 11.2** Software Development Tech Stack Overview

# 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

# Company Values

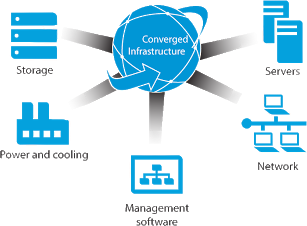
Kaashiv Infotech respects and seeks to maintain the highest standards of fairness, equality, integrity, and honesty. Our corporate philosophy is:

* + - 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

# 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.

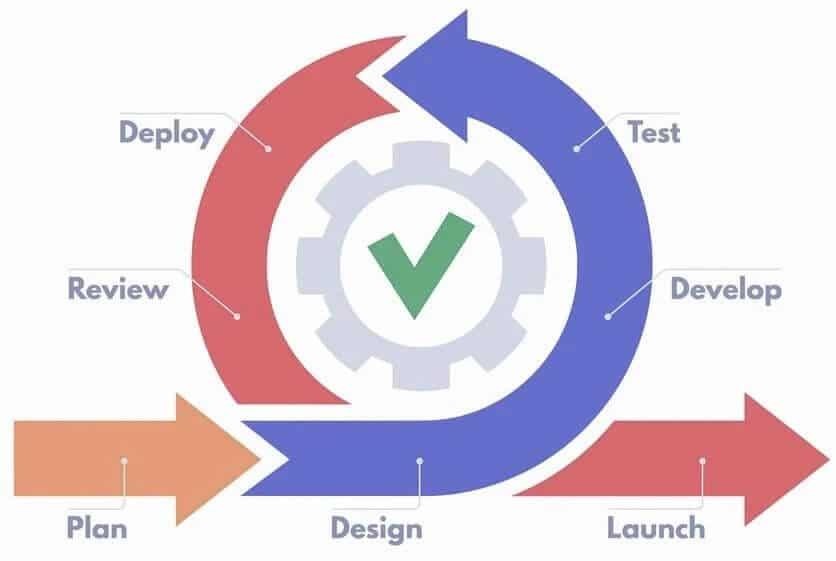
# Company Services

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

# CHAPTER 12

# CONCLUSION

**12.1 CONCLUSION**

Our pages would load faster and we would also be able to maintain the pages we have created with ease. Using the right resources for our **web design** jobs will make our **websites** contemporary. Now you have designed your very own web page. Not only have you learned the design aspects of web development, but you have also learned how to program in HTML. Through evaluating many different web sites, you are aware of what is effective and ineffective to viewers when designing a website too.

**12.2 REFERENCES**

1. **React Documentation**  
   *React – JavaScript library for building user interfaces.*  
   Available at: https://reactjs.org/docs/getting-started.html
2. **Bootstrap Documentation**  
   *Bootstrap – Front-end framework for responsive web design.*  
   Available at: https://getbootstrap.com/docs/4.4/getting-started/introduction/
3. **Font Awesome Documentation**  
   *Font Awesome – Icon toolkit used for visual icons.*  
   Available at: https://fontawesome.com/v4.7.0/icons/
4. **Expr-eval Documentation**  
   *Expr-eval – JavaScript library for evaluating mathematical expressions.*  
   Available at: <https://github.com/silentmatt/expr-eval>
5. **JQuery Documentation**  
   *JQuery – JavaScript library for simplifying HTML DOM tree traversal and manipulation.*  
   Available at: https://api.jquery.com/
6. **React Testing Library Documentation**

*@testing-library/react – Testing utilities for React applications.*  
Available at: https://testing-library.com/docs/react-testing-library/intro/

*@testing-library/user-event – Simulate user interactions in testing.*  
Available at: https://testing-library.com/docs/user-event/intro/

*@testing-library/jest-dom – Custom Jest matchers for testing DOM nodes.*  
Available at: <https://github.com/testing-library/jest-dom>

1. **Web Vitals Documentation**  
   *Web Vitals – Library for measuring web performance metrics.*  
   Available at: https://web.dev/vitals/
2. **Books**

***JavaScript:*** *The Definitive Guide* by David Flanagan. O'Reilly Media, 2020.  
Covers JavaScript fundamentals and advanced topics useful for handling UI logic in calculators.

***Eloquent JavaScript****: A Modern Introduction to Programming* by Marijn Haverbeke. No Starch Press, 2018.  
A great resource for understanding JavaScript essentials for building interactive web applications.

**Learning React**: Functional Web Development with React and Redux by Alex Banks and Eve Porcello. O'Reilly Media, 2020.  
A guide to the core concepts of React, covering Redux and the broader ecosystem.

**JavaScript**: The Definitive Guide by David Flanagan. O'Reilly Media, 2020.  
A comprehensive resource on JavaScript, including DOM manipulation, asynchronous programming, and JavaScript frameworks.

**React and React Native** by Adam Boduch and Roy Derks. Packt Publishing, 2020.  
A practical guide focusing on building scalable web and mobile applications using React and React Native.

**Fullstack React:** The Complete Guide to ReactJS and Friends by Accomazzo, Murray, and Lerner. Fullstack.io, 2017.  
Covers the complete development cycle for creating full-stack applications with React and integrating it with popular libraries.