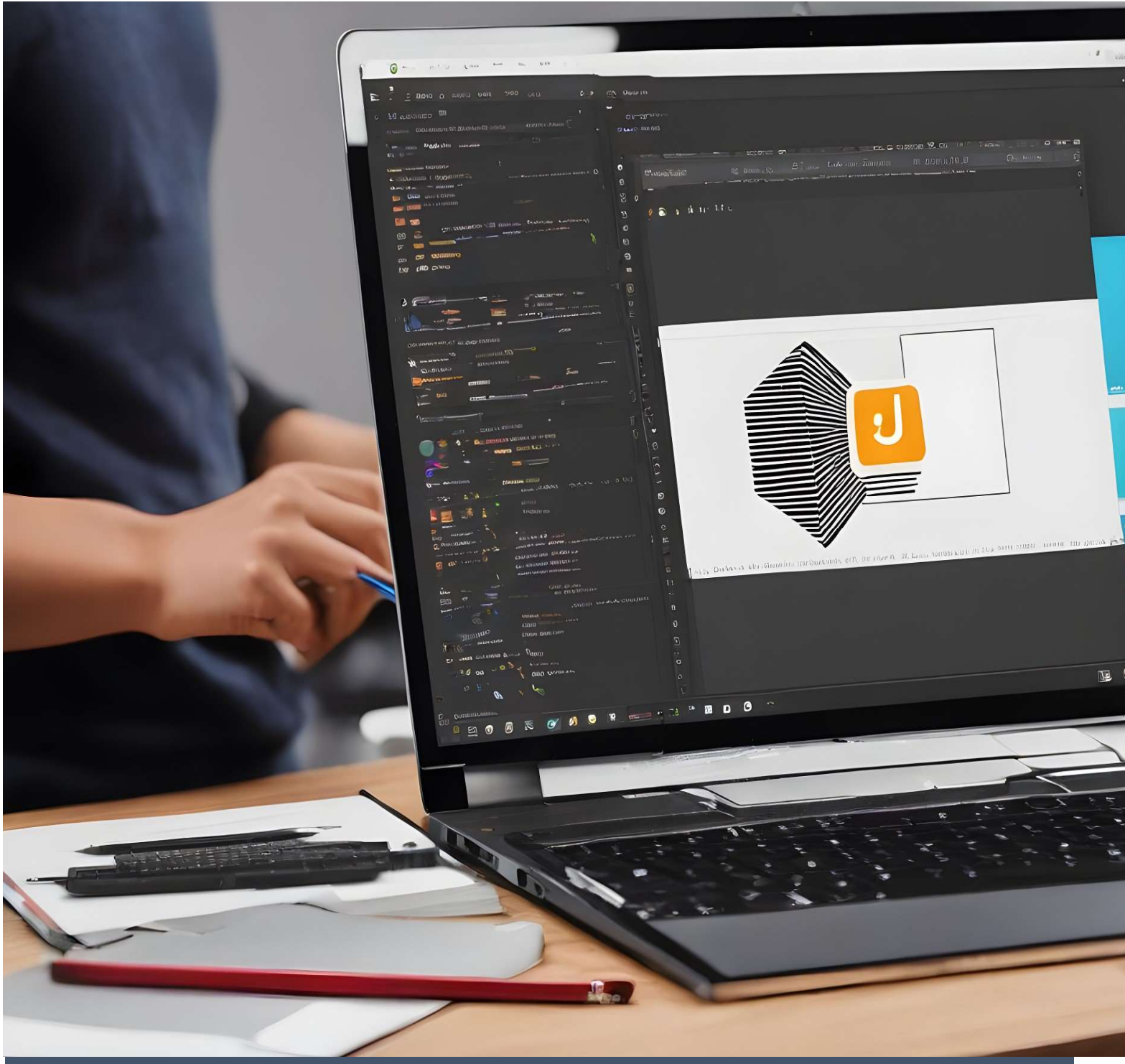


Lab Manual: Full Stack Development Lab

[FSDL – 2021-25]

Dr. Kailash Shaw

TE7942

Vision of Institute

To be a center of excellence for creation and dissemination of knowledge by imparting life skills and experiential learning for a promising future in the areas of engineering and technology.

Mission of Institute

- To promote professional ethics and experiential learning for better employability.
- To contribute towards knowledge generation and dissemination in the field of engineering.
- To address societal problems by promoting research, innovation and entrepreneurship.
- To develop global competencies amongst students by fostering value-based education.
- To strengthen industrial, Institutional, and international collaborations for synergetic relations.

Vision of Department

To impart quality education with research insights for developing competent global engineers in the field of Artificial Intelligence and Machine Learning to solve societal problems.

Mission of Department

- To educate students on cutting-edge AIML technologies with strong industry connections to develop problem-solving capabilities, leadership, and teamwork skills.
- To produce quality research through national and international collaborations leading to publications, IPR, and sponsored/funded projects.
- To inculcate professional values with lifelong learning through curricular and co-curricular activities and create globally-aware citizens.

PROGRAM OUTCOMES (PO):

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs):

1. To apply the concepts of Artificial Intelligence and Machine Learning with practical knowledge in analysis, design and development of intelligent systems and applications to multi-disciplinary problems
2. To provide a concrete foundation to the students in the cutting edge areas Artificial Intelligence and Machine Learning and excelling in the specialized areas like Natural Language Processing, Computer Vision, Reinforcement Learning, Internet of Things, Cloud computing, Data Security and privacy etc.

GENERAL LABORATORY INSTRUCTIONS

1) Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2) Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3) Student should enter into the laboratory with:
    a. Laboratory Record updated up to the last session experiments.
    b. Proper Dress code and Identity card.
4) Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
5) All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
6) Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
7) Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
8) Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
9) Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Lab Duration: 30 hours

Lab Objectives

1. To construct basic websites using HTML and Cascading Style Sheets.
2. To build dynamic web pages with validation using Java Script objects and by applying different event handling mechanisms.
3. To develop modern interactive web applications using advanced web and database technologies.

Lab Outcome

CO 1-Web Design Proficiency: Students will acquire the ability to design responsive webpages with a registration form using HTML and CSS, demonstrating an understanding of fundamental design principles and layout techniques.

CO 2-CSS Styling Mastery: students will showcase proficiency in applying various CSS styles and layouts, allowing them to create visually appealing and well-styled web pages.

CO 3-Bootstrap Implementation Skills: Through the card-flip effect task, students will gain hands-on experience in implementing Bootstrap components, enhancing their skills in leveraging popular front-end frameworks.

CO 4-JavaScript Interactivity: Students will be able to use JavaScript to add interactivity to web pages, including the implementation of pop-up boxes (alert, confirm, prompt) and event handling on form elements, thereby enhancing the user experience and functionality of their web applications.

CO 5-React.js and Node.js Competence: By building interactive interfaces with React components, implementing routing in React JS, and developing a web application with Node.js for NO SQL database interaction, students will achieve proficiency in modern web development technologies.

Lab Equipment and Software:

1) Computers with the required programming environment (e.g., Text Editor, Google Chrome, Node.js, React.js, MongoDB, Node Package Manager).
2) Integrated Development Environment (IDE) such as Visual Studio Code, Eclipse, or Code, Blocks, DEV C++. (Recommended DEV C++)
3) Sample input/output files for testing purposes.

Lab Activities:

Experiment 1: Design a Webpage for Registration Form

Experiment 2: Design a responsive webpage for a registration form using HTML and CSS.

Experiment 3: Implement a card-flip effect using Bootstrap.

Experiment 4: Use JavaScript to display pop up boxes alert box, an alert box with line breaks, confirm box, prompt box

Experiment 5: Create a registration form and demonstrate the event handling on form elements.

Experiment 6: Use React components to build interactive interfaces.

Experiment 7: Implement routing in React JS.

Experiment 8: Create a database from .json file and execute NO SQL Queries.

Experiment 9: Write a web application with Node.js to insert a document/record in NO SQL document based database collection.

Lab Report:

- Each student should submit a lab report documenting their implementation, including code snippets and output screenshots.
- The report should include a description of the implemented algorithms.
- Students should provide a detailed explanation of the testing performed and the results obtained.
- The report should also include any challenges faced during the lab and their solutions.

Lab Evaluation:

- **Students will be evaluated based on their implementation of the required algorithms.**
- **The correctness and efficiency of the implemented solutions will be assessed.**
- **The quality and completeness of the lab report will also be considered for evaluation.**

Assessment	Marks	Total
Continuous Assessment (CA)	10	25
End Sem Exam (ESE)	15	

Continuous Assessment (CA):

Each Lab will be evaluated based on following Rubrics. Finally, all the marks should be scale down to maximum 10 only.

Lab Performance for each experiment				Viva	Total
Timely Lab report Submission	Correctness of Code	Code Quality	Testing and Validation	Based on the Experiment Performed	
2	6	4	2	6	20

Timely Lab Report Submission (10%): If a student fails to submit the handwritten report by the specified due date, a penalty will be imposed.

Submission *2 pts* Timely submitted document as per requirement.	late within 3 days of dead line *1 pt* Submitted but late by 3 days	Late after 3 days *0 pts* late submitted after 3 days

The marks for report submission will decrease by 1 marks if submitted within next 3 days, and after 3 days you will get 0 marks. However, it is still mandatory for the student to submit the report, as it is crucial for claiming other marks allocated to the laboratory. Failure to submit the report will result in a total score of 0 for the entire lab. In addition to the handwritten notes, students are required to submit a single PDF copy containing the executable code, description, and output screen to the designated Google Classroom shortly after the experiment concludes. **Note: It is compulsory for you to add your name and PRN in the top left corner of each page of the report. Additionally, please be aware that in case of plagiarized code, no marks will be awarded.**

Correctness of Code (30%): Marks will be awarded based on

best *6 pts*	Very Good *5 pts*	Good *4 pts*	Average *3 pts*	Satisfactory *2 pts*	Poor *1 pt*
1. Get the expected output as per requirement	1. Get the expected output with minor issues	1. Getting the expected output but look and feel is not proper	Getting the output only. Requirement is not fulfilled.	Only output is visible with poor look and filled.	Output is visible with many element is missing or not correctly working as per requirement.

Code Quality (20%):

Best *4 pts*	Good *3 pts*	Average *2 pts*	Poor *1 pt*
• Follows proper coding conventions (e.g., indentation, variable naming). • Includes comments for better code understanding.	• Not following all proper coding conventions (e.g., indentation, variable naming). • Includes comments but is not understood	• Coding convention not followed • comments are not available or do not provide any any information.	only code is available but poorly defined

Testing and Validation (10%):

Good *2 pts*	Poor *1 pt*
• Tests the program with various test cases, including edge cases.	Validation of few elements only

Viva (30%): Based on the Experiment performed. Total 6 question will be asked; total points depends on the number of question correctly answer.

End Sem Exam (ESE): To be evaluated at the end of semester

Lab Performance				**Viva**	**Total**
Timely Completion of Code with write Up	**Correctness of Code**	**Code Quality**	**Testing and Validation**	**Based on the Experiment Performed**	
2	**3**	**3**	**2**	**5**	**30**

Experiment 1

What is HTML?

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

A Simple HTML Document

<!DOCTYPE html> <html> <head> <title>Page Title</title> </head> <body> My First Heading My first paragraph. </body> </html>	• The <!DOCTYPE html> declaration defines that this document is an HTML5 document • The <html> element is the root element of an HTML page • The <head> element contains meta information about the HTML page • The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab) • The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc. • The <h1> element defines a large heading • The element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:

<tagname> Content goes here... </tagname>

The HTML element is everything from the start tag to the end tag:

<h1>My First Heading</h1>

My first paragraph.

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
	My first paragraph.	
 	none	*none*

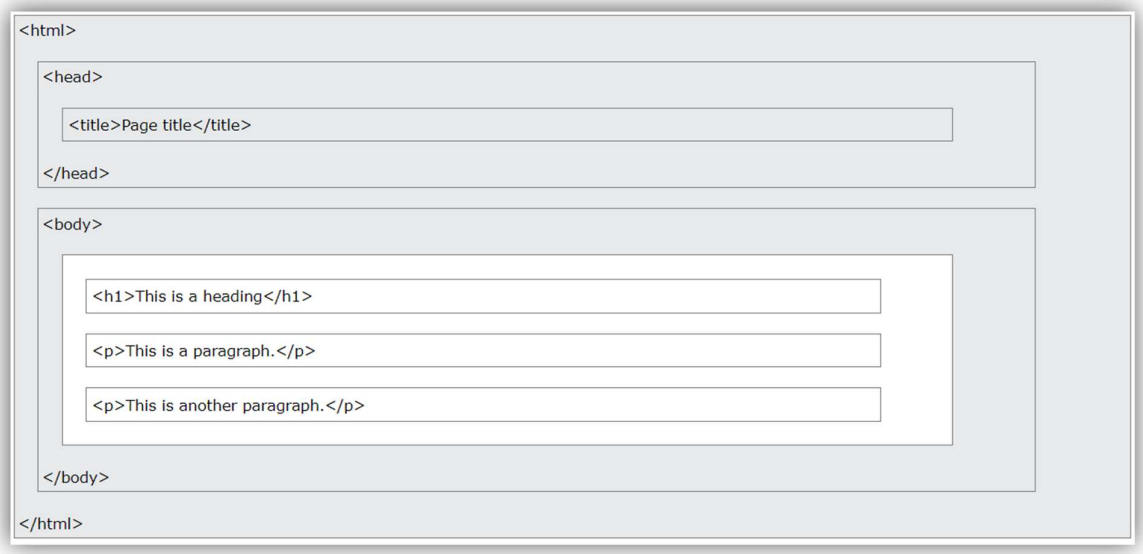
Web Browsers

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display the HTML tags, but uses them to determine how to display the document:

HTML Page Structure

Below is a visualization of an HTML page structure:

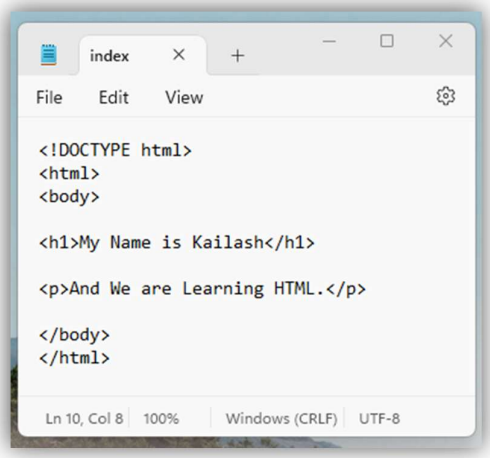


Steps to Write Code and View on Web Browser

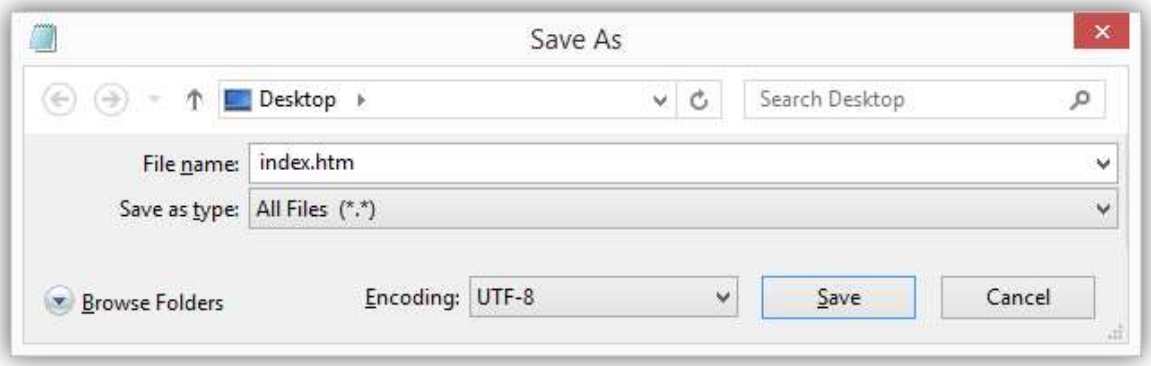
Step 1: Open Notepad (PC)

Step 2: Write Some HTML

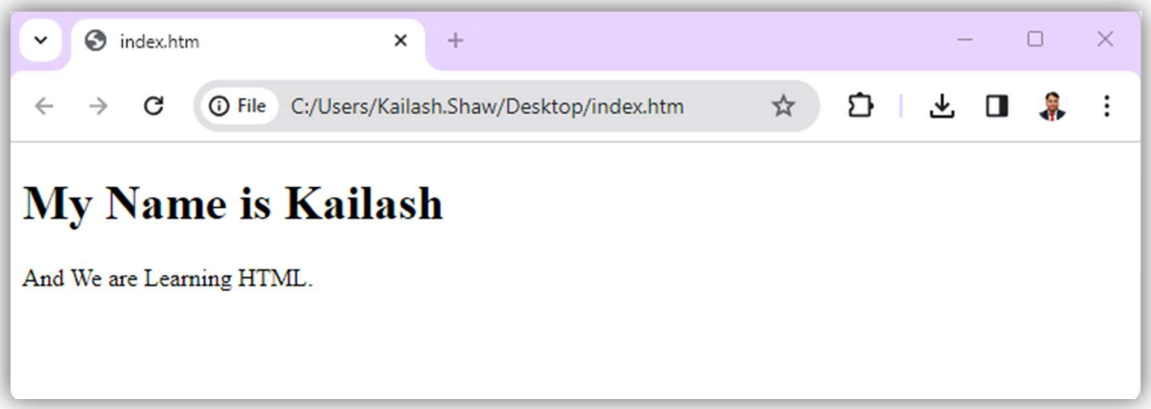
```
<!DOCTYPE html>
<html>
<body>

<h1>My Name is Kailash</h1>

<p>And We are Learning HTML.</p>

</body>
</html>
```



Step 3: Save the HTML Page as Index.htm



Step 4: View the HTML Page in Your Browser



HTML FORMS

Text Fields: The <input type="text"> defines a single-line input field for text input.

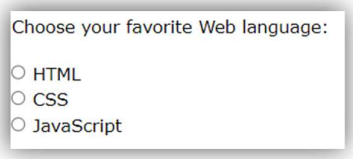
```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

First name:

Last name:

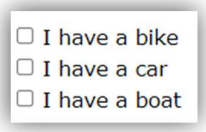
Radio Buttons: The <input type="radio"> defines a radio button. Radio buttons let a user select ONE of a limited number of choices.

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```



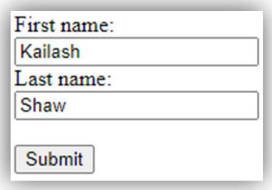
Checkboxes: The <input type="checkbox"> defines a checkbox. Checkboxes let a user select ZERO or MORE options of a limited number of choices.

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```



The Submit Button: The <input type="submit"> defines a button for submitting the form data to a form-handler. The form-handler is typically a file on the server with a script for processing input data. The form-handler is specified in the form's action attribute.

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```



Experiment 1

Problem Statement

Create a basic yet visually appealing webpage for a user registration form using only HTML. The page should include essential elements such as input fields for name, email, and password. Implement a clean and straightforward design that is responsive to different screen sizes. Ensure proper labeling and organization of form elements, demonstrating a fundamental understanding of HTML structure and layout principles. The challenge lies in creating an aesthetically pleasing and user-friendly registration page without using external CSS or JavaScript.

Example:

Output:

STUDENT REGISTRATION FORM

FIRST NAME (max 30 characters a-z and A-Z)

Middle NAME (max 30 characters a-z and A-Z)

LAST NAME (max 30 characters a-z and A-Z)

DATE OF BIRTH Day: Month: Year:

EMAIL ID

MOBILE NUMBER (10 digit number)

GENDER Male Female

ADDRESS

CITY (max 30 characters a-z and A-Z)

PIN CODE (6 digit number)

STATE (max 30 characters a-z and A-Z)

COUNTRY India

HOBBIES Coding Blog-Writing Hacking Cricket Others

QUALIFICATION

Sl.No.	Examination	Board	Percentage	Year of Passing
1	Class X			
2	Class XII			
3	Graduation			
4	Masters			
		(10 char max)	(upto 2 decimal)	

COURSES APPLIED FOR B.Tech M.Tech PhD MS

Submit Reset

Objective of Experiment:

Explain the Tag Used in Code:

Source Code, with description and with Output (Filled Your details) Need to be Uploaded to the Google Classroom.

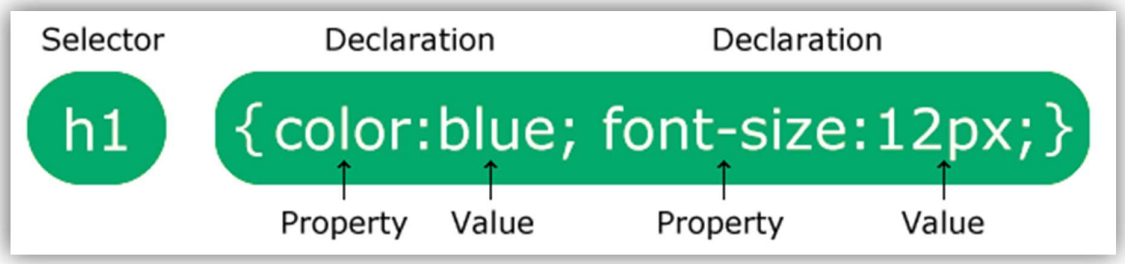
Experiment 2

What is CSS?

CSS is the language we use to style an HTML document. CSS describes how HTML elements should be displayed.

CSS Syntax

A CSS rule consists of a selector and a declaration block.



- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

Example (Without Using CSS)

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<p>Hello World!</p>
<p>These paragraphs are styled with
CSS.</p>

</body>
</html>
```

Hello World!

These paragraphs are styled with CSS.

Example (Using CSS)

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  color: red;
  text-align: center;
}
</style>
</head>
<body>
```

Hello World!

These paragraphs are styled with CSS.

- p is a selector in CSS (it points to the HTML element you want to style: <p>).

Hello World! These paragraphs are styled with CSS. </body> </html>	• color is a property, and red is the property value • text-align is a property, and center is the property value

The CSS ID Selector

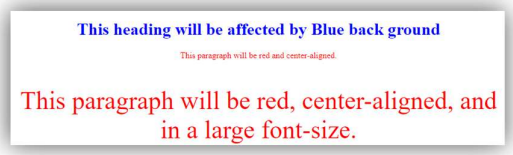
- **The id selector uses the id attribute of an HTML element to select a specific element.**
- **The id of an element is unique within a page, so the id selector is used to select one unique element!**
- **To select an element with a specific id, write a hash (#) character, followed by the id of the element.**

```
<!DOCTYPE html>
<html>
<head>
        <style>
                #para1 {
                        text-align: center;
                        color: red;
                       }
        </style>
</head>
<body>
        <p id="para1">Hello World!</p>
        <p>This paragraph is not affected by the style.</p>
</body>
</html>
```

1. design a responsive web page registration form using HTML and CSS design a responsive web page registration form using HTML and CSS

The CSS class Selector

- **The class selector selects HTML elements with a specific class attribute.**
- **To select elements with a specific class, write a period (.) character, followed by the class name.**

<!DOCTYPE html>
<html>
<head>
<style>
.center{
text-align: center;
color: blue;
}
p.center {
text-align: center;
color: red;
}
	

```
p.large {
  font-size: 300%;
}
</style>
</head>
<body>

<h1 class="center">This heading will be affected by Blue back ground</h1>

<p class="center">This paragraph will be red and center-aligned.</p>

<p class="center large">This paragraph will be red, center-aligned, and in a large font-size.</p>

</body>
</html>
```

The CSS Grouping Selector

- **The grouping selector selects all the HTML elements with the same style definitions.**
- **Look at the following CSS code (the h1, h2, and p elements have the same style definitions):**

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

Better to Group Them

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Certainly! Here are some basic CSS notes to help you get started:

1. Selectors:

- **Element Selector: Targets HTML elements by their type (e.g., p for paragraphs).**
- **Class Selector: Targets elements with a specific class (e.g., .classname).**
- **ID Selector: Targets a specific element with a unique ID (e.g., #idname).**
- **Attribute Selector: Targets elements based on their attribute values (e.g., input[type="text"]).**

2. CSS Properties:

- **Color: color, background-color.**

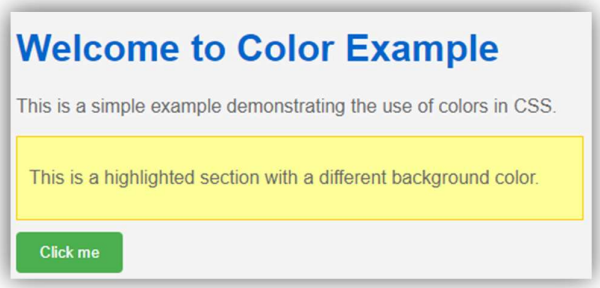
Example:

background-color: #f4f4f4

color: #333; /* Text color */

Color Chart

Color Name	Hex Code
Red	#FF0000
Green	#00FF00
Blue	#0000FF
Yellow	#FFFF00
Orange	#FFA500
Purple	#800080
Pink	#FFC0CB
Brown	#A52A2A
White	#FFFFFF
Black	#000000
Gray	#808080
Light Gray	#D3D3D3
Dark Gray	#A9A9A9




```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
   <title>Color Example</title>
   <style>
      body {
         background-color: #f4f4f4; /* Background color for the entire page */
         color: #333; /* Text color */
         font-family: Arial, sans-serif;
      }

      h1 {
         color: #0066cc; /* Heading color */
      }

      p {
```

```
        color: #666; /* Paragraph text color */
    }

    .highlight {
        background-color: #ffff99; /* Background color for highlighted section */
        border: 1px solid #ffcc00; /* Border color for highlighted section */
        padding: 10px;
        margin: 10px 0;
    }

    .button {
        background-color: #4caf50; /* Button background color */
        color: #fff; /* Button text color */
        padding: 10px 20px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
    }

    .button:hover {
        background-color: #45a049; /* Button background color on hover */
    }
  </style>
</head>
<body>
  <h1>Welcome to Color Example</h1>

  <p>This is a simple example demonstrating the use of colors in CSS.</p>

  <div class="highlight">
    <p>This is a highlighted section with a different background color.</p>
  </div>

  <button class="button">Click me</button>
</body>
</html>
```

- **Text Styling: font-family, font-size, font-weight, text-align, text-decoration.**

Example:

The font-family property in CSS allows you to specify the preferred font for text. You can provide a font family name, or you can list multiple font family names as a fallback in case the user's system doesn't have the first choice. Here are a few examples of commonly used font family values:

Generic Font Families:

- **serif: Times New Roman, Georgia, serif**
- **sans-serif: Arial, Helvetica, sans-serif**
- **monospace: Courier New, Courier, monospace**

Specific Font Families:

- **'Arial', sans-serif: Specifies Arial as the preferred font with a fallback to sans-serif if Arial is not available.**

- **'Helvetica Neue', Helvetica, Arial, sans-serif: Specifies Helvetica Neue as the preferred font with fallback options.**
- **'Times New Roman', Times, serif: Specifies Times New Roman with fallback options.**

font_size: The font-size property in CSS is used to set the size of text. It can take various values, including absolute sizes (like pixels), relative sizes (like percentages or em units), or keywords (like "medium" or "large").

font-weight: The font-weight property in CSS is used to set the thickness or boldness of the text characters. It accepts various values, including numeric values and keywords.

- font-weight: normal; /* Normal font weight (equivalent to 400) */
- font-weight: bold; /* Bold font weight (equivalent to 700) */
- font-weight: 300; /* Lighter than normal font weight */
- font-weight: 800; /* Extra bold font weight */
- font-weight: 200; /* Light font weight */

text-align: The text-align property in CSS is used to set the horizontal alignment of text content. It can be applied to block-level and inline-level elements.

- text-align: center; /* Center-align the heading */
- text-align: justify; /* Justify-align the paragraphs */
- text-align: right; /* Right-align specific elements using a class */

text-decoration: The text-decoration property in CSS is used to set or remove decorations such as underlines, overlines, and line-throughs on text.

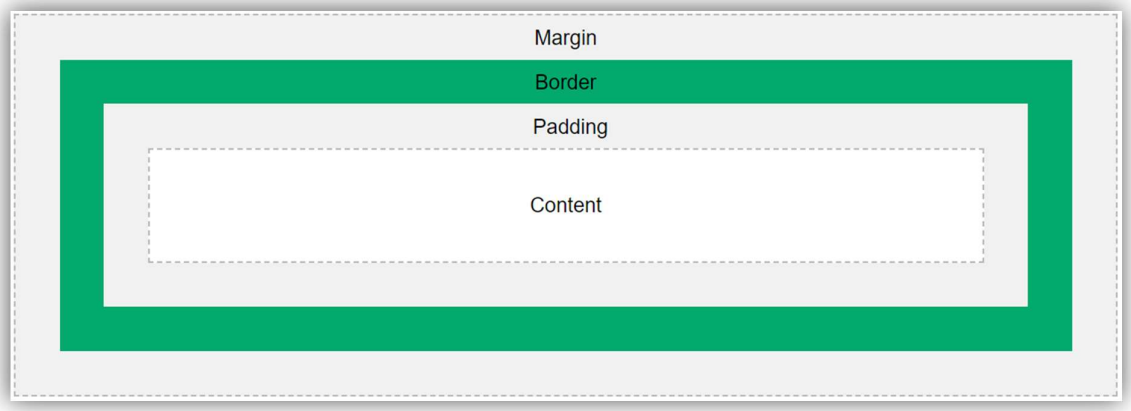
- text-decoration: underline; /* Underline the heading */
- text-decoration: line-through; /* Apply a line-through decoration to paragraphs */
- text-decoration: none; /* Remove text decoration using a class */
- text-decoration: overline; /* Apply an overline decoration using a class */

- **Box Model: width, height, margin, padding, border.**
- **Display: display, visibility, float.**
- **Positioning: position, top, right, bottom, left.**
- **Flexbox: display: flex, flex-direction, justify-content, align-items.**
- **Grid: display: grid, grid-template-columns, grid-template-rows.**
- **Background: background-image, background-size, background-repeat.**
- **Transformations: transform, rotate, scale, translate.**
- **Transition: transition-property, transition-duration, transition-timing-function.**

3. Box Model:

- **Content: The actual content of the box (width and height).**
- **Padding: Clears an area around the content inside the border.**
- **Border: A border surrounding the padding.**
- **Margin: Clears an area outside the border. It's space between borders of adjacent boxes.**

Example: In CSS, the term "box model" is used when talking about design and layout. The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model:

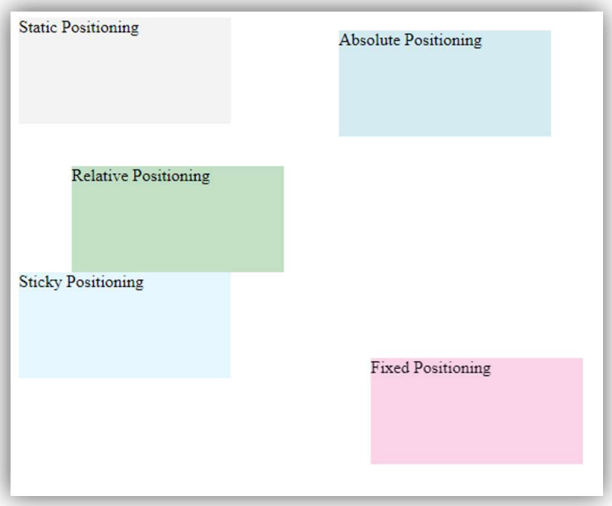



```
.paragraph-box {
        width: 300px;
        height: 150px;
        margin: 20px;
        padding: 15px;
        border: 2px solid #333;
        box-sizing: border-box;
    }
```

```
<p class="paragraph-box">This is a paragraph styled as a box using the Box Model properties.</p>
```

4. Positioning:

- **Static: Default position. Elements are positioned according to the normal flow of the document.**
- **Relative: Positioned relative to its normal position.**
- **Absolute: Positioned relative to the nearest positioned ancestor (or the containing block if no positioned ancestor).**
- **Fixed: Positioned relative to the browser window. It stays fixed even if the page is scrolled.**
- **Sticky: Acts like relative positioning until an element crosses a specified point during scrolling.**



Example:

```
.static-box {
        width: 200px;
        height: 100px;
        background-color: #f4f4f4;
```

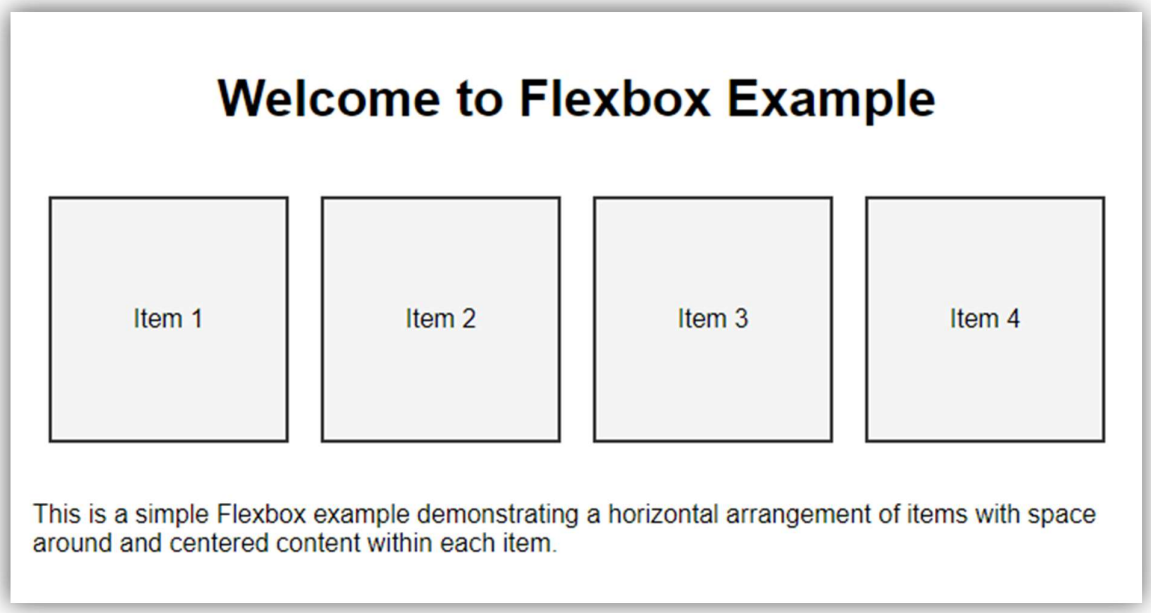
```
        margin-bottom: 20px;
    }
.relative-box {
        width: 200px;
        height: 100px;
        background-color: #c2e0c6;
        position: relative;
        top: 20px;
        left: 50px;
        margin-bottom: 20px;
    }
.absolute-box {
        width: 200px;
        height: 100px;
        background-color: #d4ebf2;
        position: absolute;
        top: 20px;
        right: 50px;
        margin-bottom: 20px;
    }
.fixed-box {
        width: 200px;
        height: 100px;
        background-color: #fbd3e9;
        position: fixed;
        bottom: 20px;
        right: 20px;
        margin-bottom: 20px;
    }
.sticky-box {
        width: 200px;
        height: 100px;
        background-color: #e6f7ff;
```

```
    position: sticky;
    top: 20px;
    margin-bottom: 20px;
}
```

5. Flexbox:

- **A layout model that allows elements to distribute space and align content.**
- **Parent (container) properties: display: flex, flex-direction, justify-content, align-items.**
- **Child (item) properties: flex-grow, flex-shrink, flex-basis.**

Example:

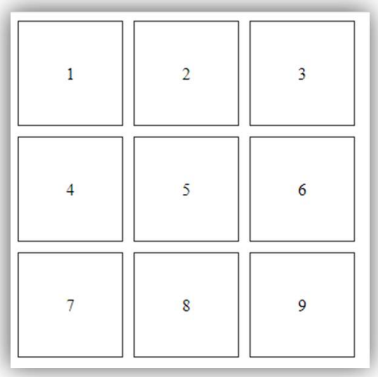



```
body {
    font-family: 'Arial', sans-serif;
    font-size: 16px;
    margin: 0;
    padding: 20px;
    display: flex;
    flex-direction: column; /* Arrange items vertically */
    align-items: center; /* Center items horizontally */
    min-height: 100vh; /* Ensure the body takes at least the full height of the viewport */
}
.flex-container {
    display: flex;
```

```
        justify-content: space-around; /* Distribute items evenly along the main axis */
        align-items: center; /* Center items vertically */
        width: 100%;
        margin: 20px 0;
    }
    .flex-item {
        width: 150px;
        height: 150px;
        background-color: #f4f4f4;
        border: 2px solid #333;
        margin: 0 10px;
        display: flex;
        justify-content: center; /* Center content horizontally within each item */
        align-items: center; /* Center content vertically within each item */
    }
<div class="flex-container">
    <div class="flex-item">Item 1</div>
    <div class="flex-item">Item 2</div>
    <div class="flex-item">Item 3</div>
      <div class="flex-item">Item 3</div>
  </div>
```

6. Grid:

- A two-dimensional layout system for the web.
- Container properties: display: grid, grid-template-columns, grid-template-rows.
- Child properties: grid-column, grid-row, grid-area.



```
<style>
    /* Define the grid container */
    .grid-container {
      display: grid;
      grid-template-columns: repeat(3, 100px); /* Three columns with 100px width each */
      grid-template-rows: repeat(3, 100px);    /* Three rows with 100px height each */
      gap: 10px;  /* Gap between grid items */
    }
    /* Style for grid items */
    .grid-item {
      border: 1px solid #333;
      text-align: center;
      line-height: 100px; /* Center content vertically */
    }
  </style>
<div class="grid-container">
    <!-- Grid items -->
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
    <div class="grid-item">7</div>
    <div class="grid-item">8</div>
    <div class="grid-item">9</div>
  </div>
```

7. Responsive Design:

- Use media queries (@media) to apply different styles based on device characteristics.
- Common breakpoints: max-width, min-width.

Example:


```
<style>
    /* Default styles for all screens */
    .grid-container {
      display: grid;
      grid-template-columns: repeat(3, 100px);
      grid-template-rows: repeat(3, 100px);
      gap: 10px;
    }

    .grid-item {
      border: 1px solid #333;
      text-align: center;
      line-height: 100px;
    }
    /* Media query for screens with a maximum width of 600px */
    @media (max-width: 600px) {
      .grid-container {
        grid-template-columns: repeat(2, 100px);
        grid-template-rows: repeat(4, 100px);
      }
```

```
    }
    /* Media query for screens with a minimum width of 1200px */
    @media (min-width: 1200px) {
      .grid-container {
        grid-template-columns: repeat(4, 100px);
        grid-template-rows: repeat(2, 100px);
      }
      .grid-item {
        font-size: 18px; /* Increase font size for larger screens */
      }
    }
  </style>
```

8. CSS Comments:

- Use /* */ to add comments in your CSS for better code readability.

Remember, these are just the basics, and CSS is a powerful styling language with many more features and properties. As you progress, you may explore advanced topics such as CSS preprocessors (e.g., Sass, Less), CSS frameworks (e.g., Bootstrap), and CSS-in-JS solutions for more complex applications.

How to Add CSS

When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.

There are three ways of inserting a style sheet:

- **External CSS**
- **Internal CSS**
- **Inline CSS**

Problem Statement

Create a fully responsive webpage for a registration form using HTML and CSS. The design should prioritize user experience on various devices and screen sizes. Additionally, implement form validation to ensure accurate user input.

Include fields for the following information:

- **Full Name (First Name, Last Name)**
- **Email Address**
- **Password**
- **Confirm Password**
- **Date of Birth**
- **Gender (Radio buttons or dropdown)**
- **Country (Dropdown menu)**
- **Profile Picture Upload (Include a file input)**

Apply appropriate input types and validation attributes in HTML.

Style form elements for a cohesive and user-friendly appearance.

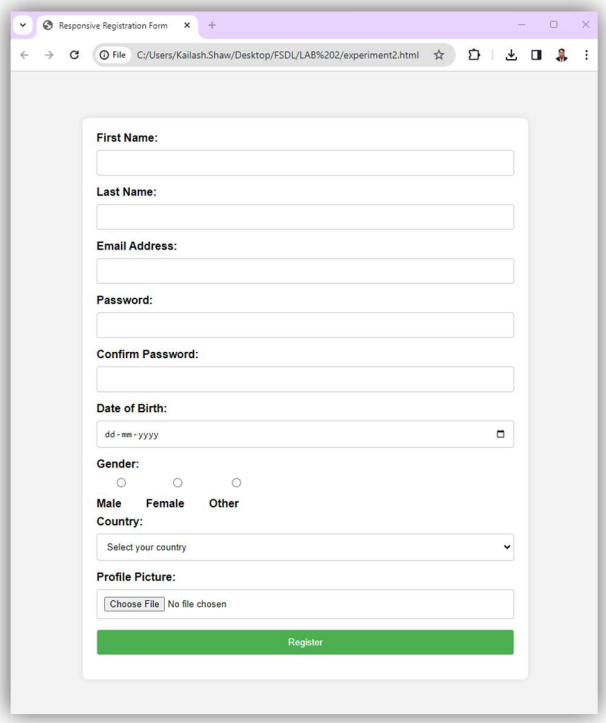
Form Validation:

- **Implement client-side validation using HTML5 validation attributes.**
- **Provide clear and concise error messages for incorrect user input.**

Responsive Design:

- **Ensure the webpage is fully responsive for different devices (mobile, tablet, desktop).**

Output



Submission Guidelines:

Objective of Experiment:

Include a README file explaining your design choices, challenges faced, and any additional features implemented and list of tags used.

Submit a compressed folder containing HTML and CSS files as an assignment in Google Classroom with the form filled with your details.

Include a README file explaining your design choices, challenges faced, and any additional features implemented and list of tags used.

Provide screenshots or a live link to demonstrate the responsiveness of your webpage.

Experiment 3

Bootstrap is an open-source front-end framework developed by Twitter. It is a powerful and popular toolkit for building responsive and mobile-first web applications. Bootstrap provides a set of pre-designed and customizable components, such as navigation bars, modals, forms, buttons, and more, along with a responsive grid system and utility classes.

Key features of Bootstrap include:

1. **Responsive Design: Bootstrap is built with a mobile-first approach, ensuring that web applications look and function well on devices of all sizes, from mobile phones to desktops.**
2. **Grid System: Bootstrap includes a responsive grid system that allows developers to create flexible and fluid layouts. It uses a 12-column grid layout that can be easily customized for various screen sizes.**
3. **Components: Bootstrap provides a variety of pre-designed UI components, such as navigation bars, modals, buttons, forms, and more. These components are styled and can be customized to fit the design of your application.**
4. **JavaScript Plugins: Bootstrap includes a set of JavaScript plugins that enhance the functionality of certain components, such as carousels, tooltips, and modals.**
5. **Customizable: While Bootstrap provides a set of default styles and components, it is highly customizable. Developers can use Sass variables and mixins to tailor Bootstrap to match the visual design of their projects.**
6. **Community and Documentation: Bootstrap has a large and active community of developers. It also comes with extensive documentation, including examples and guidelines, making it easy for developers to get started and build responsive websites quickly.**

To use Bootstrap in a project, you can either download the source files from the official Bootstrap website or include the Bootstrap CSS and JavaScript files from a content delivery network (CDN) in your HTML files.

Getting Started: Include the Bootstrap CSS and JavaScript files in your HTML file.

```
<!-- Bootstrap CSS -->

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">


<!-- Bootstrap JavaScript and dependencies -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Responsive Grid System:

- Bootstrap uses a 12-column grid system for layout.
- Columns can be combined to create various layouts for different screen sizes.

```
<div class="container">
  <div class="row">
    <div class="col-md-6">Column 1</div>
```

```
    <div class="col-md-6">Column 2</div>
  </div>
</div>
```

Components: Bootstrap provides a variety of UI components.

```
<!-- Button -->
<button class="btn btn-primary">Primary Button</button>
<!-- Alert -->
<div class="alert alert-success" role="alert">
  This is a success alert.
</div>


<!-- Navbar -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <!-- ... -->
</nav>
```

Forms: Bootstrap makes it easy to create styled forms.

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone
else.</small>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Utilities: Bootstrap includes utility classes for quick styling.

```
<div class="bg-primary text-light p-3">This is a primary background with light text. </div>
```

Responsive Images: Make images responsive with the img-fluid class.

```
<img src="example.jpg" alt="Example Image" class="img-fluid">
```

Customization

Customize Bootstrap using Sass variables and mixins.

Override default styles to match your project's design.

```
$primary-color: #3498db;

@import 'bootstrap';
```

Problem Statement

Task Description:

You are tasked with implementing a card-flip effect using Bootstrap. Your goal is to create a responsive web page that displays a set of cards, each of which can be flipped to reveal additional content on the back side. Your implementation should utilize Bootstrap for layout and styling, ensuring compatibility across various devices and screen sizes.

Requirements:

Front-end Design: Design a web page layout using Bootstrap that includes multiple cards.

Card Flip Effect: Implement a card-flip effect using Bootstrap's built-in features or custom CSS.

Content Display: Each card should initially display a front side with a brief summary or image. On flipping, the back side should reveal additional content, such as detailed information or related images.

Responsive Design: Ensure that your implementation is responsive, adapting gracefully to different screen sizes and devices.

Optional Enhancements: You may choose to add additional features or styling to enhance the user experience, such as animations or transitions.

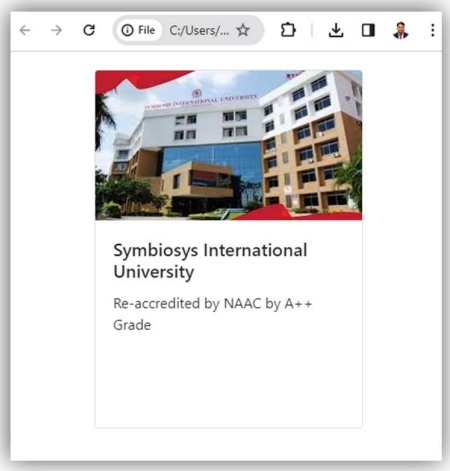
Submission Files:

1. **index.html**
2. **styles.css**
3. **images/ (folder containing necessary images)**

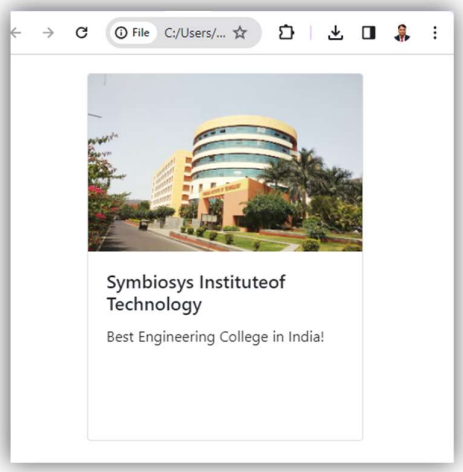
Output:

You must upload the above files in a folder in the Classroom. Choose Your Recent Photo as Front Image. And Choose Your Family Photo as back Image. Give Proper title and Text as per your wish.

In Journal Submission write the Objective and TAG used in the Program.



Front Image



Back Image

Experiment 4

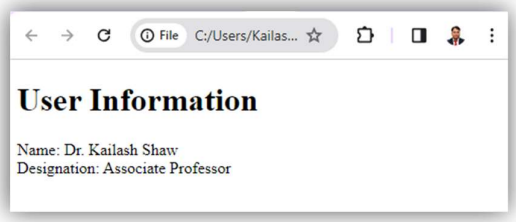
JavaScript Basics:

JavaScript is a high-level, interpreted programming language that is commonly used to add interactivity to web pages. Here's an overview of some fundamental concepts and syntax in JavaScript:

1. **Variables:** Variables are used to store data values. In JavaScript, you can declare variables using var, let, or const.

```
var name = "John";
let age = 30;
const PI = 3.14;
```

Example: Display Your name and Designation on Inner HTML body using Script




```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
content="width=device-width, initial-scale=1.0">
   <title>Display Name and Age</title>
</head>
<body>
   <h1>User Information</h1>
   <p id="userInfo"></p>

   <script>
      // Declare variables
      var name = "Dr. Kailash Shaw";
      var designation = "Associate Professor";

      // Access the HTML element where you want to display the information
      var userInfoElement = document.getElementById("userInfo");

      // Update the content of the HTML element
      userInfoElement.innerHTML = "Name: " + name + "<br>Designation: " + designation;
   </script>
</body>
</html>
```

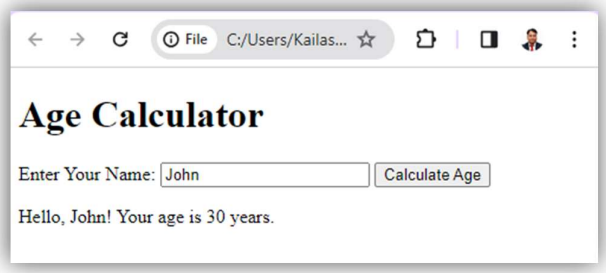
2. **Data Types:** JavaScript supports various data types including strings, numbers, booleans, arrays, objects, and more.

```
var message = "Hello";
var number = 10;
var isTrue = true;
var fruits = ["apple", "banana", "orange"];
var person = { name: "John", age: 30 };
```

Example: Enter Name in Text box, When click Button will show the age of person. Use script?

```
<html>
<head>
   <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Age Calculator</title>
</head>
<body>
    <h1>Age Calculator</h1>
    <label for="nameInput">Enter Your Name:</label>
    <input type="text" id="nameInput">
    <button onclick="calculateAge()">Calculate Age</button>
    <p id="ageDisplay"></p>

    <script>
        // Define an object to store name and
age
        var people = [
            { name: "John", age: 30 },
            { name: "Alice", age: 25 },
            { name: "Bob", age: 35 }
            // Add more people as needed
        ];

        function calculateAge() {
            // Get the value entered in the text box
            var inputName = document.getElementById("nameInput").value;

            // Search for the name in the array of people
            var foundPerson = people.find(function(person) {
                return person.name === inputName;
            });

            // If the person is found, display their age
            if (foundPerson) {
                document.getElementById("ageDisplay").innerHTML = "Hello, " +
foundPerson.name + "! Your age is " + foundPerson.age + " years.";
            } else {
                // If the person is not found, display a message
                document.getElementById("ageDisplay").innerHTML = "Sorry, the age for " +
inputName + " is not available.";
            }
        }
    </script>
</body>
</html>
```



3. **Operators:** JavaScript includes arithmetic, comparison, logical, assignment, and other types of operators.

```
var sum = 5 + 3;
var isEqual = (5 === 5);
var result = (age >= 18) ? "Adult" : "Minor";
```

4. **Functions:** Functions are blocks of code that perform a specific task. They can be declared and invoked as needed.

```
function greet(name) {
    return "Hello, " + name + "!";
}
```

```
var message = greet("Alice");
```

5. **Conditional Statements:** JavaScript supports if, else if, and else statements for decision making.

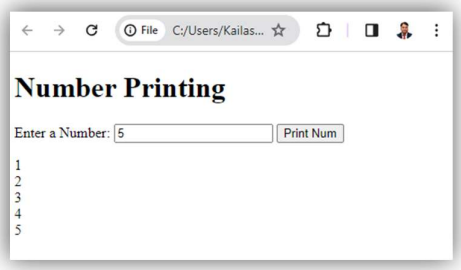
```
if (age >= 18) {
   console.log("You are an adult.");
} else {
   console.log("You are a minor.");
}
```

6. **Loops:** JavaScript provides for, while, and do-while loops for repetitive tasks.

```
for (var i = 0; i < 5; i++) {
   console.log("Iteration " + i);
}
```

Example: Accept input from Text as number and display numbers starting from 1 to entered number.

```
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Print Number</title>
</head>
<body>
   <h1>Number Printing</h1>
   <label for="nameInput">Enter a Number:</label>
   <input type="text" id="numInput">
   <button onclick="printNum()">Print Num</button>
   <p id="numDisplay"></p>
   <script>
      function printNum() {
         // Get the value entered in the text box
         var num = parseInt(document.getElementById("numInput").value);
                  msg="";
         for(var i=1;i<=num;i++){
                           msg= msg + i + "<br>";
                  }
                  document.getElementById("numDisplay").innerHTML=msg;
      }
   </script>
</body>
</html>
```



7. **Arrays:** Arrays are used to store multiple values in a single variable. JavaScript arrays are zero-indexed.

```
var fruits = ["apple", "banana", "orange"];
console.log(fruits[0]); // Output: "apple"
```

8. **Objects:** Objects are collections of key-value pairs. Keys are strings, and values can be of any data type.

```
var person = {
   name: "John",
   age: 30,
   isMale: true
};
console.log(person.name); // Output: "John"
```

9. **Events:** JavaScript allows you to respond to user actions (events) such as clicks, mouse movements, and keyboard inputs.

```
document.getElementById("myButton").addEventListener("click", function() {
   alert("Button clicked!");
});
```

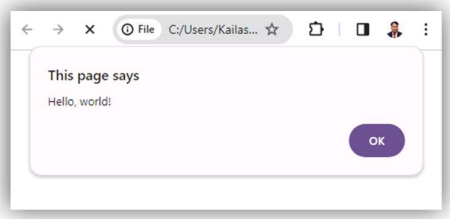
10. **DOM Manipulation:** The Document Object Model (DOM) represents the structure of HTML documents. JavaScript can be used to manipulate the DOM to change content, styles, or attributes of HTML elements.

```
document.getElementById("myElement").innerHTML = "New content";
```

In this experiment, you will explore the usage of JavaScript to display various types of popup boxes. Your task is to create a webpage that demonstrates the functionality of different popup boxes, including alert boxes, alert boxes with line breaks, confirm boxes, and prompt boxes.

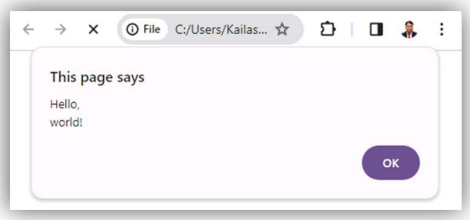
Alert Box: Implement JavaScript code to display a simple alert box with a message.

```
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Alert Box Example</title>
</head>
<body>
   <h1>Alert Box Example</h1>
   <script>
      // Display an alert box with a message
      alert("Hello, world!");
   </script>
</body>
</html>
```

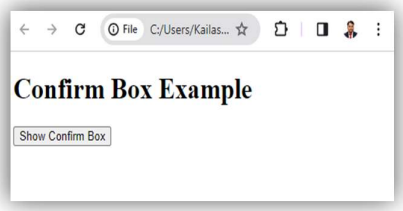


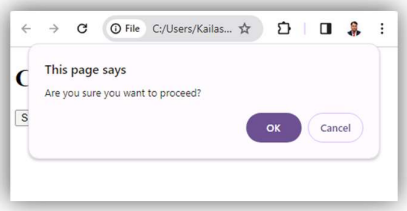
Alert Box with Line Breaks:

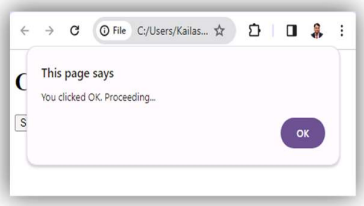
```
<html lang="en">
<head>
      <title>Alert Box with Line Break</title>
</head>
<body>
   <h1>Alert Box with Line Break</h1>

   <script>
      // Display an alert box with a message containing a line break
      alert("Hello,\nworld!");
   </script>
</body>
</html>
```



Confirm Box: Implement JavaScript code to display a confirm box with a message and two buttons (OK and Cancel). Capture the user's response and display an appropriate message based on their choice.

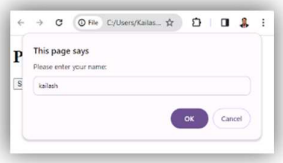







```
<html>
<head>
   <title>Confirm Box Example</title>
</head>
<body>
   <h1>Confirm Box Example</h1>
   <button onclick="showConfirm()">Show Confirm Box</button>
   <script>
      function showConfirm() {
         // Display a confirm box with a message
         var response = confirm("Are you sure you want to proceed?");
         // Check the user's response
         if (response == true) {
            alert("You clicked OK. Proceeding...");
         } else {
            alert("You clicked Cancel. Canceling...");
         }
      }
   </script>
</body>
</html>
```

Prompt Box: Implement JavaScript code to display a prompt box with a message and an input field. Capture the user's input and display it on the webpage.

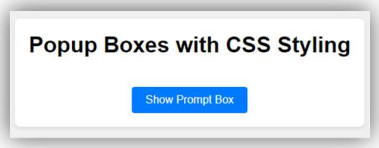


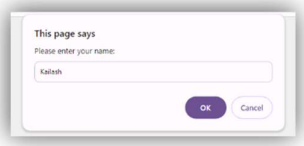

```
<html >
<head>
   <title>Prompt Box Example</title>
</head>
<body>
   <h1>Prompt Box Example</h1>

   <button onclick="showPrompt()">Show Prompt Box</button>
   <p id="userInput"></p>

   <script>
```

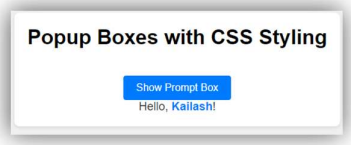
```
    function showPrompt() {
      // Display a prompt box with a message and an input field
      var userInput = prompt("Please enter your name:");

      // Check if the user entered a name
      if (userInput !== null && userInput !== "") {
        // Display the user's input on the webpage
        document.getElementById("userInput").innerText = "Hello, " + userInput + "!";
      } else {
        // Display a message if the user canceled or did not enter a name
        document.getElementById("userInput").innerText = "No name entered.";
      }
    }
  </script>
</body>
</html>
```

Styling and Presentation: Use CSS to enhance the appearance of the popup boxes and ensure readability. Design the webpage layout to provide a clear presentation of the popup boxes and their functionalities.

```
<html>
<head>
  <title>Popup Boxes with CSS Styling</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f0f0f0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }

    .container {
      background-color: #fff;
      border-radius: 8px;
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
      padding: 20px;
      text-align: center;
    }

    h1 {
      margin-top: 0;
    }

    button {
      padding: 10px 20px;
      font-size: 16px;
      margin-top: 20px;
      background-color: #007bff;
      color: #fff;
      border: none;
      border-radius: 4px;
      cursor: pointer;
```







```
            transition: background-color 0.3s ease;
        }
        button:hover {
            background-color: #0056b3;
        }
        p {
            margin: 0;
            font-size: 18px;
            color: #333;
        }
        .userInput {
            font-weight: bold;
            color: #007bff;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Popup Boxes with CSS Styling</h1>
        <button onclick="showPrompt()">Show Prompt Box</button>
        <p id="userInput"></p>
    </div>

    <script>
        function showPrompt() {
            var userInput = prompt("Please enter your name:");
            if (userInput !== null && userInput !== "") {
                document.getElementById("userInput").innerHTML = "Hello, <span class='userInput'>" +
userInput + "</span>!";
            } else {
                document.getElementById("userInput").innerHTML = "No name entered.";
            }
        }
    </script>
</body>
</html>
```

Problem Statement

Design a web-based application that enhances user interactivity by implementing a password-protected activation mechanism for input fields. The application prompts users to enter a password to activate textboxes, subsequently validating the format of mobile numbers and email addresses entered by the user.

- The application should present users with a user-friendly interface containing input fields for mobile number, email address.
- Upon accessing the application, users are prompted to enter a password to activate the input fields. The password serves as a security measure to control access to the form.
- After successfully activating the textboxes by entering the correct password, users can input their mobile number and email address.
- The application must validate the format of the mobile number entered by users, ensuring it consists of 10 digits.
- Similarly, the application must validate the format of the email address entered by users, ensuring it follows the standard email format (e.g., user@example.com).

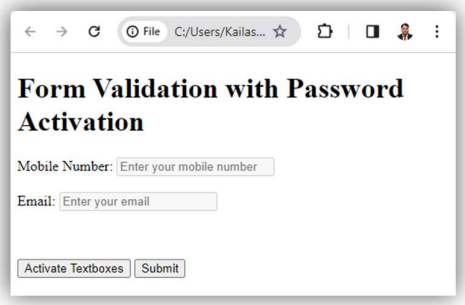
- If the format of the mobile number or email address entered by the user is incorrect, the application should display an alert with guidance on the correct format.
  Use // Regular expression to validate mobile number (10 digits)

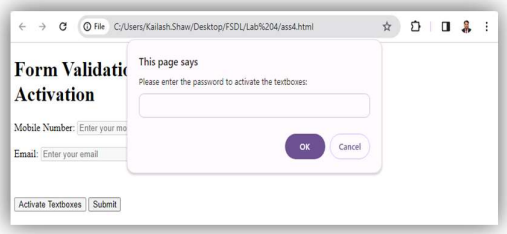
```
var mobileRegex = /^\d{10}$/;
// Check mobile number format
if (!mobileRegex.test(mobileNumber)) {
   alert("Mobile number is not correct.");
   return;
}
// Regular expression to validate email
var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
// Check email format
if (!emailRegex.test(email)) {
   alert("Email is not correct.\nEmail must be in the format: user@example.com");
   return;
}
```

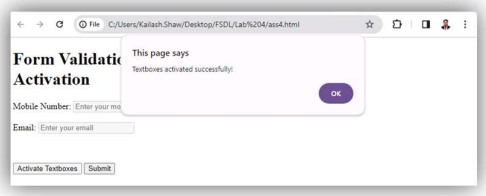
- Finally, when the user clicks the submit button, a confirmation box should appear, allowing users to confirm their submission. Upon confirmation, the application can proceed with further processing or display a success message accordingly.

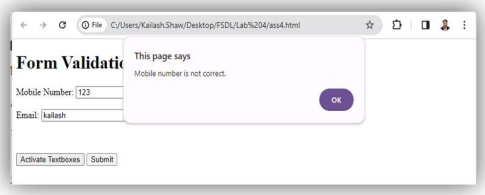
This problem statement aims to develop a user-friendly and interactive web application that enforces security through password protection while ensuring data integrity by validating user input formats.

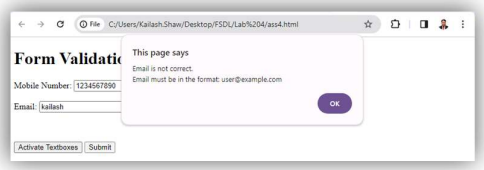
Output

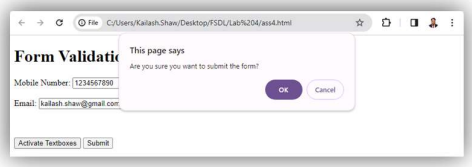


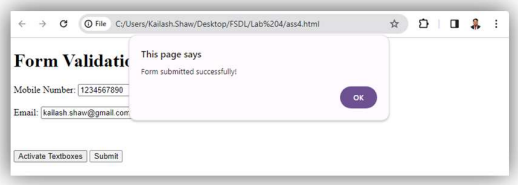












Submission Files:

1. index.html with Java Script

You must upload the above files in the Classroom.

In Journal Submission write the Objective and TAG used in the Program.

Experiment 5

In this experiment You will learn following:

- **JavaScript is used to handle the submit event of the form.**
- **Event listeners are added to the form elements to check for errors upon submission.**
- **If any field is empty, an error message is displayed next to the respective field, and form submission is prevented.**

JavaScript is used to handle the submit event of the form:

Let design a Page having one Text Field Name with Text box and One submit button. When user click submit button, the script must check that the Name field must not be empty. If it is empty, then an error message must be displayed next to the field.

Step 1: first design the form as per the following code


```
<form id="registrationForm" action="#" method="post">
        <h2>Registration Form</h2>
        <label for="firstName">First Name:</label><br>
        <input type="text" id="firstName" name="firstName">
                <span                    id="firstNameError"
class="error"></span><br>
        <input type="submit" value="Register">
</form>
```

Step 2: Now add Script

```
<script>
   const form = document.getElementById('registrationForm');
   const firstName = document.getElementById('firstName');
   const firstNameError = document.getElementById('firstNameError');

   form.addEventListener('submit', function(event) {
      let hasError = false;
      if (firstName.value === '') {
         firstNameError.textContent = 'First name is required';
         hasError = true;
      } else {
         firstNameError.textContent = '';
      }
      if (hasError) {
         event.preventDefault(); // Prevent form submission if there are errors
      }
   });
</script>
```

Here, we're selecting elements from the DOM (Document Object Model) using their respective IDs. These elements include the form itself (registrationForm) and input fields (firstName) along with their corresponding error message elements (firstNameError).

Event Listener for Form Submission:

form.addEventListener('submit', function(event) {…}
This line adds an event listener to the submit event of the form.
The second argument is an anonymous function that will be executed when the form is submitted.

let hasError = false;

This variable hasError is initially set to false. It will be set to true if any validation error is found.

```
if (firstName.value === '') {
      firstNameError.textContent = 'First name is required';
      hasError = true;
   } else {
      firstNameError.textContent = '';
   }
```

This code checks if the firstName input field is empty. If it is, it sets the textContent of the firstNameError span element to display an error message and sets hasError to true.
If the field is not empty, it clears any existing error message.

```
if (hasError) {
      event.preventDefault(); // Prevent form submission if there are errors
   }
```

If hasError is true (indicating at least one field has an error), event.preventDefault() is called to prevent the form from being submitted.

Problem Statement

Design a registration form for a website and implement JavaScript event handling to ensure proper validation of user input. Your form should include the following fields: First Name, Last Name, Email, and Password.

Requirements:

Implement event handling to validate the following conditions:

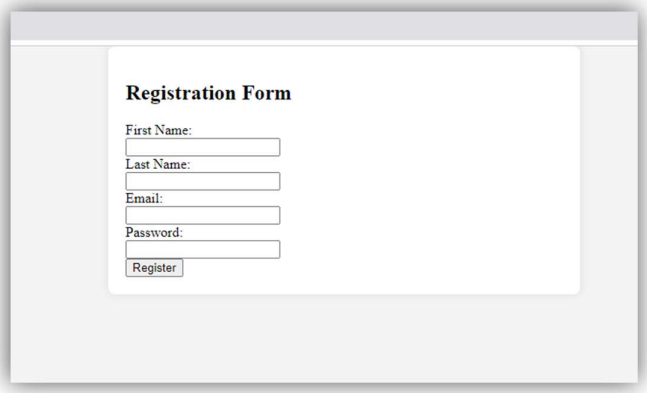
- All fields are required.
- Email field must contain a valid email address format.
- Password field must be at least 8 characters long.
- Display appropriate error messages next to each field if validation fails.
- Prevent the form from being submitted if there are validation errors.
- Provide visual feedback to the user indicating which fields have errors.
- Ensure that the form submission is successful only when all fields are properly filled out.

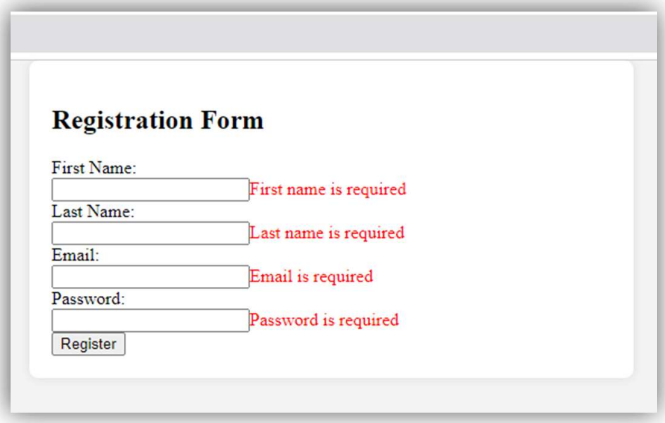
Write a JavaScript function that handles the form submission event and integrates it with the HTML form.

Test your form thoroughly by entering various inputs and verifying that the validation works as expected.

Note: Your implementation should be well-structured, readable, and demonstrate good programming practices. Additionally, consider enhancing the user experience by using CSS to style the form and error messages appropriately.

Output





Submission Guidelines:

Code Submission:

- **Submit your code as a single HTML file containing both the HTML structure and JavaScript code.**
- **Ensure that your code is properly commented to explain the purpose of each section and any complex logic.**
- **Use clear and meaningful variable and function names to enhance readability.**
- **Organize your code into separate sections (e.g., HTML structure, CSS styles, JavaScript logic) to facilitate understanding.**

Testing:

- **Describe the testing process you followed to ensure that your form functions correctly under various scenarios.**
- **Include screenshots or descriptions of test cases you performed to validate the form's behavior, including both successful submissions and error handling.**

Submission Format:

- **Submit your assignment as a single HTML file named "registration_form.html".**
- **Ensure that all external resources (e.g., CSS files, JavaScript libraries) are included within the HTML file or properly referenced from external sources.**

Additional Notes:

- **If you encountered any challenges or limitations during the implementation process, describe them briefly and explain how you addressed or worked around them.**
- **Mention any improvements or optimizations you would make if you had more time to refine your solution.**

Documentation/Journal Writing:

- **Include a brief overview of your registration form at the beginning of your HTML file, explaining its purpose and functionality.**
- **Provide instructions on how to use the form, including any specific input formats or requirements for each field.**
- **Document any additional features or enhancements beyond the basic requirements mentioned in the assignment question.**
- **Write the explanation for the Tags and Code used.**

Experiment 6

ReactJS is a JavaScript library used for building user interfaces, particularly for web applications. It was developed by Facebook and is maintained by a community of developers. React allows developers to create reusable UI components, making it easier to build complex interfaces efficiently.

Software required:

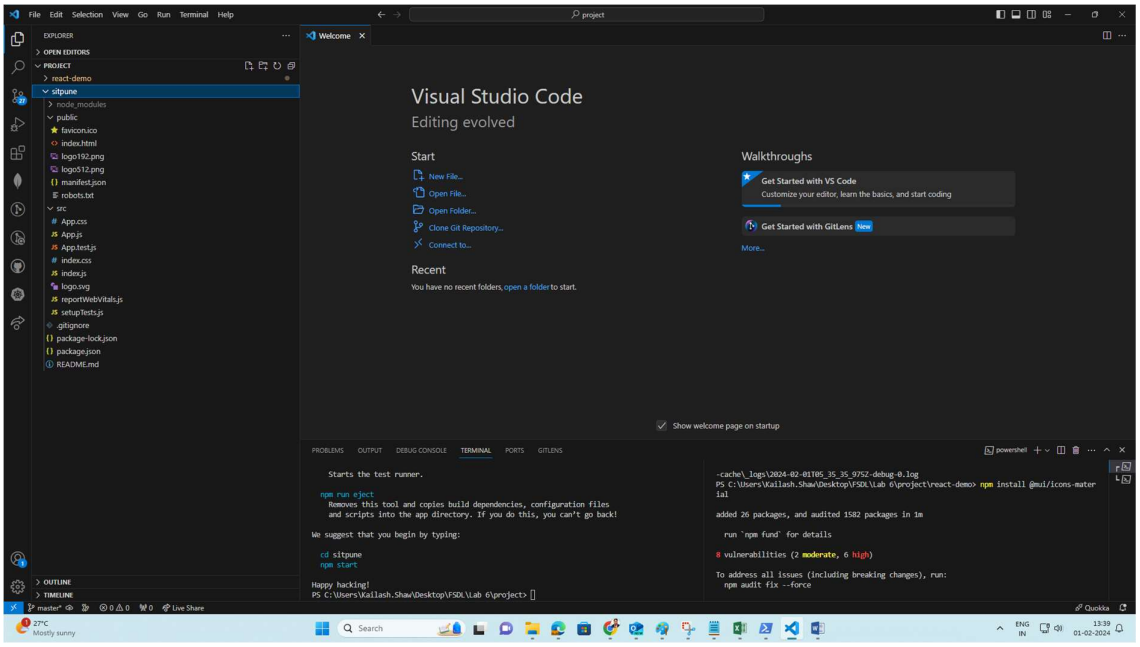
1. Node.Js Download from https://nodejs.org/en
2. Download VS Code

Guide to Open Project in VSCode

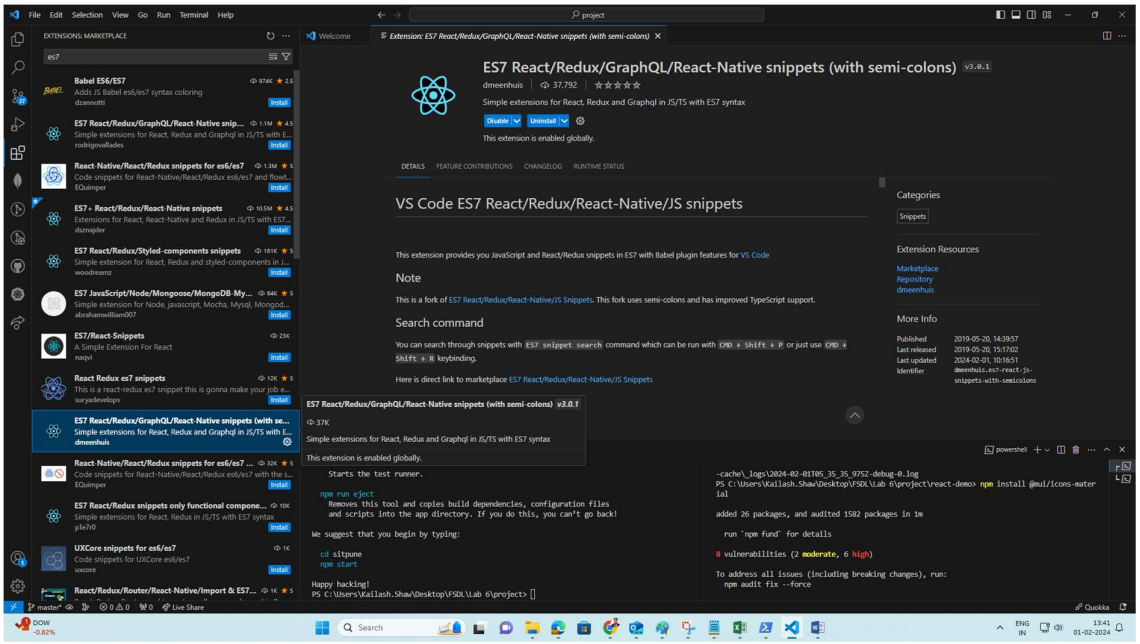
1. Open VSCode
   Open a folder you want to create Project using ReactJs
   Example: my folder is at drive with name "Project" D:\Project

2. From the Terminal in VSCode check NodeJs and NPM is installed by checking its version. Type following command.
   PS D:\project> node –version
   PS D:\project> npm --version
3. Create a React JS Project(let the project name be sitpune) using following Command
   npx create-react-app sitpune --use npm
   if error come then use command
   npm install -g create-react-app
   then use
   npx create-react-app sitpune --use npm
   It will create react-demo project in Project folder with all dependencies.

4. You can move into the react-demo folder by using CD command in terminal



5. Add extension to your project: ES7 React/Redux/GraphQL/React-Native snippets (with semi-colons)



Here are some key concepts and features of ReactJS before starting the assignement:

1. Component-Based Architecture:
   - React applications are built using reusable UI components. Components are self-contained and can be composed together to create complex interfaces.
   - Components can be either functional components (using functions) or class components (using ES6 classes). With the introduction of hooks, functional components are now the preferred way of writing components.

   Example

```
// Functional Component
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

// Usage
const element = <Welcome name="John" />;
ReactDOM.render(element, document.getElementById('root'));
```

2. Virtual DOM (Document Object Model):
   - React uses a virtual DOM to efficiently update the UI. The virtual DOM is an in-memory representation of the actual DOM elements.
   - When changes are made to the UI, React first updates the virtual DOM and then compares it with the previous version to identify the minimal set of changes needed to update the actual DOM.

   Example

```
const element = <h1>Hello, world!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

3. JSX (JavaScript XML):
   - JSX is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript.

- JSX makes it easier to create React components by providing a familiar syntax for defining the UI structure and interactivity.

Example

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

// JavaScript
const element = React.createElement('h1', null, 'Hello, world!');
```

4. State and Props:
   - State represents the data that a component can maintain and modify over time. It allows components to manage their internal state and re-render when the state changes.
   - Props (short for properties) are read-only data passed from a parent component to its child components. Props are used to configure a component and provide data to be displayed.

Example

```
// Functional Component with State
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

// Usage
ReactDOM.render(<Counter />, document.getElementById('root'));
```

5. Lifecycle Methods (Class Components):
   - Class components in React have lifecycle methods that allow developers to hook into various stages of a component's lifecycle, such as mounting, updating, and unmounting.
   - Common lifecycle methods include componentDidMount, componentDidUpdate, and componentWillUnmount.

Example

```
class MyComponent extends React.Component {
  componentDidMount() {
    // Called after the component is mounted (inserted into the DOM tree)
  }

  componentWillUnmount() {
    // Called before the component is removed from the DOM tree
  }

  render() {
    return <h1>Hello, world!</h1>;
  }
}
```

6. Hooks (Functional Components):
   - Hooks are functions that allow functional components to use state and other React features without writing a class.
   - The most commonly used hooks include useState for managing state, useEffect for handling side effects, and useContext for accessing context in functional components.

Example

```
// useState Hook
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

7. One-way Data Flow:
   - React follows a one-way data flow, where data flows downward from parent components to child components via props.
   - Changes to the data are typically handled by the parent component, and the updated data is passed down to child components as props.

Example

```
// Parent Component
function Parent() {
  const data = 'Hello from parent';
  return <Child data={data} />;
}

// Child Component
function Child(props) {
  return <p>{props.data}</p>;
}
```

8. React Router:
   - React Router is a popular library for handling routing in React applications. It allows developers to define routes and navigate between different views within a single-page application.

Example

```
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/about">About</Link></li>
          </ul>
        </nav>

        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
      </div>
    </Router>
  );
}
```

9. Styling:
   - React does not have built-in styling capabilities, so developers can use CSS, CSS preprocessors like Sass or Less, or CSS-in-JS libraries to style their components.
   - Popular CSS-in-JS libraries include styled-components and Emotion.

Example

```
import styled from 'styled-components';

const Button = styled.button`
  background-color: #4CAF50;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
```

10. State Management:
    - For managing complex application state, React developers often use state management libraries like Redux or the Context API.
    - ReactJS provides a powerful and flexible framework for building modern web applications with a focus on performance, reusability, and maintainability. By mastering the basics of React, developers can create interactive and responsive user interfaces for a wide range of applications.

Example

```
import { createStore } from 'redux';

// Reducer
function counterReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

// Store
const store = createStore(counterReducer);

// Dispatch actions
store.dispatch({ type: 'INCREMENT' });
store.dispatch({ type: 'DECREMENT' });
```

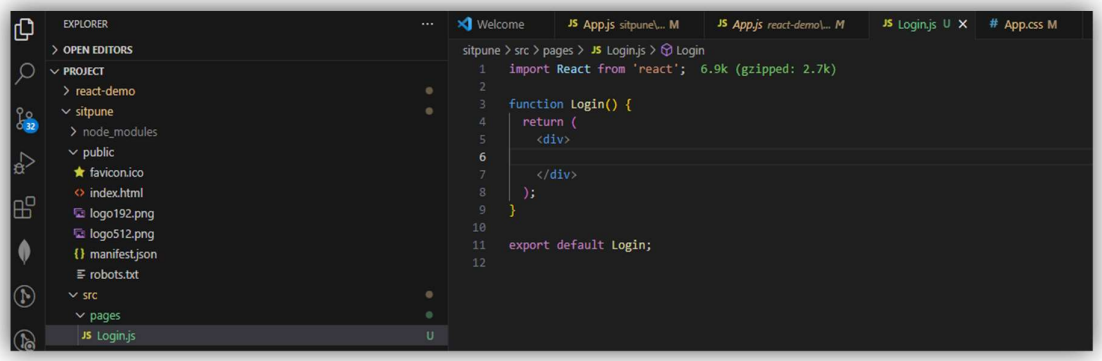
Example: Code to display sign in form

Step 1: Install router through terminal by entering following code.

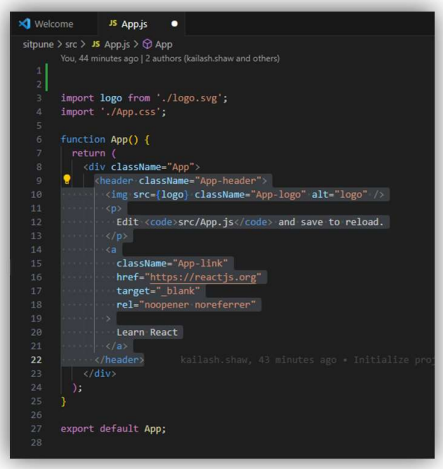
```
npm install react-router-dom
```

Step 2: Create a Folder name pages under SRC

- **Right click on SRC and create folder**
- **Inside the pages folder create Login.js file**
- **Type rfce and press enter in the Login.js file**



Step 3: Open App.js and delete the highlighted area.



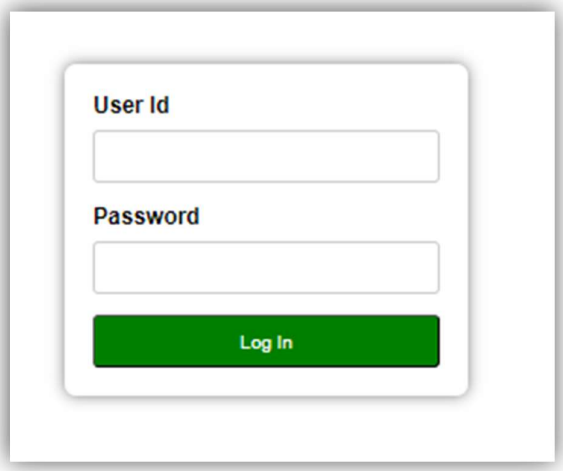
Step 4: replace the below code

```
import './App.css';
import {BrowserRouter as Router, Route, Routes}
from "react-router-dom"
import Login from "./pages/Login"
function App() {
  return (
    <div className="App">
      <Router>
        <Routes>
          <Route path="/"  exct Component={Login}/>
        </Routes>
      </Router>
    </div>
  );
}
export default App;
```

Step 5: Now design Login.js file as per requirement.

```
import React from 'react';
import "../style/Login.css"
function Login() {
  return (
    <div className='body'>

      <form className='form'>
        <label className='label'>User Id</label>
        <input type="text" name="uid" className='input'></input>
        <label className='label'>Password</label>
        <input type="password" name="uid" className='input'></input>
        <input type="submit" value="Log In" className='button'></input>
      </form>
    </div>
  );
}
export default Login;
```

Use appropriate CSS as taught in Lab 2.
Run the code through terminal by typing following command
Start npm
A browser will open localhost:300, and you can view following form



Problem Statement

Use React components to build interactive interfaces.

In this assignment, you will demonstrate your proficiency in using React components to build interactive user interfaces. You will create a web application that incorporates various React components to deliver a dynamic and engaging user experience.

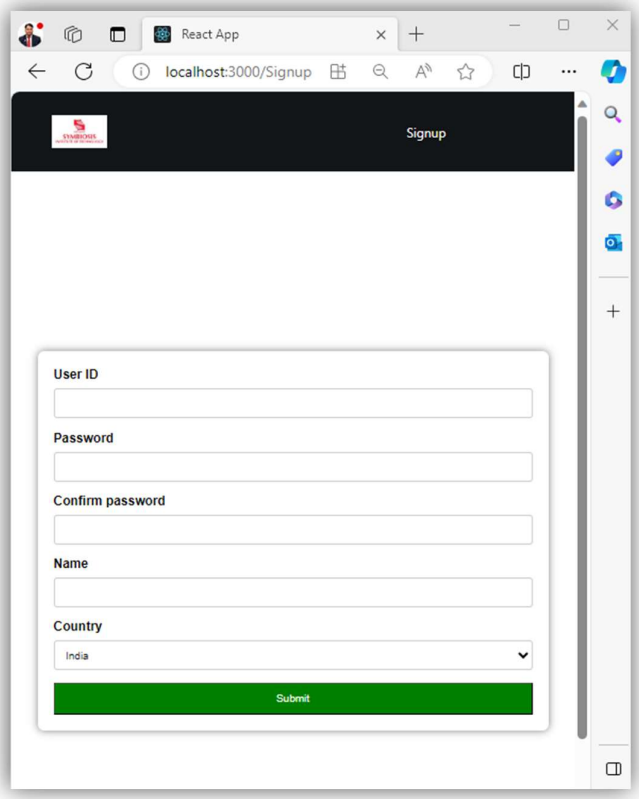
- Set up a new React project using Create React App or any preferred method.
- Ensure that your project structure is organized and follows best practices for React development.
- Design and create multiple React components to represent different parts of your application's user interface.
- Implement reusable components to enhance code modularity and maintainability.
- Apply CSS or a CSS framework (e.g., Bootstrap, Material-UI) to style your components and enhance the visual presentation of your application.
- Aim for a clean and aesthetically pleasing design while prioritizing usability and accessibility.
- Implement interactive features such as forms, buttons, navigation menus, and any other elements that enhance user engagement.

Submission Guidelines:

Project Submission:

- Submit your React project as a compressed folder containing all necessary files.
- Ensure that your project includes all source code, configuration files, and any additional assets used.

Output:



Experiment 7

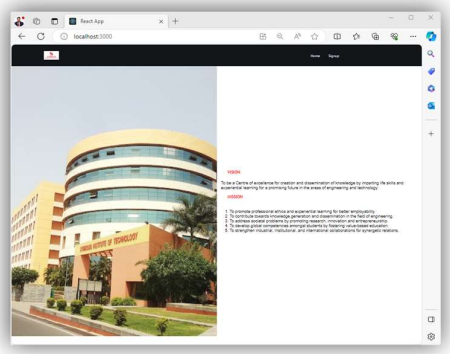
Problem Statement

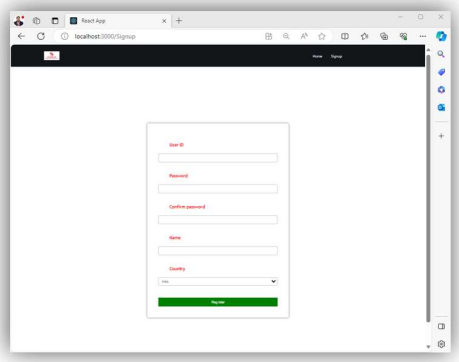
Implement routing in React JS.

With extension of experiment 6, you are supposed to add Navigation Bar menu item Home. Which will be the first page when user access the page. User can select menu for sign up or visit home.

- Add content to your page as per your wish.
- Content in home page must contains a picture in left side, and Text in right side.
- On signup it should exactly look same as experiment 6.

Output:





Project Submission:

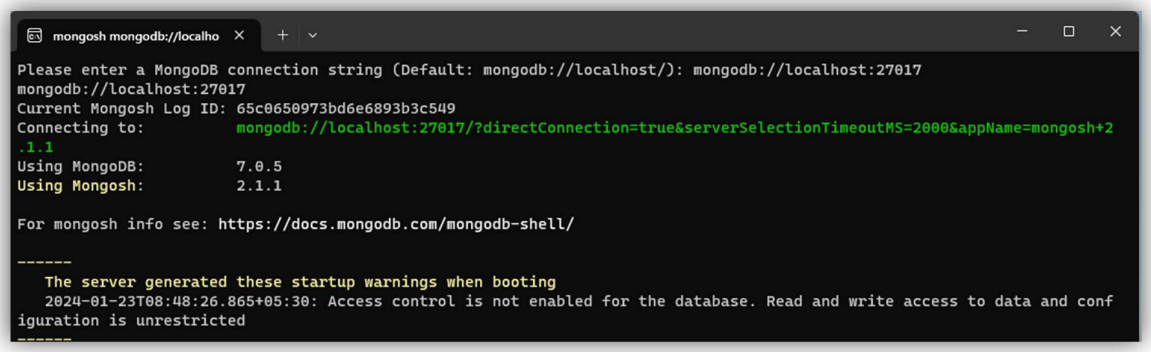
- **Submit your React project code as a compressed folder containing all necessary files.**
- **Ensure that your project includes all source code, configuration files, and any additional assets used.**
- **You may also add one-page report about what are the difficulty you faced and the how you tackle it.**

Journal write-up must include the objective and concept of Routing in ReactJs with example.

Experiment 8

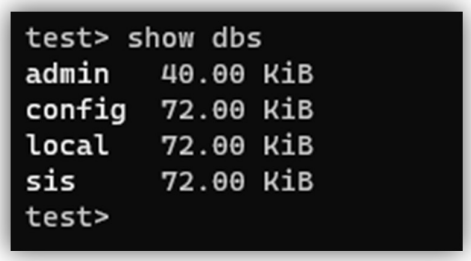
To create a database from a JSON file and execute NoSQL queries, you would typically use a NoSQL database system like MongoDB. Below are the general steps to achieve this:

- Install mongodb from mongodb.com
  To install MongoDB on Windows, first download the latest release of MongoDB from https://www.mongodb.org/downloads. Make sure you get correct version of MongoDB depending upon your Windows version.
- Install Mongdb shell
- Set environmental variable

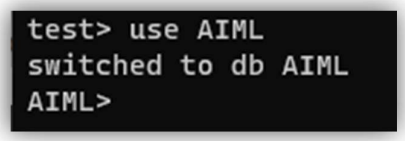


Commands:

1. Show All Databases
   Use below command to get list of all databases.
   show dbs



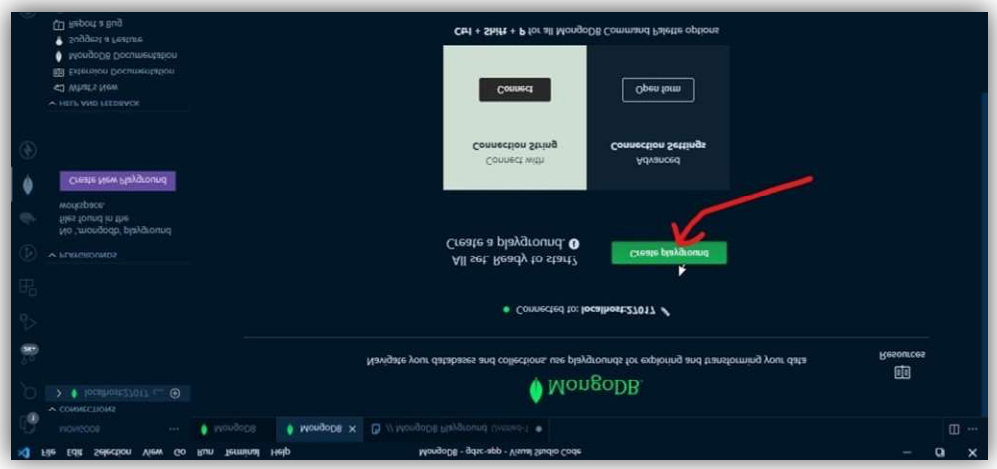
2. To Create a new database, run the following command on the cmd:
   use myDatabase



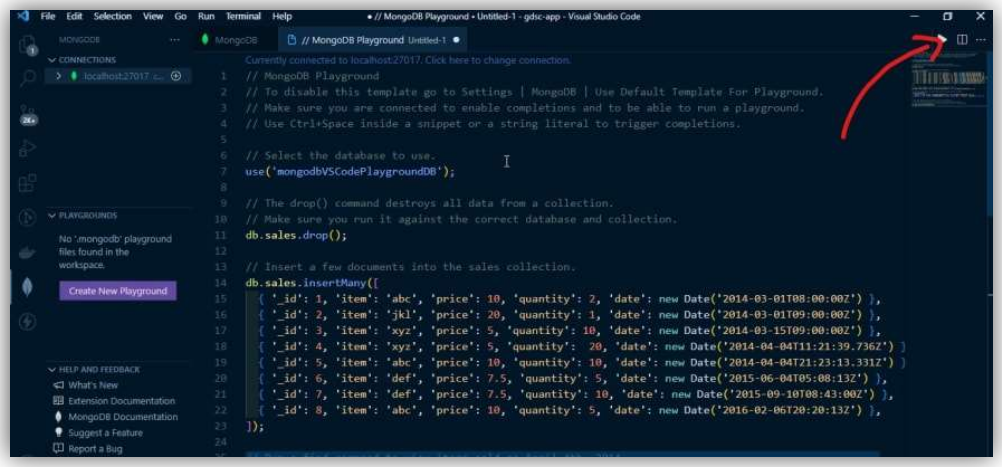
3. Great. Now, let's connect

We're going to use that on VS Code to connect to our newly created database. The connection string for our database will be: mongodb://localhost:27017/AIML, paste this in the text bar at the top of the window, that is, step 3 in the diagram below:

Your newly opened editor tab should look like below:



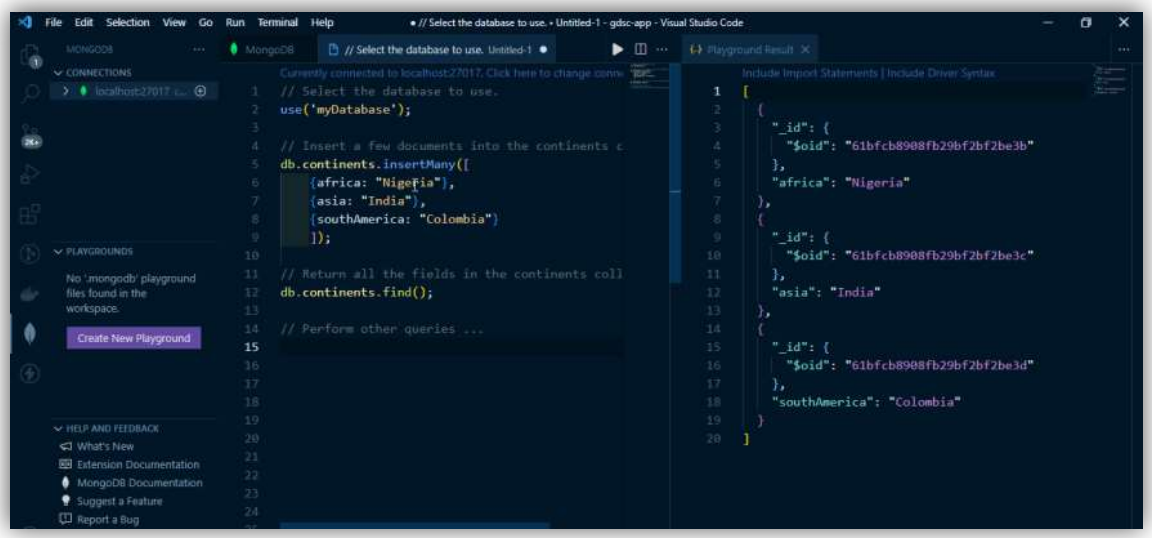
Your newly opened editor tab should look like below:



Delete the content in the default template and paste the following to test our myDatabase database:

```
// Select the database to use
use('myDatabase');

// Insert a few documents in the continents collection
db.continents.insertMany([
    { africa: "Nigeria" },
    { asia: "India" },
    { southAmerica: "Colombia" }
    ]);

// Return all the fields in the continents collection
db.continents.find();

// Write other queries and operations ...
```

Click on the play button at the top-right side to run the code. A new panel should be opened with our results like so:



Congratulations, you made it!

There you have it. Now you can work with your MongoDB databases locally using VS Code, perform database operations and see the results on the fly! Isn't that awesome? Yeah, it is.

Try the following command

```
// Select the database to use.
use('AIML');

// Create Table students
db.createCollection("students")

// insert one students
```

```
db.students.insertOne({Name:"ABC", age:22, GPA:8.2})

// search the students
db.students.find({Name:"ABC"})

// Insert many Studentd
db.students.insertMany([
  {Name:"A1", Age:21, GPA:5.5},
  {Name:"A2", Age:20, GPA:4.5},
  {Name:"A3", Age:22, GPA:8.5},
  {Name:"A4", Age:23, GPA:6.9}]
)

// Search Student Name “A3” information
db.students.find({Name:"A3"})

// sorting display where 1 is ASC and -1 is DSC
db.students.find().sort({name:1})
db.students.find().sort({name:-1})

// Limit the document
db.students.find().limit(3)

// limit + Sort
 db.students.find().sort({GPA:-1}).limit(3)

// Merge Query
db.students.find({GPA:8.2, age:22})

// Get Student Name Only
db.students.find({},{Name:true})

// Get Student Name  and GPA
db.students.find({},{Name:true, GPA:true})

// Do not want ID to display
db.students.find({},{_id:false, Name:true, GPA:true})

// Update syntax   .update(filter, update)
db.students.updateOne({Name:"ABC"},{$set:{age:25}})

// Update Many : lets one more field in collection
db.students.updateMany({},{$set:{fullTime:true}})

// delete One
db.students.deleteOne({Name:"A4"})

//Operator
 // $ Note Equal can be written as %ne
 // $lt is for lesss then
```

```
// $lte is less then equal to
// $gt and $gte can also be used
db.students.find({GPA:{$gte:6,$lte:9}})

// Logical Expression
//$and $or $not
db.students.find({$and:[{age:{$gte:21}},{GPA:{$lte:8}}]})

// Indexed
db.students.find({Name:"A3"}).explain("executionStats")
db.students.createIndex({Name:1})
db.students.getIndexes()

//drop index
db.students.dropIndex("Name_1")
```

Problem Statement

Create a database from .json file and execute NO SQL Queries.

1. Create Database: Create a MongoDB database named AIML.
2. Create Collection: Create a collection named employee within the AIML database.
3. Import Employee Records: Import employee records from a JSON file into the employee collection. Save the following data in a json file and use it for import in Mongodb.

```
[
  {
    "Id": 1,
    "Name": "Charlie Moore",
    "Age": 44,
    "Gender": "Male",
    "Project_id": 4,
    "Hrs_worked": 12
  },
  {
    "Id": 2,
    "Name": "Frank Smith",
    "Age": 36,
    "Gender": "Female",
    "Project_id": 1,
    "Hrs_worked": 12
  },
  {
    "Id": 3,
    "Name": "Hannah Brown",
    "Age": 33,
    "Gender": "Female",
    "Project_id": 2,
    "Hrs_worked": 36
  },
  {
    "Id": 4,
    "Name": "Hannah Davis",
```

```
    "Age": 31,
    "Gender": "Female",
    "Project_id": 2,
    "Hrs_worked": 22
  },
  {
    "Id": 5,
    "Name": "Bob Davis",
    "Age": 41,
    "Gender": "Female",
    "Project_id": 4,
    "Hrs_worked": 16
  },
  {
    "Id": 6,
    "Name": "Hannah Davis",
    "Age": 57,
    "Gender": "Male",
    "Project_id": 2,
    "Hrs_worked": 28
  },
  {
    "Id": 7,
    "Name": "Frank Brown",
    "Age": 49,
    "Gender": "Male",
    "Project_id": 4,
    "Hrs_worked": 30
  },
  {
    "Id": 8,
    "Name": "David Williams",
    "Age": 60,
    "Gender": "Male",
    "Project_id": 2,
    "Hrs_worked": 34
  },
  {
    "Id": 9,
    "Name": "Charlie Johnson",
    "Age": 49,
    "Gender": "Male",
    "Project_id": 3,
    "Hrs_worked": 32
  },
  {
    "Id": 10,
    "Name": "Charlie Johnson",
    "Age": 60,
    "Gender": "Male",
    "Project_id": 4,
    "Hrs_worked": 36
```

```
},
{
  "Id": 11,
  "Name": "Grace Wilson",
  "Age": 42,
  "Gender": "Female",
  "Project_id": 2,
  "Hrs_worked": 15
},
{
  "Id": 12,
  "Name": "Isaac Smith",
  "Age": 32,
  "Gender": "Male",
  "Project_id": 2,
  "Hrs_worked": 23
},
{
  "Id": 13,
  "Name": "Emma Jones",
  "Age": 52,
  "Gender": "Male",
  "Project_id": 1,
  "Hrs_worked": 33
},
{
  "Id": 14,
  "Name": "Isaac Wilson",
  "Age": 21,
  "Gender": "Female",
  "Project_id": 3,
  "Hrs_worked": 32
},
{
  "Id": 15,
  "Name": "Hannah Davis",
  "Age": 43,
  "Gender": "Male",
  "Project_id": 4,
  "Hrs_worked": 32
},
{
  "Id": 16,
  "Name": "Alice Moore",
  "Age": 29,
  "Gender": "Male",
  "Project_id": 1,
  "Hrs_worked": 39
},
{
  "Id": 17,
  "Name": "David Miller",
```

```
    "Age": 25,
    "Gender": "Female",
    "Project_id": 4,
    "Hrs_worked": 32
  },
  {
    "Id": 18,
    "Name": "Isaac Wilson",
    "Age": 32,
    "Gender": "Female",
    "Project_id": 2,
    "Hrs_worked": 31
  },
  {
    "Id": 19,
    "Name": "Charlie Williams",
    "Age": 59,
    "Gender": "Female",
    "Project_id": 1,
    "Hrs_worked": 31
  },
  {
    "Id": 20,
    "Name": "Frank Miller",
    "Age": 55,
    "Gender": "Female",
    "Project_id": 4,
    "Hrs_worked": 24
  }
]
```


4. insertOne: Inserts a single document into the collection.
   { "Id": 21, "Name": "John Doe", "Project_id": 2, "Hrs_worked": 35 }
5. insertMany: Inserts multiple documents into the collection.
   { "Id": 22, "Name": "Jane Smith", "Project_id": 1, "Hrs_worked": 28 },
   { "Id": 23, "Name": "Alice Johnson", "Project_id": 3, "Hrs_worked": 42 }
6. updateOne: Updates a single document that matches the filter.
   { "Id": 21 }, { $set: { "Hrs_worked": 40 } }
7. updateMany: Updates multiple documents that match the filter.
   { "Hrs_worked": { $gt: 30 } }, { $set: { "Overtime": true } }
8. find an employee by their ID.
9. How would you retrieve all employees who are assigned to a specific project ID?
10. Write a query to find employees who have worked more than 30 hours.
11. Can you demonstrate how to use the $gt operator to find employees who are older than 40?
12. Explain the purpose of sorting in MongoDB queries.
13. Sort the Employee table based on Age in Ascending order and display.
14. Sort the Employee table based on Hrs_worked in Descending order and display.
15. Find Employee whose age is greater then 30 and Has_Worked greater then 20.
16. Find Employee whose Gender is Male or Has_Worked greater then 25.
17. Find Employee whose Project_id is not 3.
18. Write a MongoDB query to find all employees who are between the ages of 25 and 35.
19. How would you retrieve employees who have worked between 20 and 30 hours?
20. Write a query to find employees who are either working on Project 1 or Project 2.

Project Submission:

- **Submit your code as a folder containing all necessary files.**
- **Ensure that your project includes all source code, configuration files, and any additional assets used.**
- **You may also add one-page report about what are the difficulty you faced and the how you tackle it.**

Journal write-up must include the objective and Answer to following

1. Explain the difference between insertOne and insertMany in MongoDB.
2. When would you use insertOne over insertMany, and vice versa?
3. Can you provide an example of using insertOne to insert a single document into a MongoDB collection?
4. How would you use insertMany to insert multiple documents into a MongoDB collection?
5. Describe the purpose of updateOne and updateMany in MongoDB.
6. What is the significance of the filter parameter in updateOne and updateMany?
7. How can you use updateOne to update a specific field in a document?
8. Provide an example of using updateMany to update multiple documents based on a common condition.
9. Explain the purpose of sorting in MongoDB queries.
10. Explain the role of logical expressions and operators in MongoDB queries.
11. Explain the purpose of indexing in MongoDB and how it improves query performance.
12. Write a query to list all indexes on the employee collection.

Experiment 9

Basics of Node.js

What is Node.js?

Node.js is an open-source, cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows you to run JavaScript code on the server-side, enabling the development of scalable and high-performance web applications.

Event-Driven and Non-Blocking I/O Model:

Node.js is based on an event-driven architecture that utilizes non-blocking I/O operations. This means that Node.js can handle a large number of concurrent connections efficiently without getting blocked by I/O operations.
Single-Threaded Event Loop:

Node.js operates on a single-threaded event loop, which means that all I/O operations are handled asynchronously. This architecture allows Node.js to handle multiple requests concurrently without the need for creating new threads for each request.

Package Management with npm:

npm (Node Package Manager) is the default package manager for Node.js. It allows developers to install, manage, and share packages of reusable code. npm provides access to a vast ecosystem of libraries and tools that can be easily integrated into Node.js applications.

Common Use Cases:

Node.js is commonly used for building web servers, APIs (Application Programming Interfaces), real-time applications (such as chat applications and gaming servers), microservices, and server-side rendering of JavaScript-based web applications.

Express.js Framework:

Express.js is a popular web application framework for Node.js. It provides a robust set of features for building web servers and APIs, including routing, middleware support, template engines integration, and more. Express.js simplifies the process of building scalable and maintainable web applications in Node.js.

Database Access:

Node.js can interact with various databases, including relational databases like MySQL, PostgreSQL, and SQLite, as well as NoSQL databases like MongoDB, Redis, and Elasticsearch. Libraries and modules such as mysql, pg, sqlite3, mongoose, and redis facilitate database operations in Node.js applications.

Asynchronous Programming:

Asynchronous programming is fundamental in Node.js development. Callbacks, Promises, and async/await are common patterns used to handle asynchronous operations such as reading files, making HTTP requests, and interacting with databases. Asynchronous programming helps prevent blocking I/O operations and ensures the responsiveness of Node.js applications.

Security Considerations:

Node.js applications should follow security best practices to prevent common vulnerabilities such as injection attacks, cross-site scripting (XSS), cross-site request forgery (CSRF), and insecure deserialization. Libraries like Helmet.js can be used to enhance security by setting various HTTP headers.

Monitoring and Debugging:

Monitoring and debugging tools like console.log, debugger statement, and third-party tools like Node.js Inspector and New Relic help developers troubleshoot and optimize Node.js applications. Monitoring tools can track application performance, memory usage, and other metrics to identify bottlenecks and improve efficiency.

Problem Statement

Write a web application with Node.js to insert a document/record in NO SQL

document based database collection.

Assignment Question:

Task:
Design a login system using Node.js and MongoDB. Your task is to create a web application that allows users to log in with their username and password. Upon successful login, the system should display "Welcome, [username]". If the login credentials are incorrect, the system should display "Incorrect password".

Requirements:

- **Develop the login system using Node.js, Express, and MongoDB.**
- **Implement user authentication with username and password.**
- **Use MongoDB to store user credentials securely.**
- **Display appropriate messages based on the login outcome.**
- **Design the web application interface with HTML, CSS, and Handlebars (HBS) templates.**
- **Ensure proper error handling and validation.**
- **Use Visual Studio Code (VS Code) as the development environment.**

Deliverables:

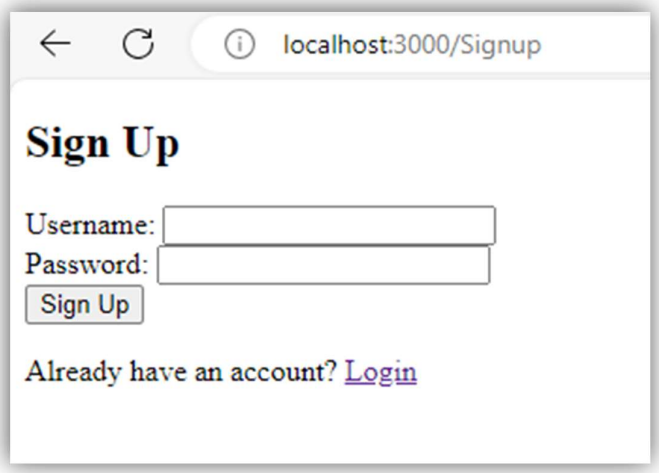
- **Complete source code files.**
- **MongoDB database setup instructions (if required).**
- **Documentation explaining the project structure, implementation details, and any additional features added.**

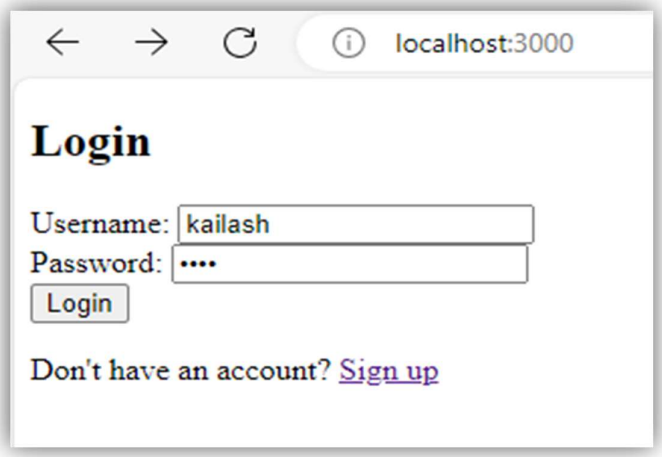
Submission Guidelines:

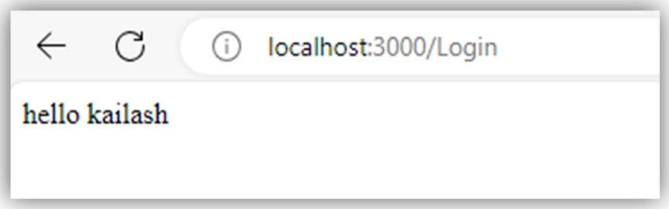
Submit the assignment before the deadline specified by your instructor.
Compress all necessary files into the classroom.

In Journal write the objective and Notes on NodeJs.

Output


localhost:3000/Signup
Sign Up
Username:
Password:
Sign Up
Already have an account? Login


localhost:3000
Login
Username: kailash
Password: ••••
Login
Don't have an account? Sign up


localhost:3000/Login
hello kailash