



Scientific Calculator



Team Number: 13

PROJECT REPORT

**SUBJECT TITLE**: SOFTWARE ENGINEERING AND PROJECT MANGEMENT (SEPM)

**SUBJECT CODE**: 18CSC206J

**SUBMITTED BY**: Sparsh Vohra (RA1911028010031)

Swarnima Gupta (RA1911028010032)

Manav Garg (RA1911028010037)

**PROJECT NAME**: WEB APPLICATION BASED SCIENTIFIC CALCULATOR

**LANGUAGES USED**: HTML, CSS, Javascript, Bootstrap

**INTRODUCTION:**

Hereby using the basics of Web development we have made a **Scientific Calculator** which can do both basic and complex mathematical calculations like addition, multiplication , subtraction , division and complex calculations of trigonometry like sine , cosine , tangent functions and others like log , radian etc. Here we have applied Javascript to do the calculations part and the Frontend part is made of HTML and CSS .

**PROJECT DESCRIPTION**: Usually while working for long shifts of work where a lot concentration is needed a calculator is a much needed thing in the life of human which simplifies their work in day to day life. Every human uses calculator at least once in a day, so we can take a idea how much we are dependent on it.

So to make work easier and avoid any kind of distraction while working on PC we are making a web application based simple calculator which is fast and has a clean user interface that will help to users to deal with more complex expression in a easier way.

This calculator will make a tremendous change how people usually think the way calculators work and people will invest heavy amounts in this project which will motivate us to make more projects like this.

**BUSINESS NEED:** Anyone who uses this Scientific calculator can solve the expressions and complex algorithms with ease as its having a clean and fast interface.

**REQUIRED SKILLS:**

|  |  |
| --- | --- |
| Skills | More Info |
| UX Designer | Designing experience of user |
| Frontend Development | Design and Develop UI and frontend layer |
| Backend Development | Design Database and Develop Service / API |
| Testing | Develop Test Cases |
| Project Management | Project Planning, Scheduling, Executing, Monitoring and Controlling |

**METHODOLOGY USED AND WHY: Agile.**

The method that fits best for the project is the Agile method due to its various principles and functionalities that can makes the workflow easier. After deciding the method the next part is the stakeholders of the company which play a crucial role in the making of the project. Managing the stakeholders, evaluating their interests and influence.

**REQUIREMENTS:**

1. FUNCTIONAL REQUIREMENTS: Functional Requirements can also be expressed in the form of “user story” which is the smallest unit of work in an agile framework. It’s an end goal, not a feature, expressed from the software user’s perspective.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement (#) | Requirement Specification | Department | Name of Business User |
| E1FR1 | Calculation | Programmers | Client |
| E1FR2 | Data manipulation | Programmers | Client |
| E1FR3 | Business Process | Marketing Team | Client |
| E1FR4 | User Interaction | Designing Team | Consumers |

1. NON-FUNCTIONAL REQUIREMENTS: It includes Performance, Availability, Confidentiality, Scalability, Flexibility, Extensibility and Reliability.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement (#) | Category of NFR | Requirement Specification | Department | Name of Business User |
| NFR1 | Performance | All pages should load within 3 seconds | Backend Team | Clients |
| NFR2 | Performance | Search should bring the results less than 7 seconds | Backend Team | Clients |
|  | Availability | Application should be available for 24x7 | Backend Team | Clients |
|  | Scalability | Calculation Service should scale to serve 1000 request per second over 5 minutes time span | Backend Team | Consumers |
|  | Confidentiality | The source code should remain confidential | Programmers | Clients |
| E1NFR2 | Usability | The application can be used by everyone worldwide. | Project Team | Consumers |
|  | Flexibility | The application can be accessed anytime | Project Team | Consumers |
| E1NFR1 | Extensibility | The application can be used by anyone from any corner of the world. | Backend Team | Consumers |
|  | Reliability | The calculator will provide accurate and fast results | Backend Team | Consumers |

1. INFRASTRUCTURE REQUIREMENTS: It includes workspace, PCs’, internet and various kind of softwares.

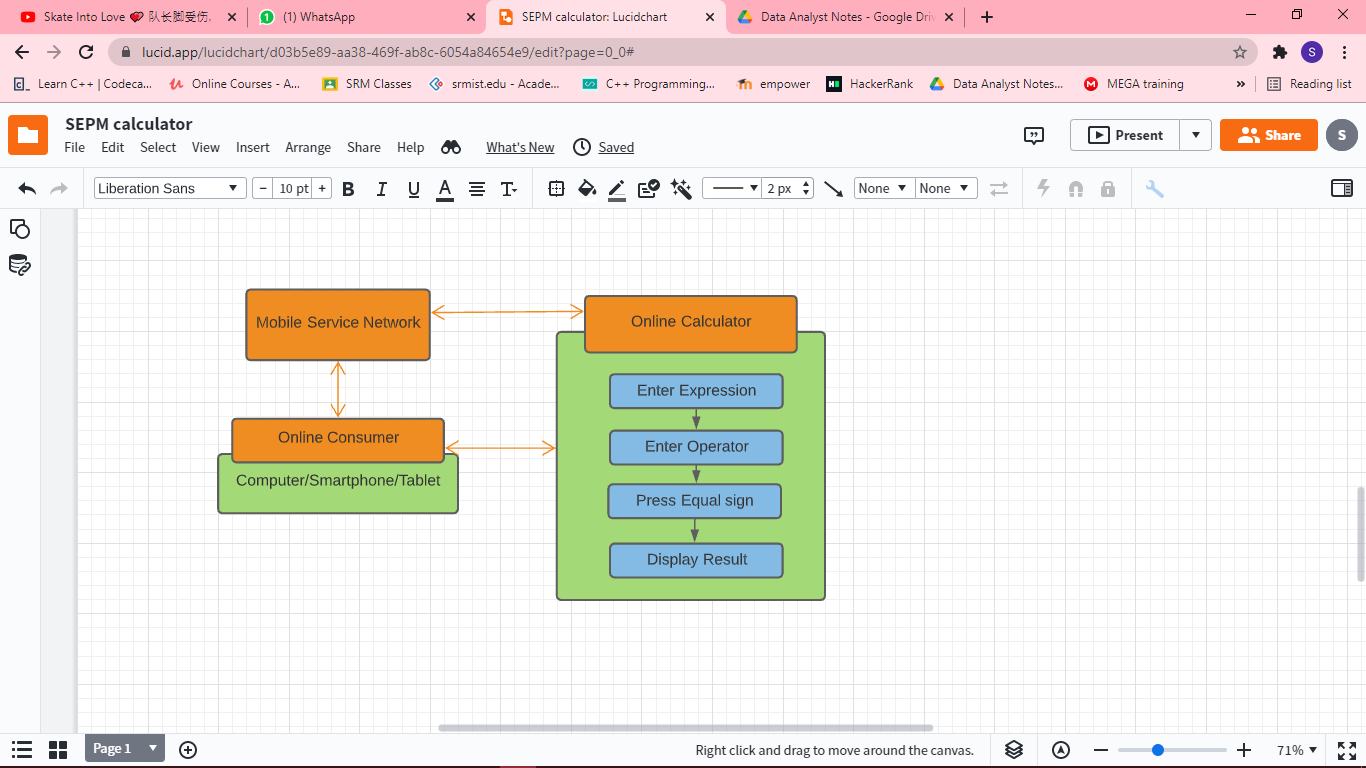
|  |  |  |
| --- | --- | --- |
| Requirement (#) | Requirement Specification | Department |
| IR1 | A supercomputer with 64GB Ram ,5TB Hard-disk and 8GB graphic card. | Programmers |
| IR2 | Code Repository: GitHub | Programmers |
| IR3 | AWS Account | Backend team |
| IR4 | IDE – Visual Studio Code | Programmers |

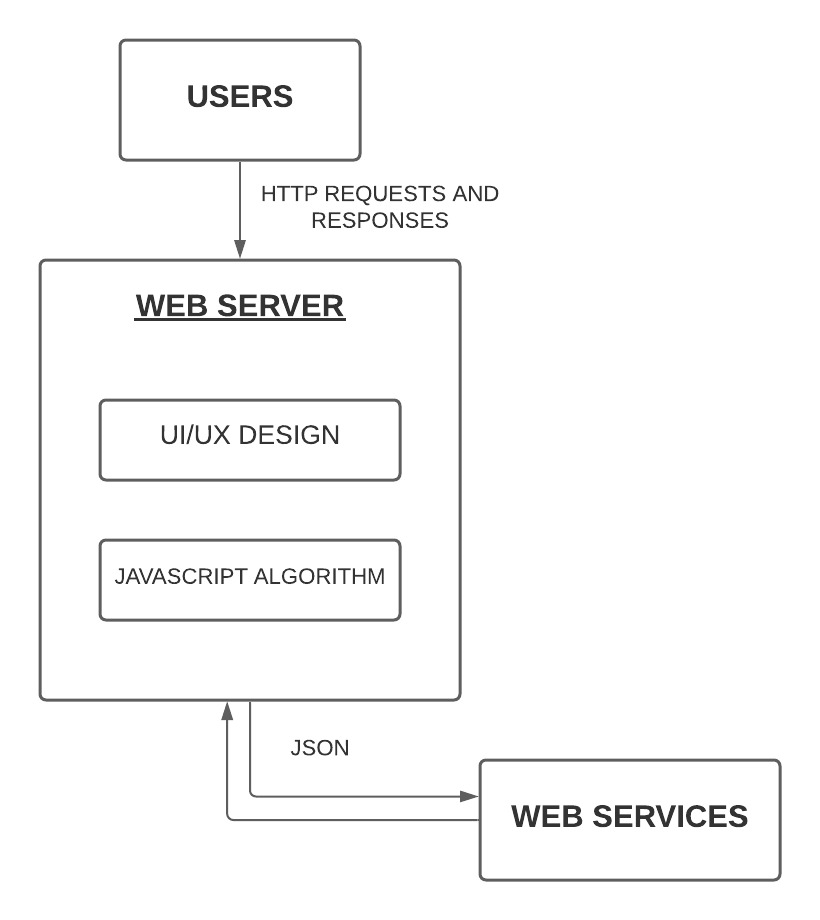
**DIFFERENT DIAGRAMS MADE:**

1. ARCHITECTURE DIAGRAM: In our project that is a “Web Based Online Calculator” the basic architecture contains of the UI design the backend code that calculates the expression and a server that stores the expressions as data.

Here the utility provider is the Internet that helps hosting the website and the consumers are the people that will be accessing the website using their mobile, tablets and the laptops.

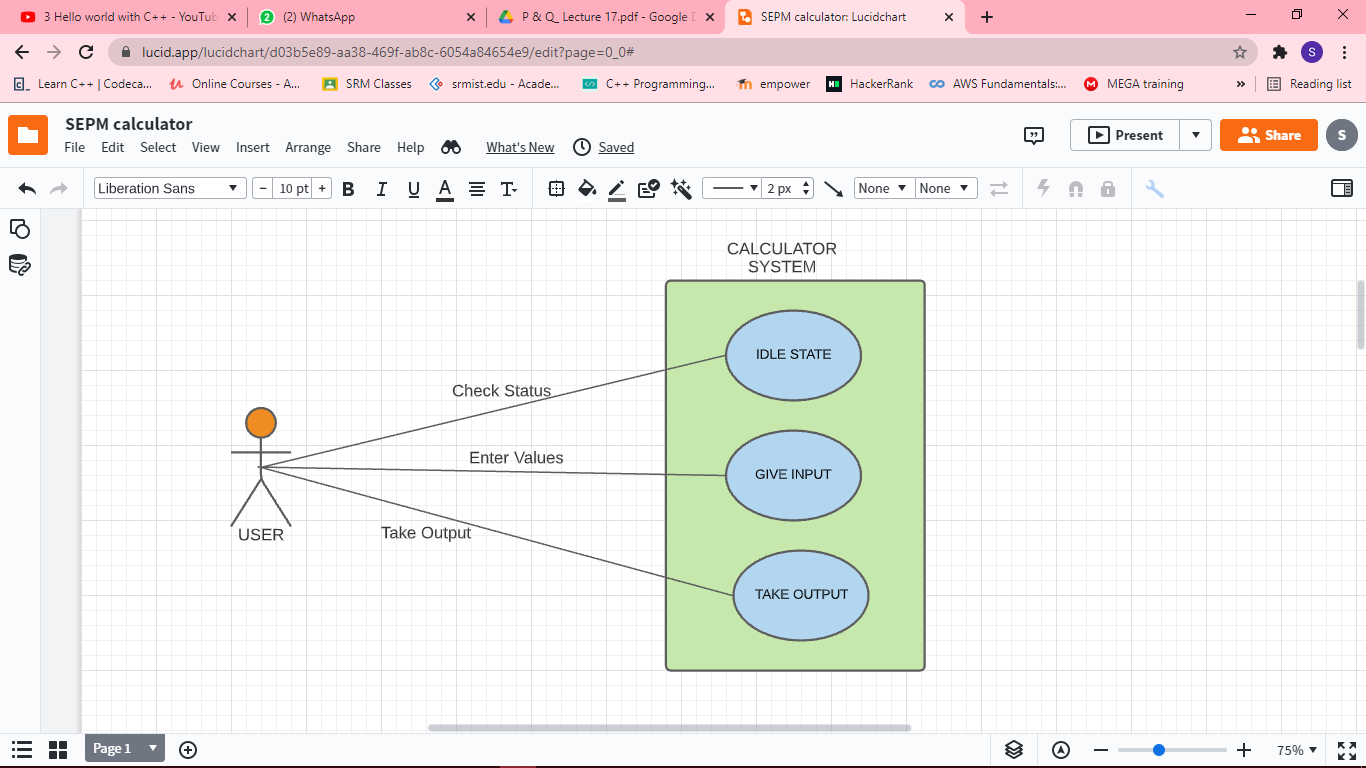
On the index page the user sees the basic layout of the calculator that a similar to the physical ones. The user enters the expression combination of numbers and operators and press the equals button which will trigger the logic code that will calculate the result and the result will be passed to the main page where it will be displayed to the user.

****

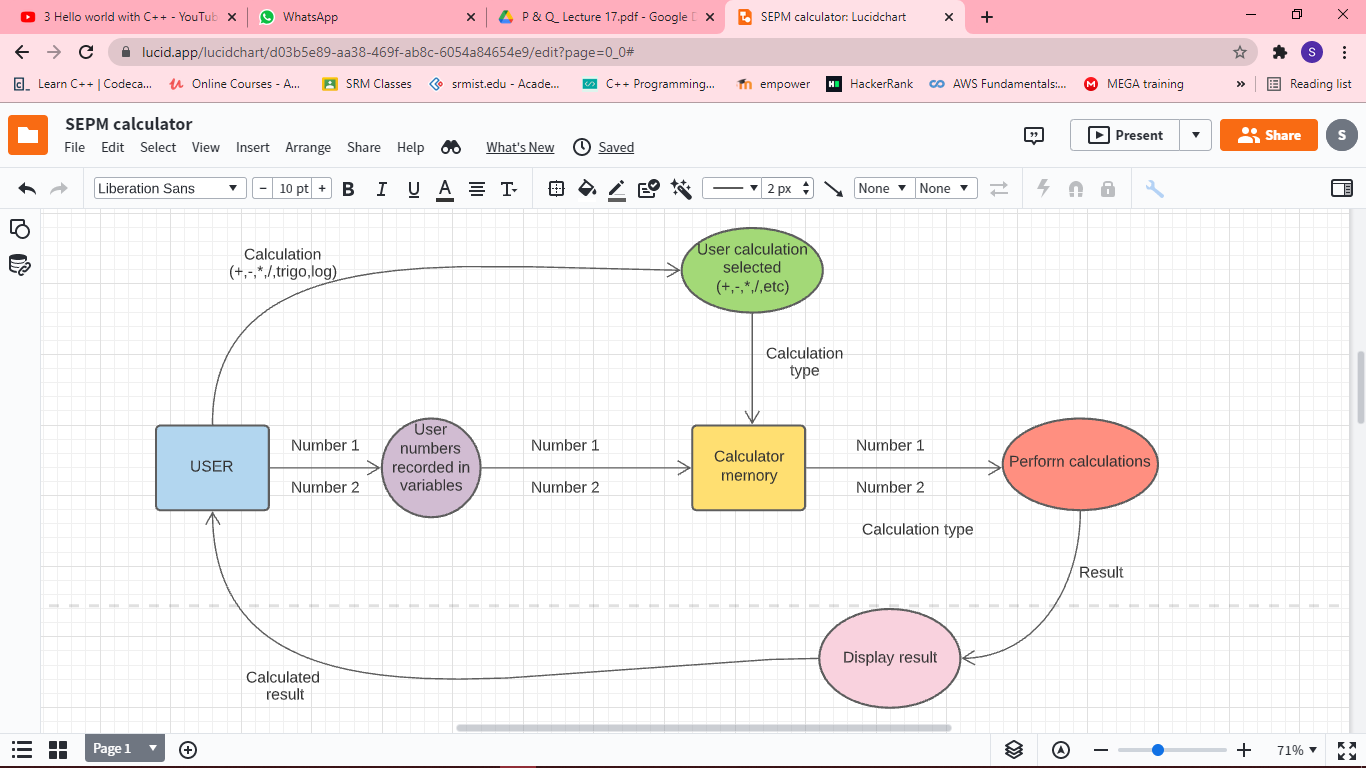
****

1. USE CASE DIAGRAM: Use cases are the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

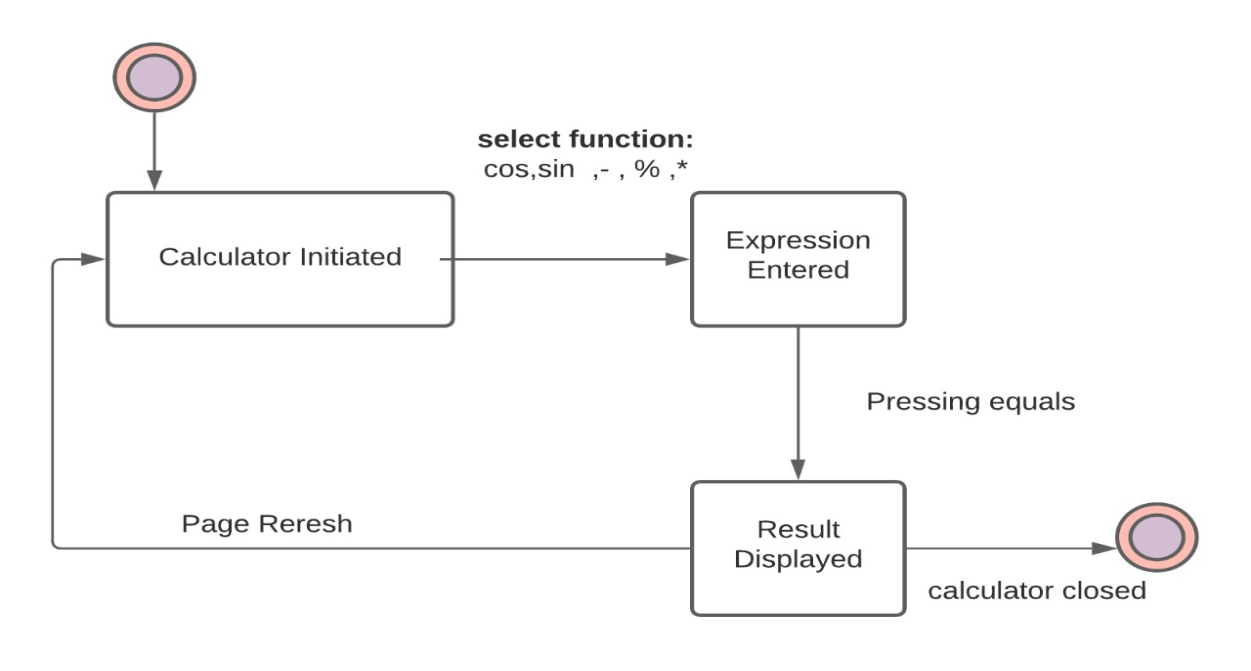
In our case here the actor will the user that will interact with the User Interface of the calculator and will be able to get various results based on the expression and operators he uses.



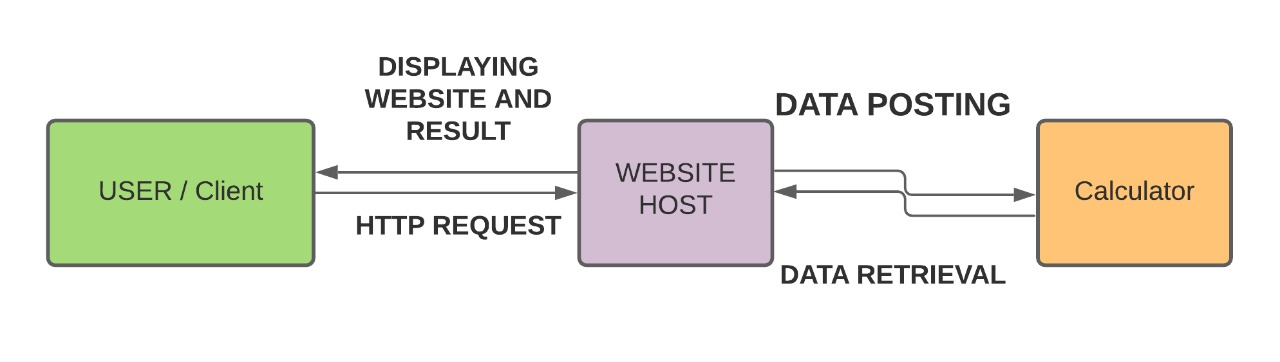
1. DATA FLOW DIAGRAMS: DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation. The data here will be entered by the user will be taken as input in a white box  from there the data will be run by the algorithm and the final result will be stores in a variable that will be displayed when the user will press the equals button.



1. STATE DIAGRAM: State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction which describes the behaviour of the system.

****

1. DEPLOYMENT DIAGRAM: A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system.

****

**MODULES**: We have divided our project into three modules:

* Module 1 consists the basic skeleton design of the calculator without any styling
* Module 2 consists the styling code of the calculator. It includes the positioning of the buttons, the color scheme etc.
* Module 3 consists the functionality code.

MODULE 1: To design the front-end or UI/UX part of the Calculator website. It includes all the buttons and the display screen that shows the calculations being done. This the raw design that does not include any design just the basic buttons with blank background.

**Code of Module 1**

<!DOCTYPE html>

<html lang="en" >

<head>

  <meta charset="UTF-8">

  <title>Scientific Calculator </title>

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normalize.min.css">

<link rel="stylesheet" href="style.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/prefixfree/1.0.7/prefixfree.min.js"></script>

</head>

<body>

<div id="app">

  <div class="calculator">

    <button @click="changeModeEvent" class="toggle-button">

      <p v-if="changeMode">Show Advanced Mode &nbsp; &nbsp; &#9864;</p>

      <p v-else>Show Basic Mode &nbsp; &nbsp; &#9862;</p>

    </button>

    <div class="results">

      <input class="input" v-model="current" />

    </div>

    <div class="mode" v-if="changeMode">

      <button class="button" @click="press">7</button>

      <button class="button" @click="press">8</button>

      <button class="button" @click="press">9</button>

      <button class="button" @click="press">\*</button>

      <button class="button" @click="press">&#60;=</button>

      <button class="button" @click="press">C</button>

      <button class="button" @click="press">4</button>

      <button class="button" @click="press($event)">5</button>

      <button class="button" @click="press">6</button>

      <button class="button" @click="press">/</button>

      <button class="button" @click="press">(</button>

      <button class="button" @click="press">)</button>

      <button class="button" @click="press">1</button>

      <button class="button" @click="press">2</button>

      <button class="button" @click="press">3</button>

      <button class="button" @click="press">-</button>

      <button class="button" @click="press">x ²</button>

      <button class="button" @click="press">±</button>

      <button class="button" @click="press">0</button>

      <button class="button" @click="press">.</button>

      <button class="button" @click="press">%</button>

      <button class="button" @click="press">+</button>

      <button class="button equal-sign" @click="press">=</button>

    </div>

    <div class="mode" v-else>

      <button class="button" @click="press">sin</button>

      <button class="button" @click="press">cos</button>

      <button class="button" @click="press">tan</button>

      <button class="button" @click="press">x^</button>

      <button class="button" @click="press">&#60;=</button>

      <button class="button" @click="press">C</button>

      <button class="button" @click="press">log</button>

      <button class="button" @click="press">ln</button>

      <button class="button" @click="press">e</button>

      <button class="button" @click="press">∘</button>

      <button class="button" @click="press">rad</button>

      <button class="button" @click="press">√</button>

      <button class="button" @click="press">7</button>

      <button class="button" @click="press">8   </button>

      <button class="button" @click="press">9</button>

      <button class="button" @click="press">/</button>

      <button class="button" @click="press">x ²</button>

      <button class="button" @click="press">x !</button>

      <button class="button" @click="press">4</button>

      <button class="button" @click="press">5</button>

      <button class="button" @click="press">6</button>

      <button class="button" @click="press">\*</button>

      <button class="button" @click="press">(</button>

      <button class="button" @click="press">)</button>

      <button class="button" @click="press">1</button>

      <button class="button" @click="press">2</button>

      <button class="button" @click="press">3</button>

      <button class="button" @click="press">-</button>

      <button class="button" @click="press">%</button>

      <button class="button" @click="press">±</button>

      <button class="button" @click="press">0</button>

      <button class="button" @click="press">.</button>

      <button class="button" @click="press">&#x003C0;</button>

      <button class="button" @click="press">+</button>

      <button class="button equal-sign" @click="press">=</button>

    </div>

  </div>

</div>

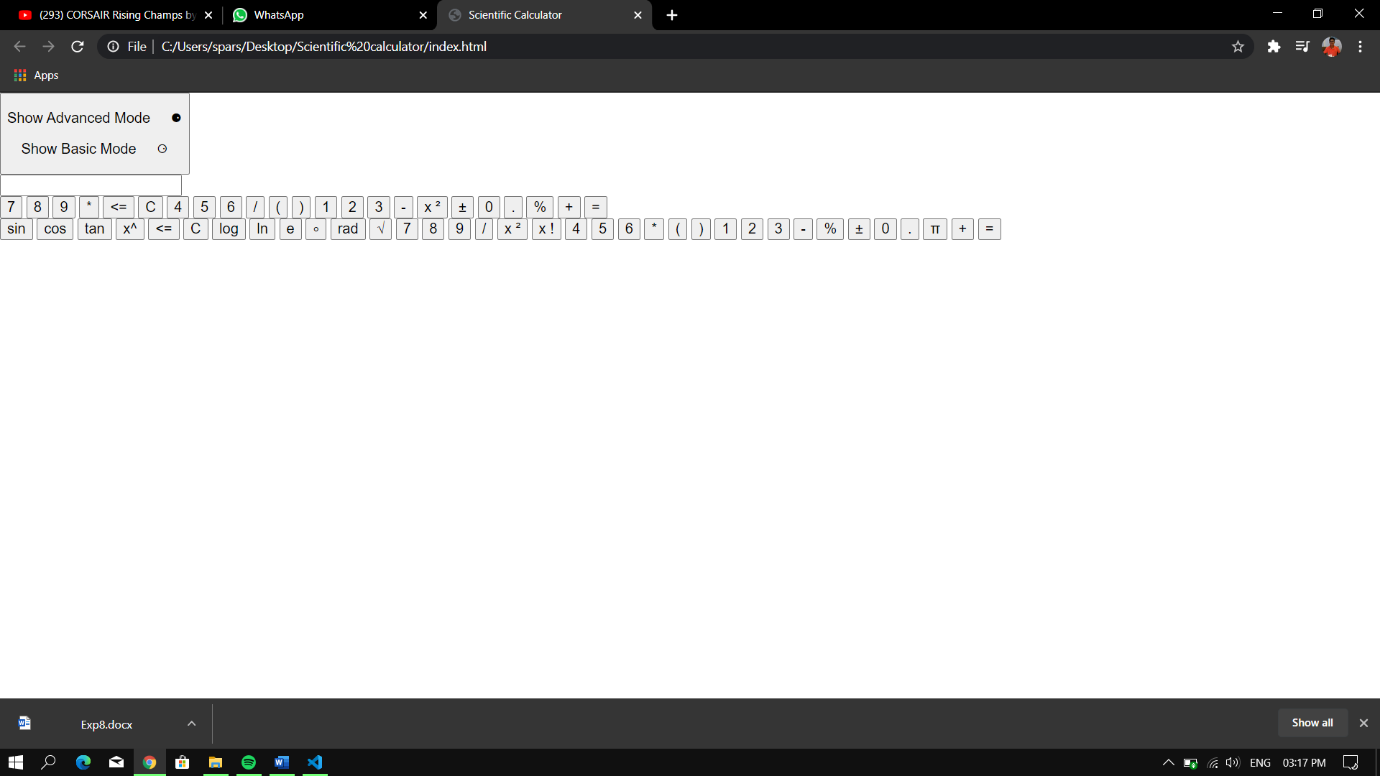
<!-- partial -->

  <script src='https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.min.js'></script><script  src="script.js"></script>

</body>

</html>

**Result of Module 1**

****

Module 2**:** To implement the styling of the frontend UI Design. The code here written is in pure CSS. The CSS takes care of structure and positioning components.

**Code of Module 2: CSS Styling**

body {

  background: linear-gradient(to right, #85D8CE, #085078);

}

#app {

  font-family: 'Avenir', Helvetica, Arial, sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  text-align: center;

  color: #2c3e50;

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

  align-item: center;

}

.calculator {

  width: 440px;

  padding: 20px;

  border-radius: 5px;

  margin: 20px auto;

  font-size: 16px;

  background-color: #333333;

}

.input {

  width: 420px;

  height: 50px;

  border-radius: 0px;

  border: 1px solid black;

  background-color: #333333;

  color: #d9d9d9;

  padding: 0 5px 0 5px;

  margin: 0 0px 10px 0px;

  font-size: 30px;

}

.input:focus,

.input:active {

  border-color: #03a9f4;

  box-shadow: 0 0 4px #03A9F4;

  outline: none 0;

}

.button {

  margin: 3px;

  width: 63px;

  border: 1px solid #0d0d0d;

  height: 30px;

  border-radius: 4px;

  color: #d9d9d9;

  background-color: #1a1a1a;

  cursor: pointer;

  outline: none;

}

.mode {

  display: flex;

  flex-wrap: wrap;

  justify-content: space-evenly;

}

.equal-sign {

  background-color: green;

  width: 133px;

}

.toggle-button {

  border: none;

  background-color: #333333;

  cursor: pointer;

  outline: none;

  font-size: 1rem;

  color: #fff;

  text-shadow: -1px -1px 0 rgba(0, 0, 0, 0.35);

}

p {

  margin-top: 0;

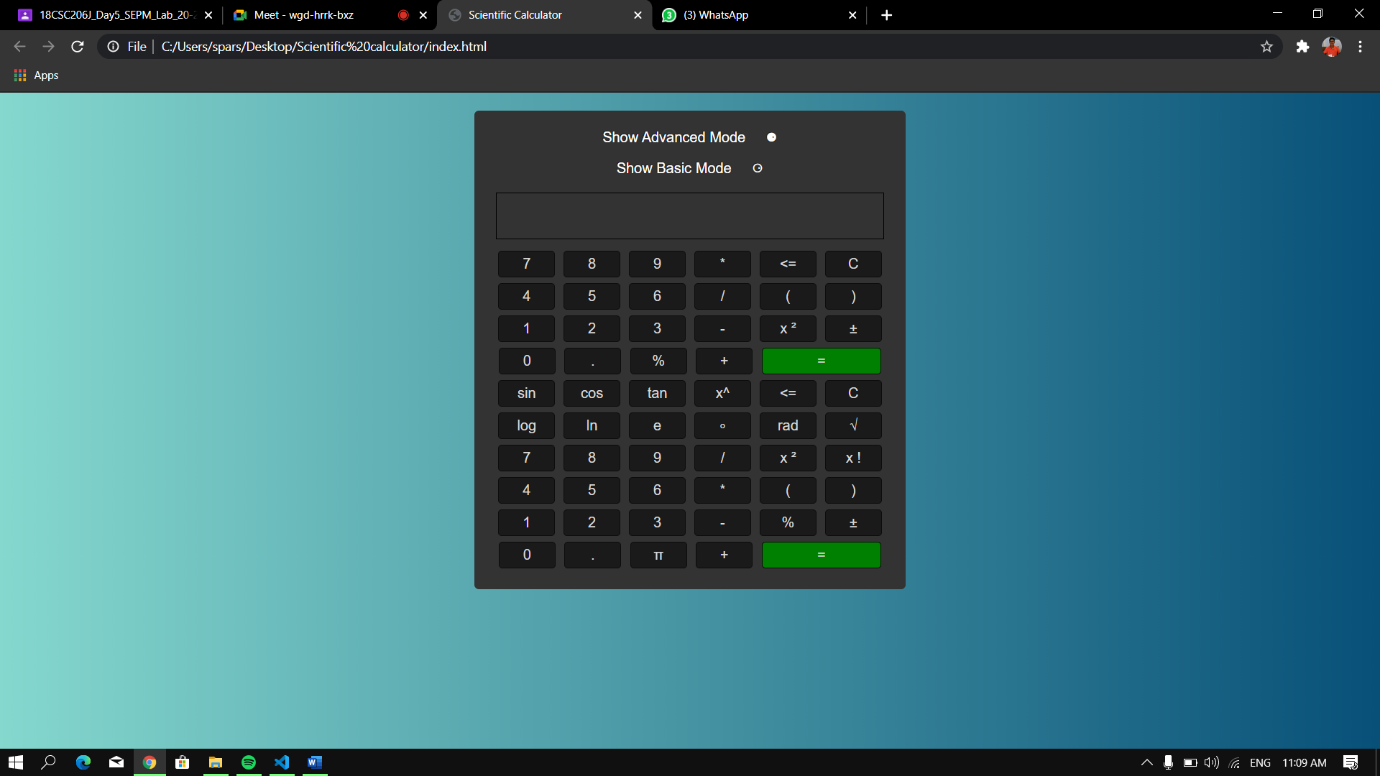
}

button::-moz-focus-inner {

  border-color: transparent;

}

**Result of Module 2**

****

**MODULE 3**: To add the functionality to the Scientific calculator.

**Code of Module 3:**

let app = new Vue({

  el: '#app',

  data() {

    return {

      current: '',

      changeMode: true };

  },

  methods: {

    press: function (event) {

      let me = this;

      let key = event.target.textContent;

      if (

      key != '=' &&

      key != 'C' &&

      key != '\*' &&

      key != '/' &&

      key != '√' &&

      key != "x ²" &&

      key != "%" &&

      key != "<=" &&

      key != "±" &&

      key != "sin" &&

      key != "cos" &&

      key != "tan" &&

      key != "log" &&

      key != "ln" &&

      key != "x^" &&

      key != "x !" &&

      key != "π" &&

      key != "e" &&

      key != "rad" &&

      key != "∘")

      {

        me.current += key;

      } else if (key === '=') {

        if (me.current.indexOf('^') > -1) {

          let base = me.current.slice(0, me.current.indexOf('^'));

          let exponent = me.current.slice(me.current.indexOf('^') + 1);

          me.current = eval('Math.pow(' + base + ',' + exponent + ')');

        } else {

          me.current = eval(me.current);

        }

      } else if (key === 'C') {

        me.current = '';

      } else if (key === '\*') {

        me.current += '\*';

      } else if (key === '/') {

        me.current += '/';

      } else if (key === '+') {

        me.current += '+';

      } else if (key === '-') {

        me.current += '-';

      } else if (key === '±') {

        if (me.current.charAt(0) === '-') {

          me.current = me.current.slice(1);

        } else {

          me.current = '-' + me.current;

        }

      } else if (key === '<=') {

        me.current = me.current.substring(0, me.current.length - 1);

      } else if (key === '%') {

        me.current = me.current / 100;

      } else if (key === 'π') {

        me.current = me.current \* Math.PI;

      } else if (key === 'x ²') {

        me.current = eval(me.current \* me.current);

      } else if (key === '√') {

        me.current = Math.sqrt(me.current);

      } else if (key === 'sin') {

        me.current = Math.sin(me.current);

      } else if (key === 'cos') {

        me.current = Math.cos(me.current);

      } else if (key === 'tan') {

        me.current = Math.tan(me.current);

      } else if (key === 'log') {

        me.current = Math.log10(me.current);

      } else if (key === 'ln') {

        me.current = Math.log(me.current);

      } else if (key === 'x^') {

        me.current += '^';

      } else if (key === 'x !') {

        let number = 1;

        if (me.current === 0) {

          me.current = '1';

        } else if (me.current < 0) {

          me.current = NaN;

        } else {

          let number = 1;

          for (let i = me.current; i > 0; i--) {

            number \*= i;

          }

          me.current = number;

        }

      } else if (key === 'e') {

        me.current = Math.exp(me.current);

      } else if (key === 'rad') {

        me.current = me.current \* (Math.PI / 180);

      } else if (key === '∘') {

        me.current = me.current \* (180 / Math.PI);

      }

    },

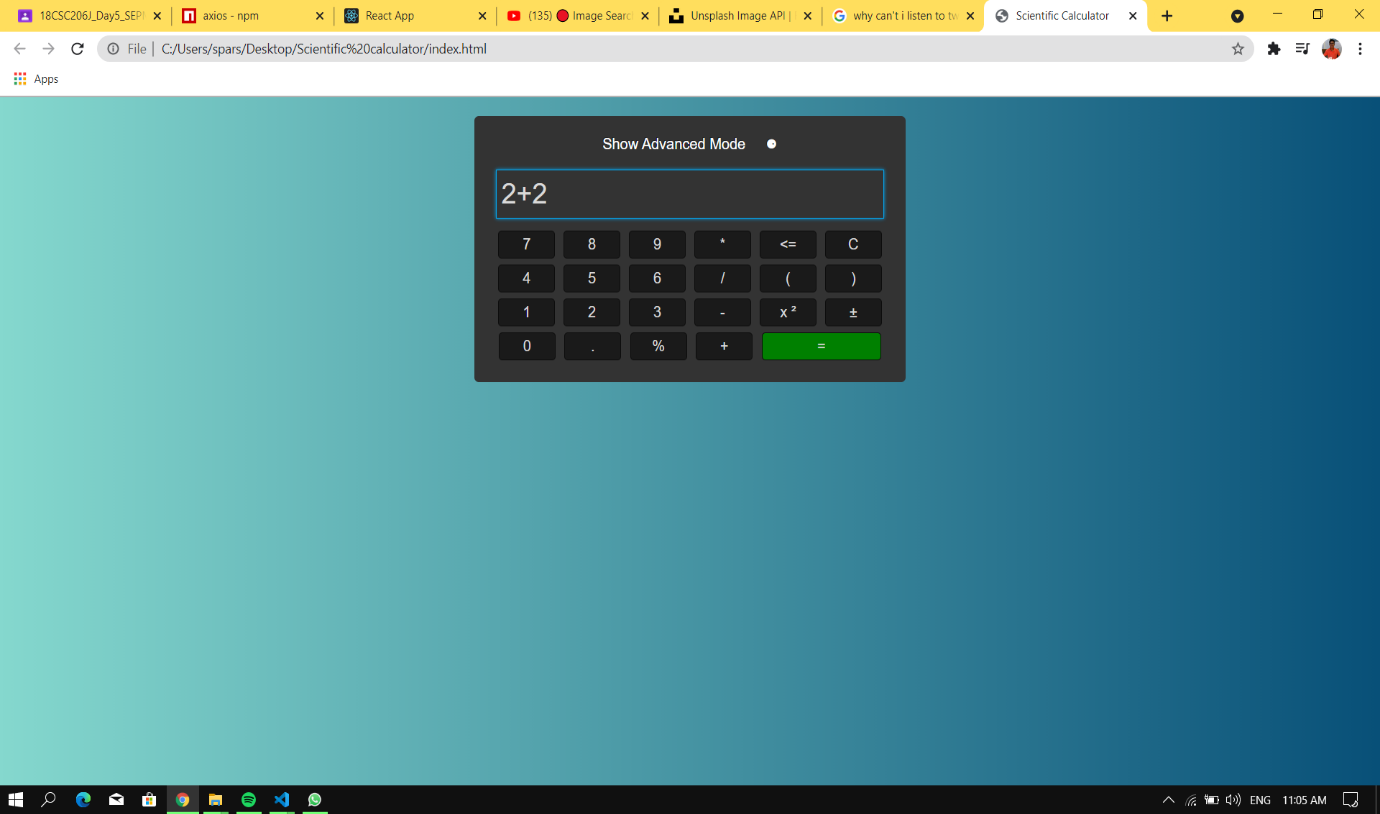
    changeModeEvent: function () {

      let me = this;

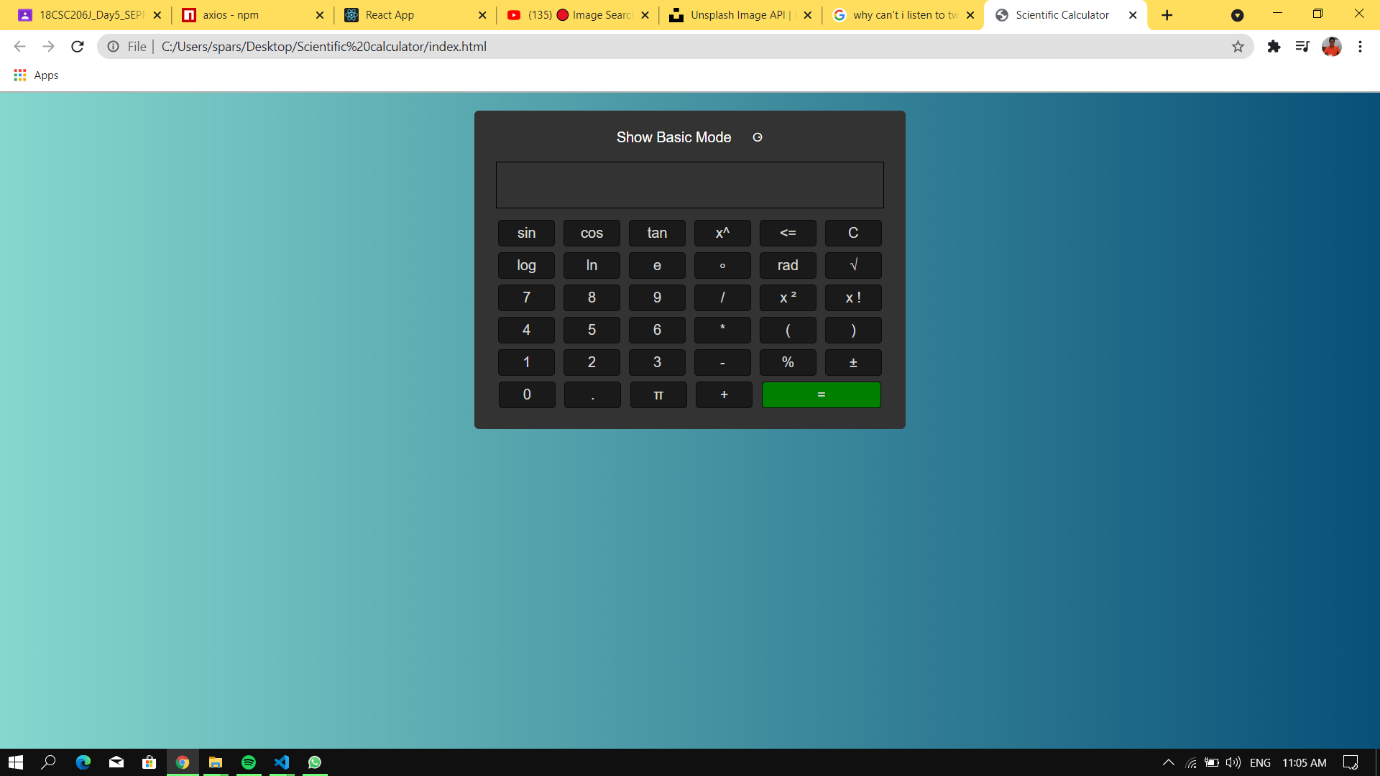
      me.changeMode = !me.changeMode;

    } } });

**Result of Module 3:**

**Simple Calculator**

**Scientific Calculator**

****

**Types of Testing:**

|  |  |  |
| --- | --- | --- |
| Category | Methodology | Tools Required |
| Design Response | Manual | Internet Connection |
| Load and Traffic | Manual | Internet Connection |
| Buffer Time | Manual | Internet Connection |

**Testing:**

The plan involves testing all modules working built together checking the efficiency of the entire product as whole if they are working accordingly to they were build. There are different aspects of testing the modules and product as a whole depending on methods that can be used.

There are basically two types of test cases namely:

1. Functional Test cases
2. Non-Functional Test cases

**1.Functional Test Cases:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test ID (#)** | **Test Scenario** | **Test Case** | **Execution Steps** | **Expected Outcome** | **Actual Outcome** | **Status** | **Remarks** |
|  | Verify that the answer is displayed | One the equals button in clicked the answer is displayed | 1.User enter the combination of digits and operators   1. 2.Clicks the equals button | The answer should be displayed on the output screen | The answer is displayed correctly | Pass | Success |
|  | Verify each and every button is working | On click, display screen shows what you have clicked | Click on the digits button and operator buttons | The number and operators should be displayed | The digits and operators are displayed | Pass | Success |

**2.Non-Functional Test Cases:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test ID (#)** | **Test Scenario** | **Test Case** | **Execution Steps** | **Expected Outcome** | **Actual Outcome** | **Status** | **Remarks** |
|  | How many users can access the calculator at one time | Load test | Login with multiple devices | Site Working | Same as expected | Pass | Success |
|  | Checking the latency | Buffer test | Click on different buttons one after another | Efficiently working | Same as expected | Pass | Success |
|  | Overall Functionality | Stress test | Check on how all NFR are working all together | Good efficiency | Same as Expected | Pass | Success |

**Test Report:**

|  |  |  |
| --- | --- | --- |
| **Category** | **Progress Against Plan** | **Status** |
| Functional Testing | Green | Completed |
| Non-Functional Testing | Green | Completed |

|  |  |  |
| --- | --- | --- |
| **Functional** | **Test Case Coverage (%)** | **Status** |
| Module 1 | 30% | Completed |
| Module 2 | 60% | Completed |

**Link to Submitted Documents:** [**https://drive.google.com/drive/u/1/folders/1xTKb6xD\_dKbjdf9vuTADfbpdjXE4JDeh**](https://drive.google.com/drive/u/1/folders/1xTKb6xD_dKbjdf9vuTADfbpdjXE4JDeh)

**Conclusion:**

The project has been successfully completed by having established a user friendly interface with the help of HTML and CSS .It consists of a basic and scientific calculator . At the same time there is some scope of improvement in the future. It can be possible to make it more user friendly and attractive by adding a variety of functions to it. If we are able to introduce user defined functions then it will give user what he wants, that will ultimate success to our attempts