textOsFtextTOsFliningLFliningTLFtextosfliningIftabulartabproportionalprop
superiorSup superiorSup

superiorSup superiorSup

fontspechyperref

Lab Manual

Web Programming

Programs:
BS Information Technology

Version: 1.0
September 15, 2025

Disclaimer

Copyright

Contact

Riphah International University
13Km, Raiwind Road
Lahore, Pakistan 54000

Email: hasan.sattar@riphah.edu.pk

Changelog

Version	**Date**	**Changed By**	Description
v1.0	2025-08-15	Mr. Hasan Sattar	This is version 1. No change applicable yet

Challenges and Attributes of Labs

Level of Openness (Schwab/Herron)

The table below describes the level of openness a lab question has.

Level	Type	Description	Problem	Methods Procedures	Answers
0	Confirmation Inquiry	Students confirm a principle though an activity when the results are known	Provided by the instructor	Provided by the instructor	Students are given the question, procedure, and the known outcome in advance
1	Structured Inquiry	Students investigate a teacher presented question through a prescribed procedure	Provided by the instructor	Provided by the instructor.	Students develop their own explanations based on the collected evidence
2	Guided Inquiry	Students investigate a teacher presented question using student designed/selected procedures	Provided by the instructor	Students design their own procedure to investigate the problem.	Students interpret their data to propose a viable solution.
3	Open Inquiry	A raw phenomenon is provided to the students who then explore it via student designed/selected procedures	Students choose and define the problem to explore.	Students design and carry out their own investigations.	Students communicate their results and propose solutions.

Table of Contents

1 Lab 01 . . . . **8**
- **1.1** HTML Intro 8
- **1.2** Additional Resources 9
- **1.3** Semantic HTML 9
- **1.4** Metadata Cheat Sheet 9
- **1.5** Responsive Design / Mobile Meta Tags 10
- **1.6** Content Tags 10
- **1.7** Inline Tags 11
- **1.8** Embedded Content and Media Tags 11
- **1.9** Table Tags 11
- **1.10** Input Types 11
- **1.11** Form Elements 12
- **1.12** Lab Task 14

2 Lab 02 . . . . **16**
- **2.1** Layout Design 16
- **2.2** More Layouts 18
- **2.3** Examples 18

3 Lab 03 . . . . **20**
- **3.1** Cascading Style Sheets (CSS) 20
- **3.2** Responsive Design in CSS 20
- **3.3** Summary of Key Units 22
- **3.4** Lab 3 22

4 Lab 04 . . . . **33**
- **4.1** Understanding Flexbox 33
- **4.2** Flexbox Properties 34
- **4.3** Alignment Properties 35
- **4.4** CSS Units of Measurement 38

5 Lab 05 . . . . **40**
- **5.1** Introduction to Bootstrap 40
- **5.2** Bootstrap Screen Size Breakpoints 40
- **5.3** How Breakpoints Work 40
- **5.4** Bootstrap Grid System 42
- **5.5** Bootstrap Components 42
- **5.6** Bootstrap Utilities 43
- **5.7** Bootstrap Forms 44
- **5.8** Media Queries and Responsive Design 44

5.9 quad Step 1: Setting Up Bootstrap 45
5.10 quad Step 2: Bootstrap Grid System (Responsive Layouts) 45
5.11 quad Step 3: Bootstrap Components 46
5.12 quad Step 4: Bootstrap Utilities 48
5.13 quad Step 5: Typography 48
5.14 quad Step 6: Bootstrap Forms 49
5.15 quad Step 7: Bootstrap Tables 50
5.16 quad Final Project 50

6 Lab 06 51

6.1 Introduction to the DOM 51
6.2 Purpose of the DOM in Web Development 51
6.3 Basic Terminology 52
6.4 DOM Tree Structure 52
6.5 DOM Methods for Selecting Elements 53
6.6 Navigating the DOM Tree 55
6.7 Best Practices for Navigating the DOM 57
6.8 Summary of DOM Navigation Properties 57
6.9 Changing Content in the DOM 57
6.10 Adding and Removing Elements 57
6.11 Modifying Attributes and Styles 58
6.12 Summary of DOM Manipulation Techniques 58
6.13 Understanding Event Listeners in the DOM 58
6.14 Common Events 59
6.15 Event Propagation and Bubbling 59
6.16 Best Practices for DOM Traversal 60
6.17 Lab Task 60

7 Lab 07: jQuery 61

7.1 Introduction to jQuery 61
7.2 Basic Selectors and Events 62
7.3 Mini Project: Notes App 63
7.4 Effects and Animations 64
7.5 FAQ with .slideToggle() 65
7.6 Traversing 65
7.7 Task: Create a Menu with Sections 68
7.8 Explanation of the Task Code 68

8 Lab 08: React App Installation, Functional Component, Class, Props, Props Children and Hooks 69

8.1 React App Installation and Components 69
8.2 Lab Task 70

8.3 Final Lab Task 72

9 Lab 09: API Calls 73

9.1 Fetch and Axios 73

9.2 Lab Task 75

10 Lab 10: Frontend + Backend 76

10.1 node + express 76

10.2 Lab Task 77

11 Lab 11: React Props 78

11.1 React.js Essentials 78

11.2 Lab 1: Event Handling 78

11.3 Lab 2: Hooks 79

11.4 Lab 3: State Management 80

11.5 Conclusion 81

12 Lab 12: Node.js Server 82

12.1 Server Programming 82

12.2 Node.js Modules 83

12.3 File Upload 84

13 Lab 13: Express.js 86

13.1 Express.js Basics 86

13.2 Middleware 86

13.3 GET/POST Requests 87

13.4 REST API 87

14 Lab 14: Express.js & MongoDB 89

14.1 Cookies, Sessions, Authentication 89

14.2 Sessions 89

14.3 Authentication 90

14.4 MongoDB & Mongoose 91

15 Lab 15: Express.js PUG & bcrypt 94

15.1 Template Engines 94

15.2 Variables & Conditionals 94

15.3 Iteration 95

15.4 Password Hashing 96

15.5 JWT Middleware 97

Reference List 98

1 Lab 01: HTML

Objectives: Understand the basic structure of an HTML document and semantic tags.

Learning Outcomes: - Identify HTML5 document structure. - Use semantic elements like `<header>`, `<section>`, `<footer>`. - Apply metadata for SEO and accessibility.

Key Competencies: Working with HTML document anatomy, semantic markup, and metadata tags.

1.1 Introduction to HTML

Every HTML document follows a specific structure so browsers can correctly interpret and display content.

```
<!DOCTYPE html>
```

Declares the document type and HTML version to the browser. `<!DOCTYPE html>` indicates HTML5. Helps browsers render pages in standards-compliant mode rather than quirks mode.

```
<html>
```

The root element that wraps all HTML content. Contains two main sections: `<head>` and `<body>`. Can have a `lang` attribute to specify the language of the document (e.g., `<html lang="en">`).

```
<head>
</head>
```

Contains metadata (information about the page, not visible to users). Typical elements inside `<head>` include:

- `<title>`: Defines the page title shown in browser tabs.
- `<meta>`: Provides metadata such as character encoding, author, description, and viewport settings.
- `<link>`: Links to external stylesheets or icons.
- `<script>`: Loads JavaScript files.

```
<body>
</body>
```

Contains all visible content (text, images, videos, links, forms, etc.), structured using HTML tags and semantic elements.

1.2 Additional Resources

These are suggested references for deeper study.

- HTML meta tags - Semantic elements - Simple bare bones HTML webpage - HTML5/CSS bare-bones newsletter template - Open graph social metadata (Twitter, Facebook) - Essential meta tags for social media - The `<meta>` element

1.3 Semantic HTML Cheat Sheet

There are hundreds of semantic tags available to help describe the meaning of your HTML documents. Below is a cheat sheet with some of the most common ones you'll use in this course and in your development career.

1.3.1 Sectioning Tags

Use the following tags to organize your HTML document into structured sections:

- `<header>` – The header of a content section or the web page.
- `<nav>` – The navigation links of a section or the web page.
- `<footer>` – The footer of a section or the web page.
- `<main>` – Specifies the main content of a section or the web page.
- `<aside>` – Secondary content not required for main understanding.
- `<article>` – An independent, self-contained block (e.g., blog post).
- `<section>` – A standalone section of content.

1.4 Metadata Cheat Sheet

1.4.1 HTML `<meta>` Tags

You can leverage meta tags to convey information to search engines. Example:

```
<meta name="author" content="John Doe">
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

1.4.2 Basic Meta Tags (SEO)

```
<meta name="description" content="Short description of the page">
<meta name="title" content="Page Title">
<meta name="author" content="Your Name">
<meta name="language" content="english">
<meta name="robots" content="index,follow">
```

- `description` – summary for search engines. - `robots` – tells search engines how to crawl. - `googlebot` – controls Google-specific behavior.

1.4.3 `http-equiv` Meta Tags

```
<meta http-equiv="refresh" content="30">
<meta http-equiv="Cache-Control" content="no-cache">
```

These control browser behavior (refresh, cache, content type).

1.5 Responsive Design / Mobile Meta Tags

```
<meta name="HandheldFriendly" content="true">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

These ensure mobile friendliness.

—

1.6 Content Tags

- `<blockquote>` – quotation.
- `<dl>` / `<dt>` / `<dd>` – description lists.
- `<figure>` / `<figcaption>` – images and captions.
- `<ol>`, `<ul>`, `<li>` – lists.
- `<p>` – paragraphs.
- `<pre>` – preformatted text.

1.7 Inline Tags

- `<a>` – hyperlinks.
- `<abbr>` – abbreviations.
- `<b>` / `<strong>` – bold text.
- `<em>` / `<i>` – emphasis or italics.
- `<mark>` – highlighted text.
- `<q>` – inline quotation.
- `<sub>` / `<sup>` – subscript/superscript.

1.8 Embedded Content and Media Tags

- `<audio>` – embed audio.
- `<video>` – embed video.
- `<canvas>` – draw graphics.
- `<iframe>` – embed nested pages.
- `<svg>` – scalable vector graphics.
- `<picture>` – responsive images.

1.9 Table Tags

```
<table>
<thead>
<tr><th>Heading</th></tr>
</thead>
<tbody>
<tr><td>Content</td></tr>
</tbody>
</table>
```

- `<thead>`, `<tbody>`, `<tfoot>` structure a table. - `<caption>` – table title. - `<colgroup>` – group of columns.

—

1.10 Input Types

HTML5 introduced many input types for forms:

- `<input type="text">` – single-line text.

- `<input type="password">` – hidden characters.
- `<input type="checkbox">` – multiple selection.
- `<input type="radio">` – single choice.
- `<input type="date">`, `datetime-local`, `month`, `week`.
- `<input type="number">`, `range`.
- `<input type="email">`, `url`, `tel`.
- `<input type="file">`, `hidden`, `image`.
- `<input type="search">`.

1.10.1 Example: Sign-up / Sign-In

```
<form>
<label for="username">Username</label>
<input type="text" id="username" required>

<label for="password">Password</label>
<input type="password" id="password" required>

<button type="submit">Log In</button>
</form>
```

Objectives: By the end of this lab you will: - Become familiar with form tags and their attributes. - Practice using different form elements.

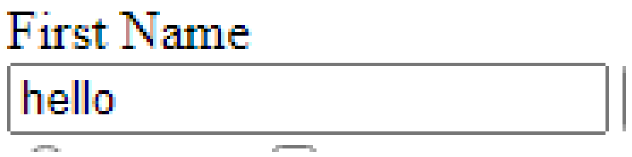
Learning Outcomes: - Create HTML forms with input fields, radio buttons, and checkboxes. - Understand form submission with the `action` and `method` attributes.

Key Competencies: Designing user-friendly and accessible forms.

1.11 Form Elements

Text Field

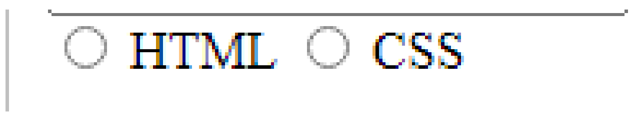
```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```



Radio Buttons

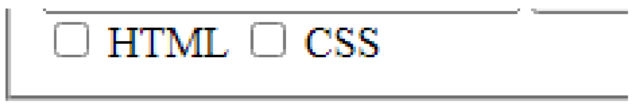
```
<input type="radio" id="html" name="fav_language" value="HTML">
```

```
<label for="html">HTML</label>

<input type="radio" id="css" name="fav_language" value="CSS">
<label for="csss">CSS</label>
```



Checkbox

```
<input type="checkbox" id="html" name="html">
<label for="html">HTML</label>


<input type="checkbox" id="css" name="css">
<label for="css">CSS</label>
```

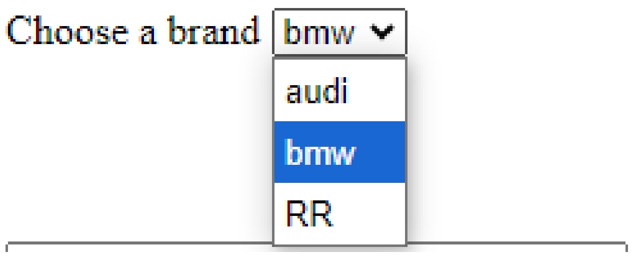


Submit Button

```
<input type="submit" value="Submit">
```

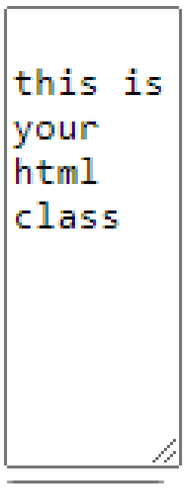
Selections

```
<select name="car">
<option value="audi">audi</option>
<option value="bmw" selected>bmw</option>
<option value="RR">RR</option>
</select>
```



Textarea

```
<textarea name="message" rows="4" cols="50">
Enter your message here...
</textarea>
```



beginfigure[h]

—

Key Competencies: Designing user-friendly and accessible forms.

1.12 Lab Task

Here is a comprehensive form design with multiple sections:

1.12.1 Part 1: Personal Details

- First Name: (Text Input) - Last Name: (Text Input) - Email: (Email Input) - Phone: (Phone Number Input) - Date of Birth: (Date Input) - Gender: (Radio Button Group: Male, Female, Other) - Marital Status: (Dropdown Select: Single, Married, Divorced, Separated)

1.12.2 Part 2: Address Details

- Street Address: (Text Input) - City: (Text Input) - State/Province: (Dropdown Select) - Postal Code: (Text Input)

- Country: (Dropdown Select) - Address Type: (Checkbox Group: Home, Work, Other)

1.12.3 Part 3: Education and Employment

- Highest Level of Education: (Dropdown Select) - Current Occupation: (Text Input) - Current Employer: (Text Input) - Years of Experience: (Number Input)

1.12.4 Part 4: Preferences and Interests

- Favorite Color: (Color Picker) - Hobbies: (Checkbox Group: Reading, Writing, Sports, Music, Travel, Photography) - How did you hear about us? (Radio Group: Social Media, Friend/Family, Online Search, Other) - What are your goals? (Textarea)

1.12.5 Part 5: Additional Information

- Message: (Textarea) - Upload Resume: (File Input) - Terms and Conditions: (Checkbox) - Newsletter Subscription: (Checkbox)

—

2 Lab 02: Website Component

As you build web pages throughout your career, you'll notice that many pages follow similar layouts and structures. This is the outcome of many years of research into user interface design and user experience. Different companies, libraries and frameworks then adopt the resulting best practices.

1

Many examples of these layouts can be seen in the popular Bootstrap framework. However, many other frameworks provide similar designs.

Objectives: Understand common website layouts and implement them in HTML/CSS.

Learning Outcomes: - Identify common web layout patterns. - Implement navbars, carousels, blogs, and dashboards using HTML/CSS. - Explore responsive designs using Bootstrap principles.

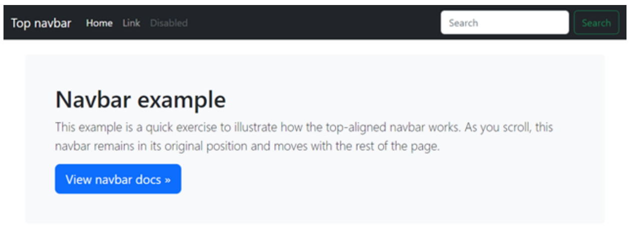
Key Competencies: UI layout design, semantic structure, responsive web patterns.

[1]These layouts are based on established usability research.

2.1 Layout Design

2.1.1 Top Navbar Layout

Websites often have a top navbar layout to provide a set of essential anchor links to the user. These typically link to the main areas of the website, such as product pages, careers pages or contact pages. This provides the visitor to the website with a consistent navigation experience.

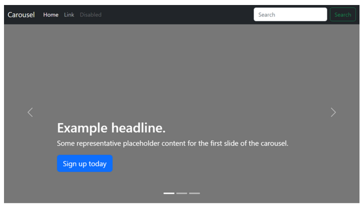


Navbar example

This example is a quick exercise to illustrate how the top-aligned navbar works. As you scroll, this navbar remains in its original position and moves with the rest of the page.

2.1.2 Carousel Layout

Product-focused websites often use a large carousel on their homepage to highlight featured products, discounts and offers. The carousel contains content items that rotate through the carousel area at a fixed interval.

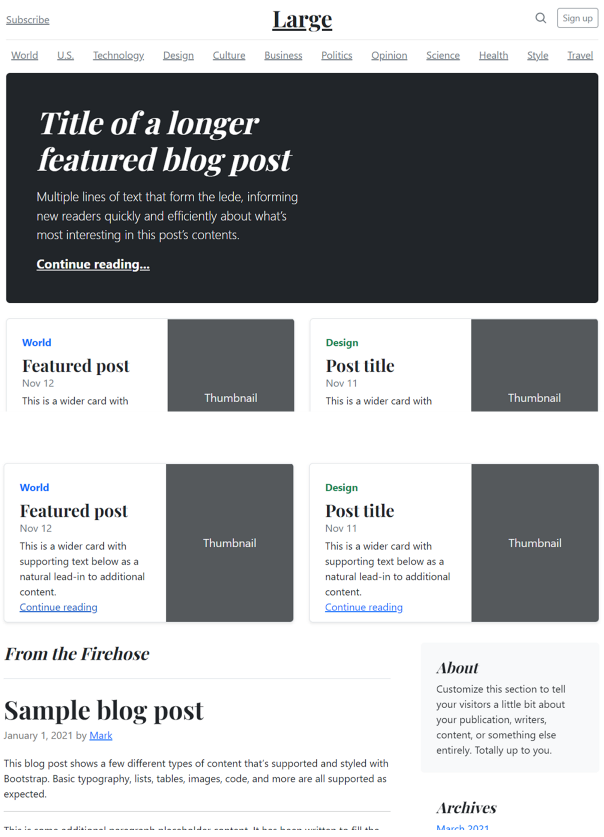


Example headline.

Some representative placeholder content for the first slide of the carousel.

2.1.3 Blog Layout

The blog layout is used to feature multiple content items of differing importance. It is often seen on news websites where new articles will appear daily based on current events. The layout typically features different-sized feature areas followed by article summaries.



Subscribe Large Sign up

World U.S. Technology Design Culture Business Politics Opinion Science Health Style Travel

Title of a longer featured blog post

Multiple lines of text that form the lede, informing new readers quickly and efficiently about what's most interesting in this post's contents.

Continue reading...

World
Featured post
Nov 12
This is a wider card with
Thumbnail

Design
Post title
Nov 11
This is a wider card with
Thumbnail

World
Featured post
Nov 12
This is a wider card with supporting text below as a natural lead-in to additional content.
Continue reading
Thumbnail

Design
Post title
Nov 11
This is a wider card with supporting text below as a natural lead-in to additional content.
Continue reading
Thumbnail

From the Firehose

Sample blog post

January 1, 2021 by Mark

This blog post shows a few different types of content that's supported and styled with Bootstrap. Basic typography, lists, tables, images, code, and more are all supported as expected.

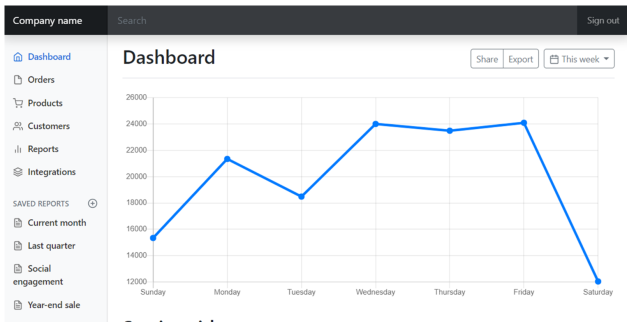
About

Customize this section to tell your visitors a little bit about your publication, writers, content, or something else entirely. Totally up to you.

Archives

2.1.4 Dashboard Layout

Dashboard layouts are often used in enterprise software for managing web applications. They typically feature a sidebar for navigation with the main content area containing forms, graphs, or tables. This layout provides a good user experience for business users.



2.2 More Layouts

You can explore more of these layouts on the Bootstrap examples page in the additional resources. Consider these layouts when building websites and applications to provide the best user experience.

—

2.3 Examples

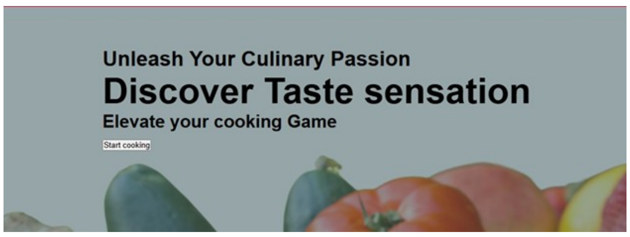
2.3.1 Text with Background Image

HTML

```
<div class="text">
<h3>Unleash Your Culinary Passion</h3>
<h1>Discover Taste Sensation</h1>
<h4>Elevate your Cooking Game</h4>
<button class="button">Start Cooking</button>
</div>
```

CSS

```
.text {
 height: 100vh; width: 100%;
 font-size: 30px;
 padding-left: 20%;
 padding-top: 5%;
 background-image: url('hero.png');
}
```



2.3.2 Centered Image Card

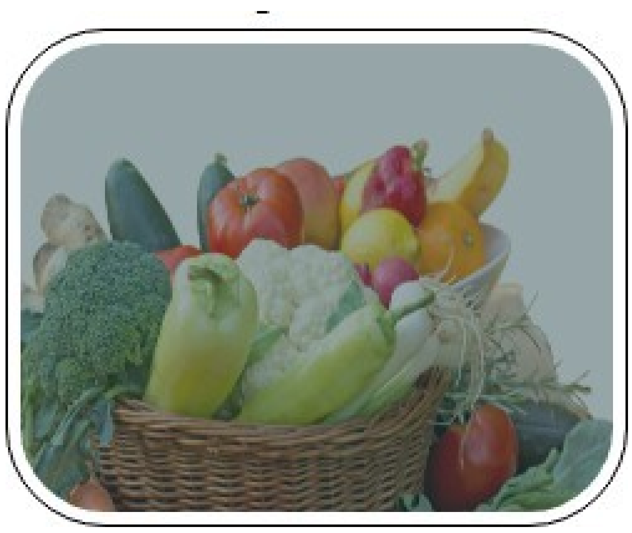
HTML

```
<div class="column">
```

```
<img class="card" src="hero.png">
</div>
```

CSS

```
.card {
 border: 1px solid;
 border-radius: 40px;
 padding: 5px;
 width: 250px; height: 200px;
}
.column {
 display: flex;
 justify-content: center;
 align-items: center;
 flex-wrap: wrap;
}
```



3 Lab 03: CSS

Objectives: - Understand Cascading Style Sheets (CSS). - Learn responsive design principles. - Apply CSS3 features to build a shopping cart web app.

Learning Outcomes: - Style web pages using CSS. - Implement responsive layouts with media queries and units. - Build a complete mini-project using flex and grid.

Key Competencies: Web design, responsive UI, CSS3 techniques.

3.1 Cascading Style Sheets (CSS)

- CSS is used to style HTML elements and control the layout of web pages.
- It separates content (HTML) from design (CSS), making websites more visually appealing and easier to maintain.
- **Tip:** The word cascading means that a style applied to a parent element also applies to all children elements unless overridden.

3.2 Responsive Design in CSS

3.2.1 Introduction to Responsive Design

Responsive design ensures that websites look good on all devices. This is achieved through fluid grids, flexible images, and media queries.

3.2.2 Media Queries

Media queries allow you to apply different styles based on screen size, orientation, resolution, etc.

Example 1: Basic Media Query

```
/* Default style for mobile */
body { font-size: 16px; }

@media (min-width: 768px) {
 body { font-size: 18px; } /* For tablets */
}

@media (min-width: 992px) {
 body { font-size: 20px; } /* For desktops */
}
```

Example 2: Orientation Media Query

```
@media (orientation: landscape) {
 body { background-color: lightblue; }
}
```

3.2.3 Responsive Units

Percentages

```
.container { width: 100%; }
.column { width: 50%; }
```

em Units

```
.container { font-size: 16px; }
.text { font-size: 2em; } /* 32px */
```

rem Units

```
html { font-size: 16px; }
.box { padding: 1.5rem; } /* 24px */
```

Viewport Units

```
.full-height { height: 100vh; }
.half-width { width: 50vw; }
```

3.2.4 Practical Examples of Responsive Design

Example 1: Responsive Columns

```
.container { width: 100%; display: flex; flex-wrap: wrap; }
.column { width: 100%; }
@media (min-width: 600px) {
 .column { width: 50%; }
}
```

Example 2: Font Scaling

```
html { font-size: 16px; }
.content { font-size: 1rem; }
h1 { font-size: 2em; }
```

Example 3: Full-Screen Section

```
.hero {
 height: 100vh; background-color: lightcoral;
 display: flex; align-items: center; justify-content: center;
}
```

Example 4: Adaptive Widths

```
.sidebar { width: 30vw; background: #f4f4f4; }
.main-content { width: 70vw; background: #fff; }
```

3.2.5 Exercise: Responsive Card

```
.card {
 padding: 1.5rem; background: #fff;
 box-shadow: 0 4px 8px rgba(0,0,0,0.2);
 max-width: 90%; margin: auto;
}
@media (min-width: 600px) { .card { max-width: 80%; } }
@media (min-width: 900px) { .card { max-width: 60%; } }
```

3.3 Summary of Key Units

- Percentage (%) – relative to parent dimensions.
- em – relative to parent font size.
- rem – relative to root font size.
- vw / vh – relative to viewport width/height.

3.4 Lab 3

3.4.1 Task

Build a responsive web application for a shopping cart with registration, login, catalog, and cart pages using CSS3, flex, and grid.

3.4.2 Solution

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="stylesheet" href="./style.css">
<title>Home - FBS</title>
</head>
<body>
<div class="wrapper">
<div class="container">
<header>
```

```
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tr>
<th width="20%"><img src="fbs.png" alt="FBS LOGO"
 width="130" height="100"/></th>
<th colspan=4>
<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>
</th>
</tr>
</table>
</header>
<nav>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tbody align="center"  style="font-weight:bold;font-size:18px;"">
<tr>
<td width="20%"><hr><a href="index.html">Home</a><hr></td>
<td width="20%"><hr><a href="login.html">Login</a><hr></td>
<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>
<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>
</tr>
</tbody>
</table>
</nav>
</div>
<div class="container1">
<div class="sidebar1"></div>
<div class="container2">
<main>
<center>
<h2>Welcome to FBS e-Book’s Website</h2>
<p>Shopping at <font size=5>FBS</font> can be both <font size=5>fun</font>
and <font size=5>savings</font>.</br>Shop with us in this special <font
size=5>discount</font> season and save upto <font size=5>90%</font> on all
purchases.</br></p>
<br/><br/><br/><br/><br/><br/><br/><br/>
</main>
</div>
<div class="sidebar2"></div>
</div>
<footer><font color="white">(C) 2024 All rights reserved by FBS
ebooks</font></footer>
</div>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="stylesheet" href="./style.css">
<title>Login - FBS</title>
</head>
<body>
<div class="wrapper">
<div class="container">
<header>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tr>
<th width="20%"><img src="fbs.png" alt="FBS LOGO"  width="130" height="100"
<th colspan=4>
<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>
</th>
</tr>
</table>
</header>
<nav>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tbody align="center"  style="font-weight:bold;font-size:18px;"">
<tr>
<td width="20%"><hr><a href="index.html">Home</a><hr></td>
<td width="20%"><hr><a href="login.html">Login</a><hr></td>
<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>
<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>
</tr>
</tbody>
</table>
</nav>
</div>
<div class="container1">
<div class="sidebar1"></div>
<div class="container2">
<main>
<center><br>
<h3> Login Details</h3> <br/>
<form name="f1">
<table width="100%" align="center" >
<tr>
<td> User Name : </td>
```

```
<td> <input type="text" name="username"></td>
</tr>
<tr><td><br></td></tr>
<tr>
<td> Password : </td>
<td> <input type="password" name="password"></td>
</tr>
<tr><td><br></td></tr>
<tr><td></td>
<td><input type="submit" value="SUBMIT">
<input type="reset" value="RESET"></td>
</tr>
</table>
</form>
</center>
</main>
</div>
<div class="sidebar2"></div>
</div>
<footer><font color="white">(C) 2024 All rights reserved by FBS
ebooks</font></footer>
</div>
</body>
</html>
```

registration.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="stylesheet" href="./style.css">
<title>Registration - FBS</title>
</head>
<body>
<div class="wrapper">
<div class="container">
<header>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tr>
<th width="20%"><img src="fbs.png" alt="FBS LOGO"
  width="130" height="100"/></th>
<th colspan=4>
<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>
</th>
```

```
</tr>
</table>
</header>
<nav>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tbody align="center"  style="font-weight:bold;font-size:18px;"">
<tr>
<td width="20%"><hr><a href="index.html">Home</a><hr></td>
<td width="20%"><hr><a href="login.html">Login</a><hr></td>
<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>
<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>
</tr>
</tbody>
</table>
</nav>
</div>
<div class="container1">
<div class="sidebar1"></div>
<div class="container2">
<main>
<center><br>
<h3>Registration Form </h3>
<br/>
<form name="f1">
<table cellpadding="1" align="center" >
<tr><td> Name:*</td>
<td><input type="text" name="username"></td></tr>
<tr><td>Password:*</td>
<td><input type="password" name="password"></td></tr>
<tr><td>Email ID:*</td>
<td><input type="text" name="email"></td></tr>
<tr><td>Phone Number:*</td>
<td><input type="text" name="phno"></td></tr>
<tr><td valign="top">Gender:*</td>
<td><input type="radio" name="radio" value="1">Male &nbsp;&nbsp;
<input type="radio" name="radio" value="2">Female</td></tr>
<tr> <td valign="top">Language Known:*</td>
<td> <input type="checkbox" name="checkbox" value="English">English<br/>
<input type="checkbox" name="checkbox" value="Telugu">Telugu<br>
<input type="checkbox" name="checkbox" value="Hindi">Hindi<br>
<input type="checkbox" name="checkbox" value="Tamil">Tamil
</td></tr>
<tr> <td valign="top">Address:*</td>
<td><textarea name="address"></textarea></td>
<tr><td></td><td><input type="submit" value="submit" hspace="10">
<input type="reset" value="reset"></td></tr>
```

```
 <tr> <td colspan=2 >*<font color="#FF0000">fields are mandatory</font>
 </td>
 </tr>
 </table>
 </form>
 </center>
 </main>
 </div>
 <div class="sidebar2"></div>
 </div>
 <footer><font color="white">(C) 2024 All rights reserved by
 FBS ebooks</font></footer>
 </div>
 </body>
 </html>
```

cart.html

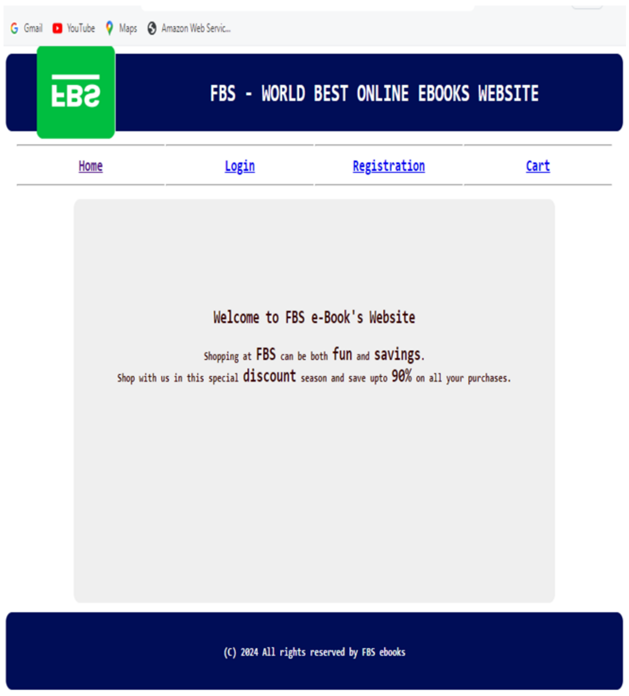
```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="stylesheet" href="./style.css">
<title>Cart - FBS</title>
</head>
<body>
<div class="wrapper">
<div class="container">
<header>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tr>
<th width="20%"><img src="fbs.png" alt="FBS LOGO"  width="130"
 height="100"/></th>
<th colspan=4>
<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>
</th>
</tr>
</table>
</header>
<nav>
<table width="100%" align="center" cellpadding="0" cellspacing="2">
<tbody align="center"  style="font-weight:bold;font-size:18px;"">
<tr>
<td width="20%"><hr><a href="index.html">Home</a><hr></td>
<td width="20%"><hr><a href="login.html">Login</a><hr></td>
```

```
<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>
<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>
</tr>
</tbody>
</table>
</nav>
</div>
<div class="container1">
<div class="sidebar1"></div>
<div class="container2">
<main>
<center>
<h3>Cart</h3>
<table width="100%" align="center" >
<tbody>
<tr>
<th width="40%"><hr>BookName<hr></th>
<th width="20%"><hr>Price<hr></th>
<th width="20%"><hr>Quantity<hr></th>
<th width="20%"><hr>Amount<hr></th> </tr>
</tbody>
<tbody align=center>
<tr> <td>Java Programming </td>
<td>Rs. 2300/-</td>
<td>2</td>
<td>Rs. 4600/-</td></tr>
<tr><td>Web Technologies</td>
<td>Rs. 3000/-</td>
<td>1</td>
<td>Rs. 3000/-</td></tr>
<tr><td></td>
<td><hr><font color="#996600">Total Amount:</font><hr></td>
<td><hr>3<hr></td>
<td><hr>Rs. 7600/-<hr></td> </tr>
</tbody>
</table>
</center>
</main>
</div>
<div class="sidebar2"></div>
</div>
<footer><font color="white">(C) 2024 All rights reserved by
FBS ebooks</font></footer>
</div>
</body>
</html>
```

style.css

```css
body{
 font-family: monospace;
}

main {
 background-color: #efefef;
 color: #330000;
 margin-left: 10px;
 height: 60vh;
}

header, footer {
 background-color: #000d57;
 color: #fff;
 padding: 1rem;
 height: 50px;
}

header, nav{
 margin-bottom: 10px;
 flex-basis: 50%;
}

footer{
 margin-top: 10px;
}
nav {
 background-color: #fff;
 color: #000;
 padding: 1rem;
 height: 20px;
}

.sidebar1, .sidebar2 {
 flex-basis: 10%;
 background-color: #fff;
 color: #000;
}

.sidebar2{
 margin-left: 10px;
}

.container1{
 display: flex;
}

.container2 {
 display: flex;
 flex-direction: column;
 flex: 1;
}

header, nav, main, .sidebar1, .sidebar2, footer{
 display: flex;
 align-items: center;
 justify-content: center;
 border-radius: 10px;
}

.wrapper {
 display: flex;
 flex-direction: column;
 font-weight: 600;
}
```

3.4.3 Output



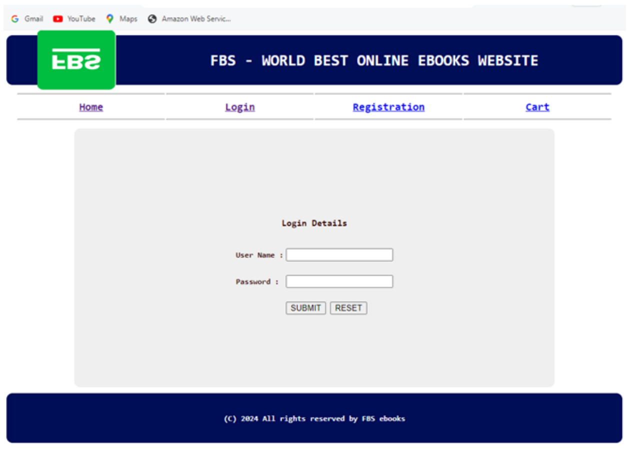
FBS - WORLD BEST ONLINE EBOOKS WEBSITE

Home Login Registration Cart

Welcome to FBS e-Book's Website

Shopping at FBS can be both fun and savings.
Shop with us in this special discount season and save upto 90% on all your purchases.

(C) 2024 All rights reserved by FBS ebooks



FBS - WORLD BEST ONLINE EBOOKS WEBSITE

Home Login Registration Cart

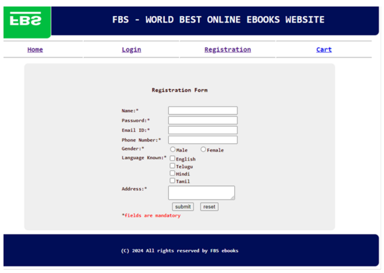
Login Details

User Name :

Password :

SUBMIT RESET

(C) 2024 All rights reserved by FBS ebooks



FBS - WORLD BEST ONLINE EBOOKS WEBSITE

Home Login Registration Cart

Registration Form

Name:*

Password:*

Email ID:*

Phone Number:*

Gender:* Male Female

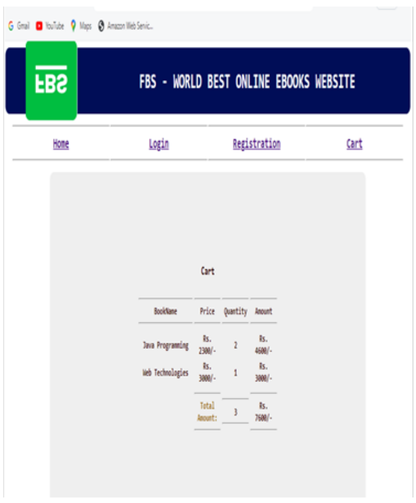
Language Known:* English Telugu Hindi Tamil

Address:*

submit reset

*fields are mandatory

(C) 2024 All rights reserved by FBS ebooks



FBS - WORLD BEST ONLINE EBOOKS WEBSITE

Home Login Registration Cart

Cart

BookName	Price	Quantity	Amount
Java Programming	Rs. 2300/-	2	Rs. 4600/-
Web Technologies	Rs. 3000/-	1	Rs. 3000/-
	Total Amount:	3	Rs. 7600/-

4 Lab 04: Flexbox CSS

Objectives: - Understand Flexbox as a layout model.
- Learn properties such as alignment, direction, wrapping, and spacing.
- Apply Flexbox to build responsive layouts.

Learning Outcomes: - Create responsive web designs using Flexbox. - Apply alignment, direction, and spacing properties.
- Compare absolute vs relative CSS units.

Key Competencies: Responsive UI design, Flexbox layout, CSS measurement units.

4.1 Understanding Flexbox

Much like the div and box container that you can create using HTML, flexbox is a type of container. Flexbox can overcome the limitations caused by containers such as block and inline because it does a better job of scaling over larger web pages and also provides more dynamic control of the containers. This is because it can grow, shrink and align the items inside it which give better control to the programmer over the contents and styling of the items inside the container.

Before learning about the common layouts built using the flexbox, it is important to understand the properties inside it and how flexbox works. Let’s examine some of the important characteristics of flexboxes and the properties that can be used to configure them.

Flexbox is single-dimensional, which means you can align it either along a row or a column and it is set to row alignment by default. There are two axes, the main and cross-axis, much like the x and y-axis used in coordinate geometry. When aligned along the row, the horizontal axis is called the main axis and the vertical axis is called the cross axis. For the items present inside the flexbox container, the placement starts from the top-left corner moving along the main or horizontal axis. When the row is filled, the items continue to the next row. Note that with the help of a property called flex-direction, you can instead flip the main axis to run vertically and the cross axis will then be horizontal. In such a case, the items will start from top left and move down along the vertical main axis. The properties you choose will help better control alignment, spacing, direction and eventually styling of the container and items present inside it.

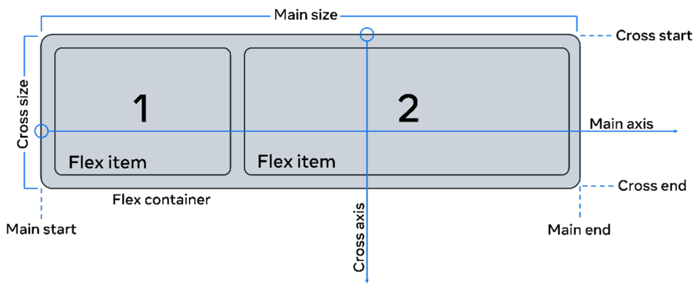


Image source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/aa-basics-and-terminology

Let’s now examine some of the important properties that will allow you to configure a flexbox.

4.2 Flexbox Properties

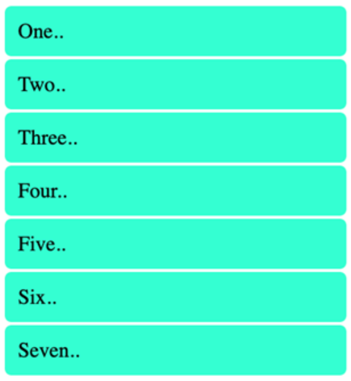
Original HTML

```
<body>
<div class="flex-container">
<div class="box box1">One..</div>
<div class="box box2">Two..</div>
<div class="box box3">Three..</div>
<div class="box box4">Four..</div>
<div class="box box5">Five..</div>
<div class="box box6">Six..</div>
<div class="box box7">Seven..</div>
</div>
</body>
```

Original CSS

```
.box {
 padding: 10px;
 border-radius: 5px;
 margin: 2px;
 background-color: aquamarine;
}
```

Output Seven boxes aligned horizontally by default.



```
One..
Two..
Three..
Four..
Five..
Six..
Seven..
```

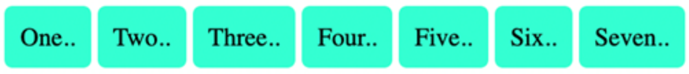
There are seven div containers inside the HTML file.

The corresponding CSS file contains rules for all seven div tags that have the box class. Note how two class names are given for each of the tags, one that is common among all classes and another independent of it. The style is applied to all the containers. Now let's add properties to the flex container by converting it into flex.

4.2.1 display: flex;

```
.flex-container {
 display: flex;
}
```

Items now flow left-to-right inside the flex container.

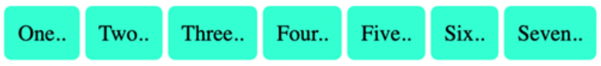


4.3 Alignment Properties

Four important alignment properties: - `justify-content` – alignment on main axis. - `align-items` – alignment on cross axis. - `align-self` – alignment of individual items. - `align-content` – spacing between multiple flex lines.

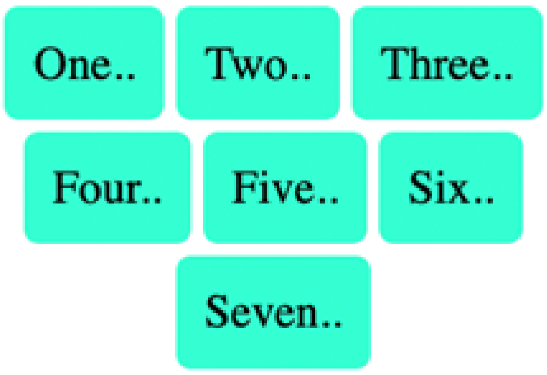
4.3.1 justify-content

```
.flex-container {
 display: flex;
 justify-content: center;
}
```



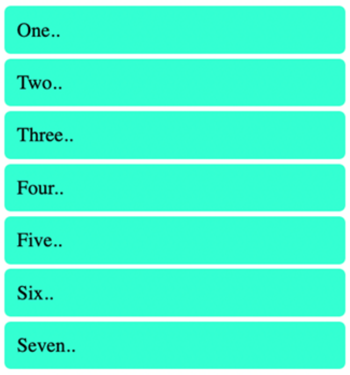
4.3.2 flex-wrap

```
.flex-container {
 display: flex;
 justify-content: center;
 flex-wrap: wrap;
}
```



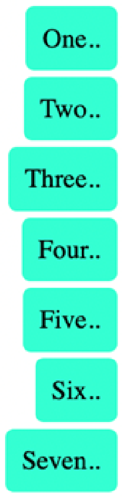
4.3.3 flex-direction

```
.flex-container {
 display: flex;
 flex-direction: column;
 flex-wrap: wrap;
}
```



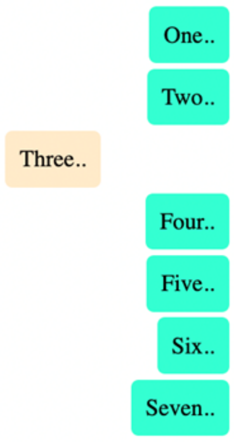
4.3.4 align-items

```
.flex-container {
 display: flex;
 align-items: flex-end;
}
```



4.3.5 align-self

```
.box3 {
 background-color: blanchedalmond;
 align-self: center;
}
```

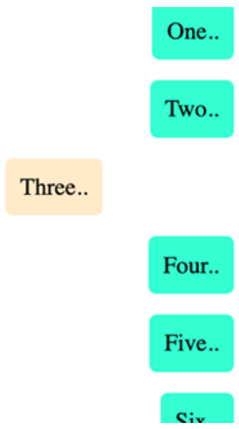


4.3.6 gap

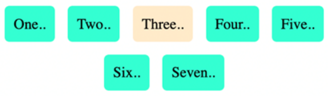
```
.flex-container {
 display: flex;
 gap: 10px;
}
```

Shorthand flex property

```
.flex-container { flex: 0 1 auto; }
.box3 { flex: 1 1 auto; }
```



Here, `flex-grow` is 1, so box3 expands to fill free space.



4.4 CSS Units of Measurement

4.4.1 Absolute Units

Absolute units are constant across different devices and have a fixed size. They are useful for activities like printing a page. They are not so suitable when it comes to the wide variety of devices in use today that have different viewport sizes. Because of this, absolute units are used when the size of the web page is known and will remain constant.

The table for absolute units can be seen below:

Unit	Name	Comparison
Q	Quarter-millimeter	$1Q = 1/40$ cm
mm	Millimeter	$1mm = 1/10$ cm
cm	Centimeter	$1cm = 37.8px$
in	Inch	$1in = 2.54cm = 96px$
pc	Pica	$1pc = 1/6$ in
pt	Point	$1pt = 1/72$ in
px	Pixel	$1px = 1/96$ in

Table 1: Absolute units in CSS

4.4.2 Relative Units

When you create a web page, you will almost never have only a single element present inside it. Even in case of containers such as flexboxes and grids, there's usually more than one element present that rules are applied to. Relative values are defined 'in relation' to the other elements present inside the parent element. Additionally, they are defined 'in relation' to the viewport or the size of the visible web page. Given the dynamic nature of web pages today and the variable size of devices in use, relative units are the go-to option in many cases. Below is a list of some of the important relative units

- em – relative to parent font size.
- rem – relative to root font size.
- vw – 1% of viewport width.
- vh – 1% of viewport height.
- % – relative to parent element.

Many of these units are used in terms of the relative size of fonts. Some units are more suitable depending on the relative context. Like when the dimensions of the viewport are important, it's more appropriate to use vw and vh. In a broader context, the relative units you will see most frequently used are percentage, em, vh, vw and rem.

Much like the absolute and relative units discussed above, certain properties have their own set of acceptable values that need to be taken into account. For example, color-based properties such as backgroundcolor will have values such as hexadecimal, rgb(), rgba(), hsl(), hsla() and so on. Each property should be explored on an individual basis and practicing with the code will help you to decide which of these units of measurement are the most suitable choice.

5 Lab 05: Bootstrap CSS

Objectives: - Understand Bootstrap and why it is useful. - Learn grid system, breakpoints, and components. - Apply Bootstrap to build responsive web applications.

Learning Outcomes: - Use Bootstrap grid system and responsive utilities. - Apply components like buttons, navbars, cards, and forms. - Develop a complete responsive project using Bootstrap.

Key Competencies: Bootstrap framework, responsive design, UI components.

5.1 Introduction to Bootstrap

- **What is Bootstrap?**
  - Free, open-source front-end framework for building responsive websites.
  - Includes components, grid system, and utilities.
- **Why Bootstrap?**
  - Mobile-first design.
  - Speeds up design process with reusable code.

Bootstrap defines several breakpoints for common device sizes.

5.2 Bootstrap Screen Size Breakpoints

Breakpoint	Class Prefix	Screen Size	Description
Extra small	xs	$<$576px	Phones
Small	sm	$\geq$576px	Larger phones
Medium	md	$\geq$768px	Tablets
Large	lg	$\geq$992px	Desktops
Extra large	xl	$\geq$1200px	Large desktops
XXL	xxl	$\geq$1400px	Extra-large desktops

Table 2: Bootstrap Breakpoints

5.3 How Breakpoints Work

- Mobile-first: designed for small screens, then scales up.
- Layout adjustments triggered at breakpoints.

Example: Grid with Breakpoints

```
<div class="container">
<div class="row">
<div class="col-12 col-sm-6 col-md-4">Column 1</div>
<div class="col-12 col-sm-6 col-md-4">Column 2</div>
<div class="col-12 col-sm-6 col-md-4">Column 3</div>
```

```
</div>
</div>
```

- On extra small screens (xs or < 576px): All columns will be stacked in a single row because of `col-12` (full width).
- On small screens (sm or ≥ 576px): The columns will be arranged in 2 columns (`col-sm-6`).
- On medium screens (md or ≥ 768px): The columns will fit in 3 columns (`col-md-4`).

Hiding/Showing Content Based on Screen Size:
You can show or hide elements based on screen size using utility classes like `.d-none`, `.d-sm-block`, `.d-md-none`, etc.

Example:

```
<p class="d-none d-md-block">
This paragraph is visible only on medium screens and larger.
</p>
```

This flexibility allows you to tailor your web pages for a variety of devices, ensuring a consistent user experience across different screen sizes.

Summary of Class Prefixes:

- `col-xs-*`: For screens smaller than 576px (default behavior).
- `col-sm-*`: For screens larger than 576px (small devices).
- `col-md-*`: For screens larger than 768px (medium devices like tablets).
- `col-lg-*`: For screens larger than 992px (desktops).
- `col-xl-*`: For screens larger than 1200px (large desktops).
- `col-xxl-*`: For screens larger than 1400px (extra large desktops).

Use Case: Show/Hide Content

```
<p class="d-none d-md-block">
Visible only on medium and larger screens.
</p>
```

5.4 Bootstrap Grid System

5.4.1 Concept

- The grid system is the core of Bootstrap's layout system, based on a 12-column layout..
- It allows you to create responsive designs that adjust based on screen size.

5.4.2 How to Teach

- **Explain:** Introduce the 12-column grid and how Bootstrap uses different classes (`col`, `col-sm`, `col-md`, `col-lg`, etc.) to specify column sizes for different screen widths.
- **Show Example:**

```
<div class="container">
<div class="row">
<div class="col-sm-4">Column 1</div>
<div class="col-sm-4">Column 2</div>
<div class="col-sm-4">Column 3</div>
</div>
</div>
```

Live Demonstration: Create a basic layout with a few columns and resize the browser window to show how the columns respond on different devices.

Activity: Create a webpage with 2 columns on small screens and 4 on large screens.

5.5 Bootstrap Components

5.5.1 Concept

Pre-built components: buttons, navbars, cards, modals.

5.5.2 How to Teach

Explain: Show a few components, how to use them, and how they improve the design consistency across websites.

Demonstrate Key Components:

Buttons

```
<button class="btn btn-primary">Primary</button>
<button class="btn btn-danger">Danger</button>
```

Cards

```
<div class="card" style="width: 18rem;">
<img src="placeholder.png" class="card-img-top" alt="Image">
<div class="card-body">
<h5 class="card-title">Card Title</h5>
<p class="card-text">Some text.</p>
<a href="#" class="btn btn-primary">Go somewhere</a>
</div>
</div>
```

Navbar

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

Activity: Build a webpage with navbar, buttons, and cards.

5.6 Bootstrap Utilities

Bootstrap provides utility classes for quick styling and layout adjustments like margins, padding, colors, and text alignment

Spacing

```
<div class="p-3 m-2 bg-primary text-white">Padding and Margin</div>
```

Text Alignment

```
<p class="text-center">Centered text</p>
```

Background Colors

```
<div class="bg-danger text-white p-2">Red background</div>
```

5.7 Bootstrap Forms

Example

```
<form>
<div class="form-group">
<label>Email</label>
<input type="email" class="form-control" placeholder="Enter email">
</div>
</form>
```

Activity: Build a registration form with name, email, password.

5.8 Media Queries and Responsive Design

Concept:

- Bootstrap's mobile-first approach ensures that your web pages look good on any device.

How to Teach:

- **Explain:** Show how media queries are built into Bootstrap's grid system, making it easy to create responsive designs.
- **Example:** Create a page with columns that stack vertically on small screens but are side-by-side on larger screens.

Objective: Introduction to Bootstrap, covering its grid system, components, utilities, and responsive design principles through hands-on exercises.

Sample Code:

```
<!DOCTYPE html>
<html>
<head>
<title>Responsive Design Example</title>
<link rel="stylesheet" href="https://cdn.jsdelivr.net
/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<div class="container">
<div class="row">
<div class="col-sm-12 col-md-6">Column 1</div>
```

```
<div class="col-sm-12 col-md-6">Column 2</div>
</div>
</div>
</body>
</html>
```

5.9 quad Step 1: Setting Up Bootstrap

1. Create a new HTML file.
2. Include Bootstrap via CDN in the <head> section:

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com
/bootstrap/4.5.2/css/bootstrap.min.css">
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js">
</script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3
/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap
/4.5.2/js/bootstrap.min.js"></script>
```

5.10 quad Step 2: Bootstrap Grid System (Responsive Layouts)

Bootstrap's grid system allows you to create responsive layouts with up to 12 columns across the page.

Key Concepts:

- **Rows:** Containers for columns, ensuring proper alignment.
- **Columns:** Spanning from 1 to 12 across the page.
- **Breakpoints:** Control how columns behave on different screen sizes (xs, sm, md, lg, xl).

Examples:

1. Basic Grid Layout:

```
<div class="container">
<div class="row">
<div class="col">Column 1</div>
<div class="col">Column 2</div>
<div class="col">Column 3</div>
</div>
```

```
      </div>
```

2. Responsive Columns:

```
<div class="row">
<div class="col-sm-6 col-lg-4">Small-6 / Large-4</div>
<div class="col-sm-6 col-lg-4">Small-6 / Large-4</div>
<div class="col-sm-12 col-lg-4">Small-12 / Large-4</div>
</div>
```

Tasks:

1. Create a 12-column layout that collapses into 6 columns on medium screens (`col-md-6`) and 3 columns on large screens (`col-lg-4`).
2. Use Bootstrap's grid system to create a webpage with a sidebar (25% width) and main content area (75% width) that stacks vertically on smaller screens.

5.11 quad Step 3: Bootstrap Components

1. Buttons

Bootstrap provides predefined button styles.

Example:

```
<button class="btn btn-primary">Primary Button</button>
<button class="btn btn-success">Success Button</button>
```

Task: Create a button group with buttons of different colors (primary, success, danger, etc.).

2. Cards

Cards are versatile containers for content like text, images, and buttons.

Example:

```
<div class="card" style="width: 18rem;">
<img src="https://via.placeholder.com/150"
 class="card-img-top" alt="Image">
<div class="card-body">
```

```
<h5 class="card-title">Card Title</h5>
<p class="card-text">Some example text to build on the card title.</p>
<a href="#" class="btn btn-primary">Go somewhere</a>
</div>
</div>
```

Task: Create a grid of cards (3 cards per row) with titles, descriptions, and buttons.

3. Alerts

```
<div class="alert alert-success">Success alert!</div>
<div class="alert alert-danger">Danger alert!</div>
```

Task: Create a form with success and error alerts displayed after submitting data.

4. Navbars

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
<a class="navbar-brand" href="#">Navbar</a>
<button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarNav">
<ul class="navbar-nav">
<li class="nav-item active"><a class="nav-link" href="#">Home</a></li>
<li class="nav-item"><a class="nav-link" href="#">Features</a></li>
<li class="nav-item"><a class="nav-link" href="#">Pricing</a></li>
</ul>
</div>
</nav>
```

Task: Create a responsive navigation bar that collapses into a hamburger menu on small screens and includes at least four links.

5. Modals

```
<button type="button" class="btn btn-primary"
 data-toggle="modal" data-target="#myModal">
Open Modal
</button>
```

```
<div class="modal fade" id="myModal" tabindex="-1"
 aria-labelledby="myModalLabel" aria-hidden="true">
<div class="modal-dialog">
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title" id="myModalLabel">Modal Title</h5>
<button type="button" class="close" data-dismiss="modal"
 aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
</div>
<div class="modal-body">This is the modal content.</div>
<div class="modal-footer">
<button type="button" class="btn btn-secondary"
 data-dismiss="modal">Close</button>
</div>
</div>
</div>
</div>
```

Task: Create a modal that opens when a button is clicked and contains a form.

5.12 quad Step 4: Bootstrap Utilities

1. Spacing Utilities

```
<div class="p-5 m-3 bg-primary text-white">This has
 padding and margin.</div>
```

Task: Create a container with different levels of padding and margins using Bootstrap utilities.

2. Text Alignment

```
<div class="text-center">This text is centered.</div>
<div class="text-right">This text is right-aligned.</div>
```

Task: Create a section with different text alignment (left, center, and right).

5.13 quad Step 5: Typography

Bootstrap provides a wide range of typography styles.

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>

<blockquote class="blockquote">
<p>This is a blockquote.</p>
<footer class="blockquote-footer">Someone famous</footer>
</blockquote>
```

Task: Style different paragraphs using Bootstrap's typography classes and create a blockquote.

5.14 quad Step 6: Bootstrap Forms

1. Basic Form

```
<form>
<div class="form-group">
<label for="exampleInputEmail1">Email address</label>
<input type="email" class="form-control" id="exampleInputEmail1"
placeholder="Enter email">
</div>
<div class="form-group">
<label for="exampleInputPassword1">Password</label>
<input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Task: Create a login form with email and password fields and style it using Bootstrap's form components.

2. Inline Form

```
<form class="form-inline">
<div class="form-group mb-2">
<label for="staticEmail2" class="sr-only">Email</label>
<input type="text" class="form-control" id="staticEmail2"
 value="email@example.com">
</div>
<button type="submit" class="btn btn-primary mb-2">
Confirm identity</button>
</form>
```

Task: Build a search bar using Bootstrap's inline form utilities.

5.15 quad Step 7: Bootstrap Tables

```
<table class="table">
<thead>
<tr>
<th>#</th>
<th>First Name</th>
<th>Last Name</th>
<th>Username</th>
</tr>
</thead>
<tbody>
<tr>
<th>1</th>
<td>John</td>
<td>Doe</td>
<td>@johndoe</td>
</tr>
</tbody>
</table>
```

Task: Create a table to display a list of students, including columns for name, roll number, and grade.

5.16 quad Final Project

So now you have created a complete responsive webpage using Bootstrap. The page should include:

- A navbar
- A grid layout
- A set of cards
- A form
- A footer

Tip: Visit Bootstrap Documentation.

6 Lab 06: JavaScript

Objectives: - Understand the Document Object Model (DOM). - Learn how JavaScript interacts with HTML through the DOM. - Explore DOM methods for selecting and manipulating elements.

Learning Outcomes: - Define DOM and explain its purpose in web development. - Demonstrate how to access and modify DOM elements with JavaScript. - Use DOM selection methods (`getElementById`, `querySelector`, etc.).

Key Competencies: DOM tree structure, JavaScript DOM methods, event-driven manipulation.

6.1 Introduction to the DOM

6.1.1 What is the DOM?

The Document Object Model (DOM) is a programming interface that represents an HTML or XML document as a structured tree of objects. Each HTML element becomes a *node* in the DOM tree, with parent-child relationships.

Example HTML Document

```
<!DOCTYPE html>
<html>
<head>
<title>DOM Introduction</title>
</head>
<body>
<h1 id="title">Hello, DOM!</h1>
<p class="description">This is a
brief introduction to the Document
 Object Model.</p>
</body>
</html>
```

Example DOM Tree Representation

```
Document
 html
 head
    title
 body
 h1
 p
```

6.2 Purpose of the DOM in Web Development

The DOM allows JavaScript to access, modify, or add elements dynamically.

Example: Updating Text Using JavaScript

```
// Select the element with the id "title"
let title = document.getElementById("title");
// Update the text content
title.textContent = "Welcome to the DOM!";
```

Resulting HTML:

```
<h1 id="title">Welcome to the DOM!</h1>
```

6.3 Basic Terminology

- **Document**: Entire HTML page loaded in browser.
- **Element**: Any HTML tag (<div>, <p>, <h1>).
- **Node**: Single point in DOM tree (element, text, or attribute).
- **Attributes**: Properties like id, class, src.
- **Methods**: DOM API functions (`getElementById()`, `appendChild()`).
- **Properties**: Characteristics of nodes (`textContent`, `innerHTML`).

Example of Properties and Methods

```
let description = document.querySelector(".description");

// Properties
console.log(description.textContent);
description.textContent = "Updated description using DOM properties.";

// Methods
let newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph
added using a DOM method.";
document.body.appendChild(newParagraph);
```

6.4 DOM Tree Structure

The HTML document is structured hierarchically like a family tree.

Example: HTML + DOM Tree

```
<!DOCTYPE html>
```

```
<html>
<body>
<div id="container">
<h2>Welcome</h2>
<p>This is a paragraph.</p>
<ul>
<li>Item 1</li>
<li>Item 2</li>
</ul>
</div>
</body>
</html>
```

DOM Tree Representation

```
Document
 html
 body
 div (id="container")
 h2
 p
 ul
 li
 li
```

JavaScript Manipulation Example

```
// Add border to the container
let container = document.getElementById("container");
container.style.border = "1px solid black";

// Add new <li> to the <ul>
let ul = container.querySelector("ul");
let newItem = document.createElement("li");
newItem.textContent = "Item 3";
ul.appendChild(newItem);
```

6.5 DOM Methods for Selecting Elements

JavaScript provides multiple ways to select DOM elements:

6.5.1 Selecting Elements by ID, Class, Tag, and Attributes

getElementById()

```
<h1 id="mainTitle">Welcome to the Website</h1>

let title = document.getElementById("mainTitle");
title.textContent = "Updated Title";
```

getElementsByClassName()

```
<div class="card">Card 1</div>
<div class="card">Card 2</div>

let cards = document.getElementsByClassName("card");
for (let card of cards) {
 card.style.backgroundColor = "lightblue";
}
```

getElementsByTagName()

```
<p>Paragraph 1</p>
<p>Paragraph 2</p>

let paragraphs = document.getElementsByTagName("p");
for (let paragraph of paragraphs) {
 paragraph.style.fontWeight = "bold";
}
```

querySelector()

```
<div class="box special">Special Box</div>
<div class="box">Regular Box</div>

let specialBox = document.querySelector(".special");
specialBox.style.border = "2px solid red";
```

querySelectorAll()

```
<li class="item">Item 1</li>
<li class="item">Item 2</li>
<li class="item special">Special Item</li>

let items = document.querySelectorAll(".item");
items.forEach((item) => {
 item.style.color = "green";
});
```

6.5.2 Explanation and Comparison of Query Selectors

- **Specificity:**
  - `getElementById` → most specific.
  - `getElementsByClassName` → group by class.
  - `getElementsByTagName` → broader, tag-based.
  - `querySelector/querySelectorAll` → CSS selectors, flexible.
- **Performance:**
  - `getElementById`, `getElementsByClassName` are fastest.
  - `querySelector`, `querySelectorAll` are slightly slower but more flexible.

6.6 Navigating the DOM Tree

Once elements are selected, you can navigate between them within the DOM hierarchy. Understanding the relationships in the DOM is key for moving across elements.

6.6.1 Key DOM Navigation Properties

parentNode Accesses the parent node of the selected element.

Listing 1: Example of parentNode

```
<div id="container">
<p>Content inside container</p>
</div>
```

```
let paragraph = document.querySelector("p");
let parentDiv = paragraph.parentNode;
parentDiv.style.border = "1px solid black";
```

childNodes Returns a NodeList of all child nodes, including whitespace.

```
<div id="content">
<p>Paragraph 1</p>
<p>Paragraph 2</p>
</div>
```

```
let content = document.getElementById("content");
content.childNodes.forEach((node) => {
   console.log(node);
});
```

firstChild and lastChild

```
let firstChild = content.firstChild;
console.log(firstChild);
```

firstElementChild and lastElementChild

```
let firstElement = content.firstElementChild;
console.log(firstElement);
```

nextSibling and previousSibling

```
<p id="para1">First paragraph</p>
<p id="para2">Second paragraph</p>
```

```
let para1 = document.getElementById("para1");
let next = para1.nextSibling;
console.log(next);
```

nextElementSibling and previousElementSibling

```
let para1 = document.getElementById("para1");
let nextElement = para1.nextElementSibling;
console.log(nextElement);
```

6.6.2 Summary Example: Selection and Navigation

```
<div id="menu">
<ul>
<li class="item">Home</li>
<li class="item">About</li>
<li class="item">Contact</li>
</ul>
</div>
```

```
let menu = document.getElementById("menu");
let ul = menu.firstElementChild;

ul.childNodes.forEach((node) => {
   if (node.nodeType === Node.ELEMENT_NODE) {
      node.style.color = "blue";
   }
});
```

—

6.7 Best Practices for Navigating the DOM

- Use nodeType checking to avoid whitespace nodes.
- Prefer `firstElementChild`, `lastElementChild`, etc.
- Minimize unnecessary whitespace in HTML.
- Use `querySelector` or `querySelectorAll` for direct targeting.

—

6.8 Summary of DOM Navigation Properties

Property	Description	Includes Text Nodes?
parentNode	Returns parent node	No
childNodes	All child nodes	Yes
firstChild/lastChild	First/last child node	Yes
nextSibling/previousSibling	Next/previous sibling node	Yes
firstElementChild/lastElementChild	First/last element only	No
nextElementSibling/previousElementSibling	Next/previous element only	No

Table 3: DOM Navigation Properties

—

6.9 Changing Content in the DOM

Key Properties

- `innerHTML`: Sets/gets HTML content.
- `textContent`: Sets/gets plain text only.
- `innerText`: Like textContent but respects CSS styling.

—

6.10 Adding and Removing Elements

```
let newP = document.createElement("p");
newP.textContent = "New paragraph";
content.appendChild(newP);

let another = document.createElement("p");
another.textContent = "Inserted before new paragraph";
```

```
content.insertBefore(another, newP);

content.removeChild(newP);

let replacement = document.createElement("p");
replacement.textContent = "Replacement!";
content.replaceChild(replacement, another);
```

—

6.11 Modifying Attributes and Styles

```
replacement.setAttribute("class", "highlight");
let value = replacement.getAttribute("class");
replacement.removeAttribute("class");

replacement.style.color = "blue";
replacement.style.fontWeight = "bold";
```

—

6.12 Summary of DOM Manipulation Techniques

Method/Property	**Purpose**	**Example**
innerHTML	Insert HTML	element.innerHTML="<p>Content</p>";
textContent	Plain text	element.textContent="Text";
appendChild	Add child	parent.appendChild(newEl);
insertBefore	Insert child	parent.insertBefore(newEl, refEl);
removeChild	Remove child	parent.removeChild(el);
replaceChild	Replace child	parent.replaceChild(newEl, oldEl);
setAttribute	Set attribute	el.setAttribute("class","highlight");
getAttribute	Get attribute	el.getAttribute("class");
removeAttribute	Remove attribute	el.removeAttribute("class");
.style	Inline style	el.style.color="blue";

Table 4: DOM Manipulation Techniques

—

6.13 Understanding Event Listeners in the DOM

Basic Syntax

```
element.addEventListener(event, function, options);
```

Example

```
let content = document.getElementById("content");
content.addEventListener("click", function() {
   alert("Content clicked!");
});
```

Removing an Event Listener

```
function handleClick() {
   alert("Content clicked!");
}
content.addEventListener("click", handleClick);
content.removeEventListener("click", handleClick);
```

—

6.14 Common Events

- Mouse: click, dblclick, mouseover, mouseout
- Keyboard: keydown, keyup
- Form: submit, focus, blur
- Window: load, resize, scroll

—

6.15 Event Propagation and Bubbling

```
<div id="parent">
<div id="child">
<button id="button">Click me</button>
</div>
</div>
```

```
document.getElementById("parent").addEventListener("click", () =>
    {
   console.log("Parent clicked!");
}, true);

document.getElementById("child").addEventListener("click", () => {
   console.log("Child clicked!");
});

document.getElementById("button").addEventListener("click", (event
    ) => {
   console.log("Button clicked!");
   event.stopPropagation();
});
```

—

6.16 Best Practices for DOM Traversal

- Cache elements instead of re-querying.
- Minimize reflows and repaints.
- Batch DOM updates with `DocumentFragment`.

Concept	**Description**	**Example**
Event Listener	Attach/remove functions	addEventListener("click", fn)
Propagation	Capturing/bubbling control	event.stopPropagation()
Caching	Store queried element	let el = getElementById("id");
Reflows/Repaints	Reduce layout updates	el.classList.add("style");
Batching	Use DocumentFragment	document.createDocumentFragment();

Table 5: Best Practices for DOM Manipulation

—

6.17 Lab Task

1. Build a calculator where:
   - User inputs first number, operator, and second number.
   - A Calculate button shows result.
2. Take an input number and display its multiplication table using JavaScript.

7 Lab 07: jQuery

- jQuery is a fast, small, cross-platform, and feature-rich JavaScript library.
- It is designed to simplify client-side scripting of HTML.
- It simplifies DOM manipulation, event handling, animations, and AJAX with an easy-to-use API.

Objectives:
- Understand the basics of jQuery
- Learn how it simplifies JavaScript coding

Learning Outcomes:
- Use jQuery selectors and events
- Perform DOM manipulation with jQuery
- Apply effects and animations
- Build an FAQ section with `.slideToggle()`

Key Competencies:
- DOM manipulation
- Event handling
- jQuery animations

7.1 Introduction to jQuery

Objective: Understand the basics of jQuery and how it simplifies JavaScript coding.

Topics:

- **What is jQuery?** A lightweight, “write less, do more” JavaScript library. Simplifies DOM manipulation, events, animations, and AJAX.
- **Downloading vs. CDN**
  - Downloading: Allows offline usage.
  - CDN: Faster, reduces server load (Google, Microsoft, Cloudflare).
- **Syntax Overview** Basic syntax: `$(selector).action();` Example: `$("p").hide();` hides all <p> elements.

Task:

1. Create a simple HTML file.
2. Include jQuery via CDN:

Listing 2: Including jQuery via CDN

```
      <script src="https://code.jquery.com/jquery-3.6.0.min.js"
          ></script>
```

3. Add script to display alert:

Listing 3: Testing jQuery

```
      alert("jQuery is working!");
```

7.2 Basic Selectors and Events

Objective: Learn how to select elements and handle events.

Selectors:

- `$()` → Selects HTML elements
- `#id` → Select by ID
- `.class` → Select by class
- Pseudo-selectors for advanced selection

Event Handling: Common events: `click`, `dblclick`, `hover`, `focus`, `blur`.

Listing 4: Simple click event

```
$("#myButton").click(function() {
   alert("Button clicked!");
});
```

Task: To-Do List

1. Create `<ul id="todoList"></ul>` and input box.
2. Add items with `.click()`.
3. Toggle completion with `.toggleClass("completed")`.

Listing 5: To-Do List Application

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>jQuery To-Do List</title>
<style>.completed { text-decoration: line-through; }</style>
</head>
<body>
<input type="text" id="newItem" placeholder="Add a new task">
<button id="addItem">Add Task</button>
<ul id="todoList"></ul>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script
    >
<script>
$(document).ready(function() {
   $("#addItem").click(function() {
      const task = $("#newItem").val();
      if (task) {
         $("#todoList").append("<li>" + task + "</li>");
         $("#newItem").val("");
      }
   });
   $(document).on("click", "#todoList li", function() {
      $(this).toggleClass("completed");
   });
});
</script>
</body>
```

```
</html>
```

7.2.1 Adding, Removing, and Modifying Elements

Adding Elements

```
// Append
$("#container").append("<p>Appended Item</p>");

// Prepend
$("#container").prepend("<p>Prepended Item</p>");

// After
$("#container").after("<p>Added after</p>");

// Before
$("#container").before("<p>Added before</p>");
```

Removing Elements

```
$("#container").remove();
$("#container").empty();
```

7.2.2 Modifying Content and Attributes

```
// HTML and Text
$("#container").html("<p>New HTML</p>");
let html = $("#container").html();

$("#container").text("New Text");
let text = $("#container").text();

// Attributes
$("#container").attr("data-type","info");
let val = $("#container").attr("data-type");
$("#container").removeAttr("data-type");
```

7.3 Mini Project: Notes App

Objective: Use `.append()`, `.html()`, and `.remove()` to create a notes app.

Listing 6: Notes App

```
<!DOCTYPE html>
<html>
<head>
<title>Notes App</title>
</head>
<body>
<h2>Notes Application</h2>
<input type="text" id="noteInput">
<button id="addNoteButton">Add Note</button>
<div id="notesContainer"></div>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script
    >
```

```
<script>
$(document).ready(function() {
   $("#addNoteButton").click(function() {
      let note = $("#noteInput").val();
      if (note) {
         $("#notesContainer").append(
         "<p>" + note +
         " <button class='deleteBtn'>Delete</button>" +
         " <button class='editBtn'>Edit</button></p>"
         );
         $("#noteInput").val("");
      }
   });
   $(document).on("click",".deleteBtn",function() {
      $(this).parent().remove();
   });
   $(document).on("click",".editBtn",function() {
      let newText = prompt("Edit:", $(this).parent().text().
          replace("Delete Edit",""));
      if (newText) {
         $(this).parent().html(newText +
         " <button class='deleteBtn'>Delete</button>" +
         " <button class='editBtn'>Edit</button>");
      }
   });
});
</script>
</body>
</html>
```

7.4 Effects and Animations

7.4.1 Fading

```
$("#fadeInButton").click(() => $("#fadeBox").fadeIn());
$("#fadeOutButton").click(() => $("#fadeBox").fadeOut());
$("#fadeToggleButton").click(() => $("#fadeBox").fadeToggle());
```

7.4.2 Sliding

```
$("#slideUpButton").click(() => $("#slideBox").slideUp());
$("#slideDownButton").click(() => $("#slideBox").slideDown());
$("#slideToggleButton").click(() => $("#slideBox").slideToggle());
```

7.4.3 Custom Animation

```
$("#animateButton").click(function() {
   $("#animateBox").animate({
      width: "200px",
      height: "200px",
      opacity: 0.5
   }, 1000);
});
```

7.5 FAQ with .slideToggle()

Objective: Build FAQ with toggle answers.

Listing 7: FAQ Section

```
<!DOCTYPE html>
<html>
<head>
<title>FAQ</title>
<style>
.question { cursor:pointer; font-weight:bold; }
.answer { display:none; margin-left:20px; }
</style>
</head>
<body>
<p class="question">What is jQuery?</p>
<p class="answer">A fast, feature-rich library.</p>

<p class="question">How do I include it?</p>
<p class="answer">Use CDN or download.</p>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script
    >
<script>
$(document).ready(function() {
   $(".question").click(function() {
      $(this).next(".answer").slideToggle();
   });
});
</script>
</body>
</html>
```

7.6 Traversing

Objective: Understand how to navigate the DOM using jQuery to find elements based on relationships and filter specific elements.

7.6.1 Traversing the DOM

Listing 8: .parent() Example

```
<div class="container">
<p class="child">This is a paragraph.</p>
</div>
<button id="parentButton">Find Parent</button>
<script>
$("#parentButton").click(function() {
   $(".child").parent().css("border", "2px solid blue");
});
</script>
```

Explanation: Finds the direct parent of `.child` and applies a blue border.

Listing 9: .children() Example

```
<ul class="list">
<li>Item 1</li>
<li>Item 2</li>
</ul>
<button id="childrenButton">Find Children</button>
<script>
$("#childrenButton").click(function() {
   $(".list").children().css("color", "green");
});
</script>
```

Explanation: Finds all `li` elements inside `.list` and makes them green.

Listing 10: .siblings() Example

```
<p class="item">Item A</p>
<p class="item">Item B</p>
<p class="item selected">Item C</p>
<p class="item">Item D</p>
<button id="siblingsButton">Find Siblings</button>
<script>
$("#siblingsButton").click(function() {
   $(".selected").siblings().css("opacity", "0.5");
});
</script>
```

Explanation: Selects all siblings of `.selected` and changes their opacity.

Listing 11: .find() Example

```
<div class="container">
<p>Paragraph 1</p>
<p class="highlight">Paragraph 2</p>
</div>
<button id="findButton">Find Element</button>
<script>
$("#findButton").click(function() {
   $(".container").find(".highlight").css("background", "yellow");
});
</script>
```

Explanation: Finds `.highlight` inside `.container` and highlights it.

7.6.2 Filtering Elements

Listing 12: .filter() Example

```
<div class="items">
<p class="item">Item 1</p>
<p class="item highlight">Item 2</p>
<p class="item">Item 3</p>
</div>
<button id="filterButton">Filter Highlighted</button>
```

```
<script>
$("#filterButton").click(function() {
   $(".item").filter(".highlight").css("font-weight", "bold");
});
</script>
```

Explanation: Selects only items with `.highlight` and makes them bold.

Listing 13: .not() Example

```
<ul class="list">
<li class="item">Item 1</li>
<li class="item ignore">Item 2</li>
<li class="item">Item 3</li>
</ul>
<button id="notButton">Filter Exclude</button>
<script>
$("#notButton").click(function() {
   $(".item").not(".ignore").css("text-decoration", "underline");
});
</script>
```

Explanation: Selects `.item` elements except those with `.ignore`.

Listing 14: .first() Example

```
<div class="items">
<p class="item">Item 1</p>
<p class="item">Item 2</p>
<p class="item">Item 3</p>
</div>
<button id="firstButton">Select First</button>
<script>
$("#firstButton").click(function() {
   $(".item").first().css("background-color", "lightblue");
});
</script>
```

Explanation: Selects the first `.item` and sets its background to light blue.

Listing 15: .last() Example

```
<button id="lastButton">Select Last</button>
<script>
$("#lastButton").click(function() {
   $(".item").last().css("background-color", "lightcoral");
});
</script>
```

Explanation: Selects the last `.item` and sets its background to light coral.

7.7 Task: Create a Menu with Sections

Objective: Build a menu that allows users to navigate sections of a page. Clicking a menu item highlights the target section and hides its siblings.

Listing 16: Menu Navigation with jQuery

```
      <ul>
      <li class="menu" data-target="#section1">Section 1</li>
      <li class="menu" data-target="#section2">Section 2</li>
      <li class="menu" data-target="#section3">Section 3</li>
      </ul>

      <div id="section1" class="section">Content for Section 1</div>
      <div id="section2" class="section">Content for Section 2</div>
      <div id="section3" class="section">Content for Section 3</div>

      <script>
      $(document).ready(function() {
         $(".menu").click(function() {
            let target = $(this).data("target");
            $(target).addClass("visible")
            .siblings(".section").removeClass("visible");
         });
      });
      </script>
```

7.8 Explanation of the Task Code

- When a menu item is clicked, the `data-target` attribute identifies the target section.
- The `.siblings()` method hides all other sections.
- Visibility is controlled using `.addClass("visible")` and `.removeClass("visible")`.

8 Lab 08: React App Installation, Functional Component, Class, Props, Props Children and Hooks

Objectives: Understand the basics of creating React apps, functional and class components, props (including props.children), and how to use React Hooks (`useState`, `useEffect`, `useRef`).

Learning Outcomes: - Install and set up a React app. - Differentiate between functional and class components. - Use props and props.children effectively. - Apply React Hooks (`useState`, `useEffect`, `useRef`) in practical tasks.

Key Competencies: React setup, component structure, props handling, hooks, and state management.

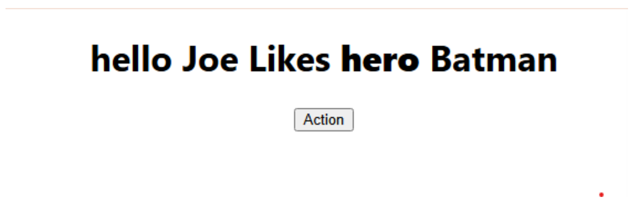
React is a popular JavaScript library for building user interfaces. In this lab, we will explore component creation, props, and hooks.

8.1 React App Installation and Components

Listing 17: React Functional and Class Components

```
import React, { Component } from 'react'

// Functional Component
function Welcome(){
   return <h1>Hello World</h1>
}
export default Welcome

// Arrow Function Component
export const Welcome = () => <h1>My first Program</h1>

// Class Component
class Welcome extends Component{
   render(){
      return <h1>My first class</h1>
   }
}
export default Welcome

// Props with Children
const Welcome = props => {
   console.log(props)
   return (
   <div>
   <h1>Hello {props.name} Likes <b>hero</b> {props.heroName}</h1>
   {props.children}
   </div>
   )
}
export default Welcome
```

Output:


hello Joe Likes hero Batman

8.2 Lab Task

You are asked to build a simple React component named `StudentIntro` that introduces a student by name and favorite subject. Additional content (like a note or hobby) should be passed using `props.children`. Try implementing it both as a functional component and as a class.

8.2.1 Sub Tasks

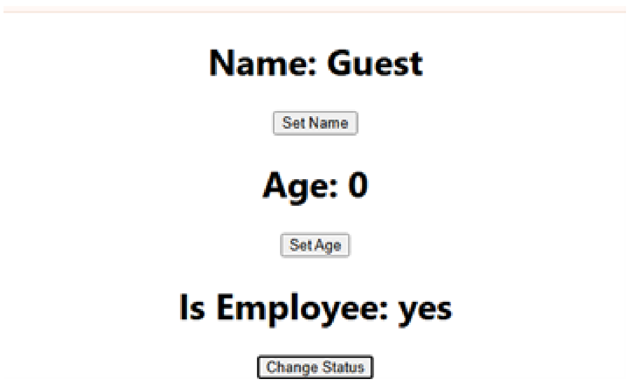
Objective: Understand how to use React Hooks for state management and interactivity.

UseState Hook

Listing 18: Using useState Hook

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

function Welcome(){
   const [name, setName] = useState("Guest");
   const [age, setAge] = useState(0);
   const [isEmployee, setStatus] = useState(true);

   const updateName = () => { setName("Mehwish"); }
   const updateAge = () => { setAge(age + 1); }
   const employeeStatus = () => { setStatus(!isEmployee); }

   return (
   <div>
   <h1>Name: {name}</h1>
   <button onClick={updateName}>Set Name</button>

   <h1>Age: {age}</h1>
   <button onClick={updateAge}>Set Age</button>

   <h1>Is Employee: {isEmployee ? "Yes" : "No"}</h1>
   <button onClick={employeeStatus}>Change Status</button>
   </div>
   )
}
export default Welcome;
```

Output:



Name: Guest

Age: 0

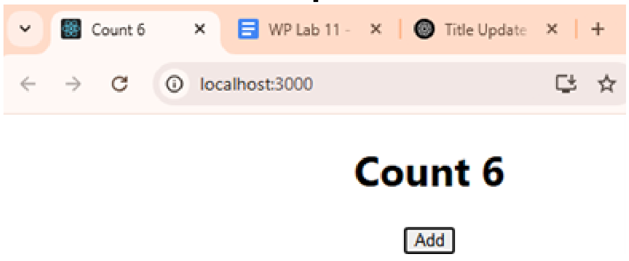
Is Employee: yes

UseEffect Hook

Listing 19: Using useEffect Hook

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

function Welcome(){
   const [count, setCount] = useState(0);

   useEffect(() => {
      document.title = 'Count ${count}';
   }, [count])

   const addCount = () => { setCount(count + 1); }

   return (
   <div>
   <h1>Count {count}</h1>
   <button onClick={addCount}>Add</button>
   </div>
   )
}
export default Welcome;
```

Output:



Count 6

Listing 20: Using useRef Hook

UseRef Hook

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

function Welcome(){
   const [count, setCount] = useState(0);
   const [name, setName] = useState("Mehwish");
   const refElement = useRef("");

   function increaseCount(){ setCount(count + 1); }
   function change(){
      setName(" ");
      refElement.current.focus();
```

```
    }

    useEffect(() => {
       document.title = ‘Count ${count}‘;
    }, [count])

    return (
    <div>
    <h1>Count : {count}</h1>
    <button onClick={increaseCount}>Click</button>

    <input ref={refElement} type="text"
    value={name}
    onChange={(e) => setName(e.target.value)} />

    <button onClick={change}>Change</button>
    </div>
    )
 }
 export default Welcome;
```

8.3 Final Lab Task

- Implement increment, decrement, and reset functions in a React app using `useState`.
- Create a function `windowResize` that displays the current size of the window (hint: use `window.addEventListener("resize", windowResize)`).
- Add three input fields (`First Name`, `Last Name`, `ID`) and change the text color dynamically using `useRef`.

9 Lab 09: Calling API using Fetch Method, Axios Post and Get method

Objectives: Understand how to call APIs in React applications using the `fetch()` method and Axios library with GET/POST requests.

React enables seamless integration with external APIs. In this lab, we will learn two approaches for making API calls: the native `fetch()` method and Axios.

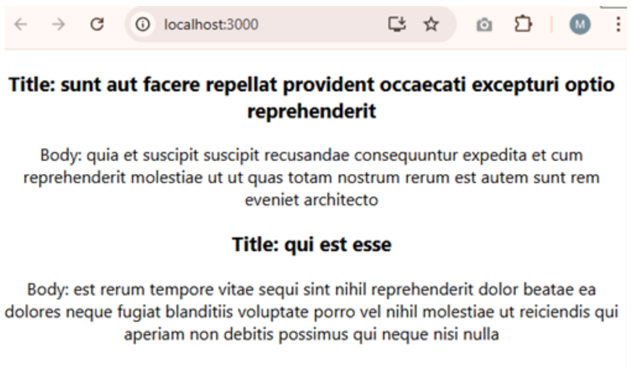
9.1 Fetch Method, Axios Post and Get Method

9.1.1 API Calling (Fetch Method)

Listing 21: API Calling with Fetch Method

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

const FetchApi = () => {
   const [data, setData] = useState([])

   const getData = async () => {
      try {
         const response = await fetch("https://jsonplaceholder.
             typicode.com/posts")
         const data = await response.json()
         console.log(data)
         setData(data)
      } catch (err) {
         console.log(err)
      }
   }

   useEffect(() => { getData() }, [])

   return (
   <div>
   {data ?
      data.map((curVal) => {
         console.log(curVal)
         return (
         <div>
         <h3>Title: {curVal.title}</h3>
         <p>Body: {curVal.body}</p>
         </div>
         )
      })
      : <p>Data not found</p>}
   </div>
   )
}

export default FetchApi
```

Output:



localhost:3000

Title: sunt aut facere repellat provident occaecati excepturi optio reprehenderit

Body: quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

Title: qui est esse

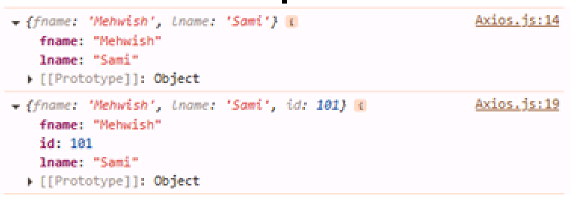
Body: est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

9.1.2 Axios Get/Post Method

Listing 22: Axios API Calls

```
import React, { useState } from 'react'
import axios from 'axios'

const Axios = () => {
   const data = { fname: "", lname: "" }
   const [inputData, setInputData] = useState(data)

   const handleInput = (event) => {
      console.log(event.target.value)
      setInputData({ ...inputData, [event.target.name]: event.
          target.value })
   }

   const handleForm = (event) => {
      event.preventDefault()
      axios.post("https://jsonplaceholder.typicode.com/posts",
          inputData)
      .then((res) => { console.log(res.data) })
      .catch((err) => { console.log(err) })
   }

   return (
   <div>
   <form onSubmit={handleForm}>
   <input type="text" placeholder="Enter name" name="fname"
       onChange={handleInput}/>
   <input type="text" placeholder="Enter last name" name="lname"
       onChange={handleInput}/>
   <button>Submit</button>
   </form>
   </div>
   )
}

export default Axios
```

Output:



```
▾ {fname: 'Mehwish', lname: 'Sami'}                     Axios.js:14
    fname: "Mehwish"
    lname: "Sami"
  ▸ [[Prototype]]: Object
▾ {fname: 'Mehwish', lname: 'Sami', id: 101}            Axios.js:19
    fname: "Mehwish"
    id: 101
    lname: "Sami"
  ▸ [[Prototype]]: Object
```

9.2 Lab Task

1. Design a React App and call an API using the **fetch method**, then display the result on the web page in **card form**.
2. Design a React App and use **Axios POST** to submit four input fields: `Sapid`, `FName`, `LName`, and `Department`.

10 Lab 10: FrontEnd and BackEnd Combined working with the help of Node.js and Express

Objectives: Integrate frontend (React) and back-end (Node.js + Express) to perform client–server communication.

In this lab, we will connect a React-based frontend with an Express backend using query strings for data exchange.

10.1 Node.js and Express

10.1.1 Index.js (Backend with Express)

Listing 23: Index.js - Backend with Express

```
const express = require('express');
const app = express();
const cors = require('cors');
app.use(cors());

app.get('/add', (req, res) => {
   const num1 = parseFloat(req.query.num1);
   const num2 = parseFloat(req.query.num2);

   if (!isNaN(num1) && !isNaN(num2)) {
      const sum = num1 + num2;
      res.json({ result: sum });
   } else {
      res.status(400).json({ error: 'invalid number' });
   }
});

const port = 3000;
app.listen(port, () => {
   console.log(`server is running on http://localhost:${port}`);
});
```

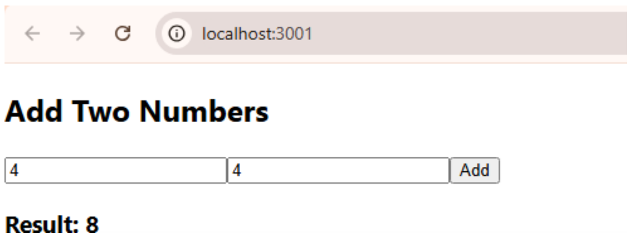
10.1.2 App.js (Frontend with React)

Listing 24: App.js - React Frontend

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

const App = () => {
   const [num1, setNum1] = useState('');
   const [num2, setNum2] = useState('');
   const [result, setResult] = useState(null);
   const [error, setError] = useState('');

   const handleAdd = async () => {
      try {
         const response = await fetch(
         `http://localhost:3000/add?num1=${num1}&num2=${num2}`
         );
         const data = await response.json();

```

```
            if (response.ok) {
               setResult(data.result);
               setError('');
            } else {
               setError(data.error || 'error');
               setResult(null);
            }
         } catch (err) {
            setError('failed to fetch');
         }
      };

      return (
      <div>
      <h2>Add Two Numbers</h2>
      <input type="number" value={num1}
      onChange={(e) => setNum1(e.target.value)}
      placeholder="first num"/>
      <input type="number" value={num2}
      onChange={(e) => setNum2(e.target.value)}
      placeholder="second num"/>
      <button onClick={handleAdd}> Add</button>
      {result !== null && <h3>Result: {result}</h3>}
      </div>
      );
   }

   export default App
```

10.1.3 Output



Add Two Numbers

Result: 8

10.2 Lab Task

1. Design a **Student Info Form** with input fields: `Sapid`, `FirstName`, `LastName`, `Department`, and `Age`.
2. On clicking the **Submit** button, the input data should be displayed on the web page using:
   - Node.js
   - Express
   - Query String for passing data

11 Lab 11: Understanding and Using Props in React

11.1 React.js Essentials: Props, Events, Hooks, and State Management

Objective: Learn how to pass data between components using props.

Theory: Props (short for properties) are used to pass data from one component to another in a unidirectional flow (parent → child).

Instructions:

1. Create a new React App:

```
npx create-react-app props-demo
cd props-demo
npm start
```

2. Create two components: `App.js` and `Message.js`.

11.1.1 Code:

Listing 25: Message.js

```
function Message({ text }) {
   return <h1>{text}</h1>;
}
export default Message;
```

Listing 26: App.js

```
import Message from './Message';

function App() {
   return (
   <div>
   <Message text="Hello from Props!" />
   </div>
   );
}
export default App;
```

Expected Output: The browser should display `Hello from Props!`.

11.2 Lab 1: React Event Handling

Objective: Learn to handle user events like clicks and form submissions.

11.2.1 Code:

Theory: React handles events using camelCase syntax and passes functions as event handlers.

Listing 27: ClickEvent.js

```
function ClickEvent() {
   const handleClick = () => {
      alert("Button clicked!");
   };
   return <button onClick={handleClick}>Click Me</button>;
}
export default ClickEvent;
```

Listing 28: App.js

```
import ClickEvent from './ClickEvent';

function App() {
   return (
   <div>
   <ClickEvent />
   </div>
   );
}
export default App;
```

Expected Output: Clicking the button triggers an alert: `Button clicked!`.

11.3 Lab 2: Using React Hooks

Objective: Understand and use `useState` and `useEffect`.

Theory: Hooks are functions that let you “hook into” React features. `useState` is for stateful values. `useEffect` is for side effects like API calls.

11.3.1 Code:

Listing 29: Counter.js

```
import { useState } from 'react';

function Counter() {
   const [count, setCount] = useState(0);

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

Listing 30: App.js

```
import Counter from './Counter';

function App() {
   return (
   <div>
   <Counter />
   </div>
   );
}
export default App;
```

Expected Output: A button that increments the counter on click.

11.4 Lab 3: State Management Techniques in React

Objective: Explore various state management techniques: Local state (`useState`), Shared state (lifting state up), and Global state using Context API.

11.4.1 Part A: Lifting State Up

Listing 31: Parent.js

```
import { useState } from 'react';
import Child from './Child';

function Parent() {
   const [name, setName] = useState('');
   return (
   <div>
   <Child name={name} setName={setName} />
   <p>Typed Name: {name}</p>
   </div>
   );
}
export default Parent;
```

Listing 32: Child.js

```
function Child({ name, setName }) {
   return (
   <input type="text" value={name}
   onChange={(e) => setName(e.target.value)} />
   );
}
export default Child;
```

11.4.2 Part B: Global State using Context API

Listing 33: Context.js

```
import { createContext, useState } from 'react';

export const AppContext = createContext();

export function AppProvider({ children }) {
   const [user, setUser] = useState('Alice');
   return (
   <AppContext.Provider value={{ user, setUser }}>
   {children}
   </AppContext.Provider>
   );
}
```

Listing 34: Profile.js

```
import { useContext } from 'react';
import { AppContext } from './Context';

function Profile() {
   const { user } = useContext(AppContext);
   return <h1>User: {user}</h1>;
}
export default Profile;
```

Listing 35: App.js

```
import { AppProvider } from './Context';
import Profile from './Profile';

function App() {
   return (
   <AppProvider>
   <Profile />
   </AppProvider>
   );
}
export default App;
```

11.5 Conclusion

This lab manual introduced:

- Props for component communication
- Event handling for user interactions
- React Hooks (`useState`, `useEffect`) for functional state and side effects
- State management using local, shared, and global techniques

12 Lab 12: Creating a Basic HTTP Server and Introduction to Node.js and Modules

12.1 Server Side Programming with Node.js

Objective: Build a simple HTTP server using Node.js.

Theory: Node.js allows us to build scalable network applications using JavaScript on the server side. The built-in `http` module enables handling of HTTP requests and responses.

Instructions:

1. Install Node.js from `https://nodejs.org/`.
2. Create a new folder and navigate into it via terminal.
3. Create a file named `server.js`.

Listing 36: server.js

```
const http = require('http');

const server = http.createServer((req, res) => {
   res.statusCode = 200;
   res.setHeader('Content-Type', 'text/plain');
   res.end('Hello, World!\n');
});

server.listen(3000, () => {
   console.log('Server running at http://localhost:3000/');
});
```

Run with: node `server.js` Open browser at: `http://localhost:3000/`

Expected Output: A web page displaying `Hello, World!`.

12.1.1 Exploring the Request/Response Cycle

Objective: Understand how the request and response objects work in Node.js.

Theory: Every HTTP interaction involves a client request and a server response. Node.js provides objects `req` and `res` to handle these.

Listing 37: Modified server.js

```
const http = require('http');

const server = http.createServer((req, res) => {
   res.statusCode = 200;
   res.setHeader('Content-Type', 'text/plain');
   res.end(`Requested URL: ${req.url}\nMethod: ${req.method}`);
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

Expected Output: Displays the requested URL and HTTP method.

12.1.2 Route Handling in Node.js

Objective: Learn how to handle multiple routes in a Node.js server.

Theory: By checking `req.url`, different responses can be served for different routes.

Listing 38: Route handling in Node.js

```
const http = require('http');

const server = http.createServer((req, res) => {
   res.setHeader('Content-Type', 'text/plain');
   if (req.url === '/') {
      res.end('Home Page');
   } else if (req.url === '/about') {
      res.end('About Page');
   } else {
      res.statusCode = 404;
      res.end('Page Not Found');
   }
});

server.listen(3000, () => {
   console.log('Server running at http://localhost:3000/');
});
```

Expected Output: Displays different responses for '/', '/about', and others.

12.1.3 Conclusion

This section covered:

- Building a basic HTTP server
- Understanding the Request/Response cycle
- Handling multiple routes

12.2 Module: Node.js Programming: File System & File Upload Handling

Objective: Understand Node.js fundamentals and use built-in modules.

Theory: Node.js modules such as `fs` and `http` are essential for server-side tasks.

Listing 39: app.js

```
console.log("Node.js is working!");
```

Expected Output: Terminal prints `Node.js is working!`.

12.2.1 Working with File System Module

Objective: Learn to read, write, and append files using the `fs` module.

Theory: The `fs` module enables file operations like reading, writing, updating, and deleting.

Listing 40: fileOps.js

```
const fs = require('fs');

// Write to a file
fs.writeFileSync('example.txt', 'This is the initial content.');

// Read file content
const data = fs.readFileSync('example.txt', 'utf8');
console.log('File content:', data);

// Append to the file
fs.appendFileSync('example.txt', '\nAppended content.');

console.log('File updated.');
```

Expected Output: Console logs file content and updates.

12.3 Handling File Uploads in Node.js

Objective: Implement file upload functionality using the `formidable` module.

Theory: File uploads can be processed by third-party modules like `formidable`.

HTML Form:

Listing 41: upload.html

```html
<html>
<body>
<form action="upload" method="post" enctype="multipart/form-data">
<input type="file" name="fileToUpload"><br>
<input type="submit">
</form>
</body>
</html>
```

Node.js Upload Server:

Listing 42: uploadServer.js

```
const http = require('http');
const fs = require('fs');
const formidable = require('formidable');
```

```

http.createServer((req, res) => {
   if (req.url === '/upload' && req.method.toLowerCase() === 'post
      ') {
      const form = new formidable.IncomingForm();
      form.parse(req, (err, fields, files) => {
         const oldPath = files.fileToUpload[0].filepath;
         const newPath = './uploads/' + files.fileToUpload[0].
             originalFilename;
         fs.rename(oldPath, newPath, (err) => {
            if (err) throw err;
            res.write('File uploaded and moved!');
            res.end();
         });
      });
   } else {
      fs.readFile('upload.html', (err, data) => {
         res.writeHead(200, { 'Content-Type': 'text/html' });
         res.write(data);
         return res.end();
      });
   }
}).listen(8080);
```

Run with: node `uploadServer.js` and open `http://localhost:8080/`.

Expected Output: Uploaded file appears in the `uploads` folder.

12.3.1 Conclusion

This section covered:

- Writing basic Node.js scripts
- Using the File System module
- Uploading files with `formidable`

13 Lab 13: Creating a Basic Express Server

13.1 Module: Server-Side Programming with Express.js

Objective: Set up a simple Express.js server and handle basic routing.

Theory: Express.js is a web framework for Node.js that simplifies routing and middleware integration for building scalable web applications.

Instructions:

1. Create a new folder and initialize it: `npm init -y`
2. Install Express: `npm install express`
3. Create a file `server.js` and add the following code:

Listing 43: server.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
   res.send('Hello, Express!');
});

app.listen(3000, () => {
   console.log('Server running at http://localhost:3000');
});
```

Run with: `node server.js` and visit `http://localhost:3000/`

Expected Output: `Hello, Express!` displayed in the browser.

13.2 Middleware in Express.js

Objective: Learn how to use middleware functions to process requests.

Theory: Middleware functions in Express execute during the request-response cycle. They have access to `req`, `res`, and `next`.

Instructions:

1. In your `server.js`, add a middleware:

Listing 44: Adding middleware in Express

```
app.use((req, res, next) => {
   console.log(`${req.method} ${req.url}`);
   next();
});
```

Make requests to / or other routes and observe logs.

Expected Output: Logs like GET / in the terminal.

13.3 Lab 3: Handling GET and POST Requests

Objective: Learn to handle GET and POST requests and process data.

Theory: Express routes respond to HTTP methods like GET and POST. POST requests usually include data in the request body.

Instructions:

1. Install body-parser: `npm install body-parser`
2. Update `server.js`:

Listing 45: GET and POST in Express

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.get('/greet', (req, res) => {
   res.send(`Hello, ${req.query.name || 'Guest'}!`);
});

app.post('/submit', (req, res) => {
   res.send(`Received data: ${JSON.stringify(req.body)}`);
});
```

Test With: - Browser: `http://localhost:3000/greet?name=Alice` - Postman for POST requests to `/submit`

Expected Output: - GET: `Hello, Alice!` - POST: Echoed JSON data

13.4 Implementing a RESTful Web Service

Objective: Build a RESTful API using Express.js.

Theory: RESTful APIs follow a stateless, uniform interface using standard HTTP verbs: GET, POST, PUT, DELETE.

Instructions: Extend `server.js` with:

Listing 46: RESTful API in Express

```
let items = [];

app.get('/api/items', (req, res) => {
   res.json(items);
});

app.post('/api/items', (req, res) => {
   items.push(req.body);
   res.status(201).json({ message: 'Item added' });
});

app.delete('/api/items/:id', (req, res) => {
   const id = parseInt(req.params.id);
   items = items.filter((_, index) => index !== id);
   res.json({ message: 'Item deleted' });
});
```

Test With: - GET `/api/items` - POST `/api/items` with JSON body - DELETE `/api/items/:id`

Expected Output: JSON data and success messages.

13.4.1 Conclusion

This lab manual provided hands-on experience with:

- Setting up a basic Express.js server
- Integrating middleware for request processing
- Handling GET and POST requests
- Creating a simple RESTful web service

14 Lab 14: Working with Cookies in Express.js and Connecting to MongoDB using Mongoose

14.1 Module: Cookies, Sessions, and Authentication with Express.js

Objective: Understand how to set, read, and clear cookies in Express.js.

Theory: Cookies are small pieces of data stored on the client-side to maintain stateful information across sessions.

Instructions:

1. Install the required package: `npm install express cookie-parser`
2. Create a file `cookies.js`:

Listing 47: cookies.js

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();

app.use(cookieParser());

app.get('/setcookie', (req, res) => {
   res.cookie('username', 'john', { maxAge: 900000 });
   res.send('Cookie set');
});

app.get('/getcookie', (req, res) => {
   res.send(`Cookie value: ${req.cookies.username}`);
});

app.get('/clearcookie', (req, res) => {
   res.clearCookie('username');
   res.send('Cookie cleared');
});

app.listen(3000, () => console.log('Server started on http://
    localhost:3000'));
```

Expected Output: Cookie is set, retrieved, and cleared successfully.

14.2 Lab 2: Managing Sessions with Express.js

Objective: Learn to manage server-side sessions using `express-session`.

Theory: Sessions are server-side storage solutions used to track user data between requests securely.

Listing 48: session.js

```
const express = require('express');
const session = require('express-session');
const app = express();

app.use(session({
   secret: 'mySecret',
   resave: false,
   saveUninitialized: true
}));

app.get('/', (req, res) => {
   if (req.session.views) {
      req.session.views++;
      res.send(`Welcome back! Views: ${req.session.views}`);
   } else {
      req.session.views = 1;
      res.send('Welcome! This is your first visit.');
   }
});

app.listen(3000, () => console.log('Session app on http://
    localhost:3000'));
```

Expected Output: User session is maintained and visit count increases on each refresh.

14.3 Lab 3: Implementing Basic Authentication with Express.js

Objective: Implement a basic username/password login system using Express.js and sessions.

Theory: Authentication verifies user identity. A common method is username/password with sessions to persist the login.

Listing 49: auth.js

```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.urlencoded({ extended: false }));
app.use(session({
   secret: 'secretKey',
   resave: false,
   saveUninitialized: true
}));

const USER = { username: 'admin', password: '1234' };

app.get('/', (req, res) => {
   if (req.session.user) {
      res.send(`Hello ${req.session.user}. <a href="/logout">
          Logout</a>`);
   } else {
      res.sendFile(__dirname + '/login.html');
   }
});

app.post('/login', (req, res) => {
   const { username, password } = req.body;
   if (username === USER.username && password === USER.password) {
        req.session.user = username;
        res.redirect('/');
    } else {
        res.send('Invalid credentials. <a href="/">Try again</a>');
    }
});

app.get('/logout', (req, res) => {
    req.session.destroy();
    res.redirect('/');
});

app.listen(3000, () => console.log('Auth app running on http://
    localhost:3000'));
```

Listing 50: login.html

```html
<!DOCTYPE html>
<html>
<body>
<h2>Login</h2>
<form method="POST" action="/login">
Username: <input type="text" name="username"><br>
Password: <input type="password" name="password"><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

Expected Output:

- Valid users are logged in and greeted.
- Sessions are used to maintain login state.

14.3.1 Conclusion

This lab covered:

- Cookie creation, retrieval, and clearing
- Session management using `express-session`
- Basic login authentication using sessions

14.4 Module: Connectivity and CRUD Operations with MongoDB and Mongoose using Express.js

Objective: Establish a connection to a MongoDB database using Mongoose.

Theory: Mongoose is an ODM library for MongoDB. It provides schema-based data modeling for Node.js apps.

Listing 51: db.js

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/mydatabase', {
   useNewUrlParser: true,
   useUnifiedTopology: true
}).then(() => console.log('MongoDB Connected'))
.catch(err => console.log(err));
```

Expected Output: `MongoDB Connected` message in the console.

14.4.1 Creating a Mongoose Model

Objective: Define a data model using Mongoose Schema.

Listing 52: userModel.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
   name: String,
   email: String,
   age: Number
});

module.exports = mongoose.model('User', userSchema);
```

Expected Output: User model is exported and ready for use in routes.

14.4.2 Implementing CRUD Operations

Objective: Perform Create, Read, Update, and Delete operations on MongoDB using Express.js and Mongoose.

Listing 53: server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const User = require('./userModel');

mongoose.connect('mongodb://localhost:27017/mydatabase', {
   useNewUrlParser: true,
   useUnifiedTopology: true
});

const app = express();
app.use(bodyParser.json());

// Create
app.post('/users', async (req, res) => {
   const user = new User(req.body);
   await user.save();
   res.send(user);
});

// Read All
app.get('/users', async (req, res) => {
   const users = await User.find();
   res.send(users);
});

// Update
```

```
app.put('/users/:id', async (req, res) => {
   const user = await User.findByIdAndUpdate(req.params.id, req.
       body, { new: true });
   res.send(user);
});

// Delete
app.delete('/users/:id', async (req, res) => {
   await User.findByIdAndDelete(req.params.id);
   res.send({ message: 'User deleted' });
});

app.listen(3000, () => console.log('Server started at http://
    localhost:3000'));
```

Expected Output: CRUD operations performed via Postman update MongoDB data.

14.4.3 Conclusion

This lab module taught:

- Connecting Express.js to MongoDB using Mongoose
- Defining and using Mongoose schemas and models
- Implementing CRUD functionality with Express routes

15 Lab 15: Setting Up PUG in Express.js and Password Hashing using bcrypt

15.1 Module: Template Engines and PUG with Express.js

Objective: Set up the PUG template engine with an Express.js application.

Theory: PUG is a template engine for Node.js used to write cleaner HTML using indentation and shorthand syntax.

Instructions:

1. Install required packages: `npm install express pug`
2. Create `app.js`:

Listing 54: app.js

```
const express = require('express');
const app = express();

app.set('view engine', 'pug');
app.set('views', './views');

app.get('/', (req, res) => {
   res.render('index', { title: 'Welcome', message: 'Hello from
       PUG!' });
});

app.listen(3000, () => console.log('Server is running at http://
    localhost:3000'));
```

Listing 55: index.pug

```
doctype html
html
head
title= title
body
h1= message
```

Expected Output: Visit `http://localhost:3000` to see a page rendered by PUG.

15.2 Using Variables and Conditionals in PUG

Objective: Learn to use variables and conditionals in PUG templates.

Listing 56: app.js update

```
app.get('/user', (req, res) => {
   res.render('user', { name: 'Alice', loggedIn: true });
});
```

Listing 57: user.pug

```
doctype html
html
head
title User Page
body
if loggedIn
h1 Welcome, #{name}!
else
h1 Please log in.
```

Expected Output: Visiting `/user` renders a personalized welcome message if loggedIn is true.

15.3 Iteration in PUG Templates

Objective: Render lists using iteration in PUG.

Listing 58: app.js update

```
app.get('/items', (req, res) => {
   res.render('items', { items: ['Pen', 'Notebook', 'Eraser'] });
});
```

Listing 59: items.pug

```
doctype html
html
head
title Items List
body
h1 Items Available:
ul
each item in items
li= item
```

Expected Output: A list of items is rendered dynamically.

15.3.1 Conclusion

This module introduced:

- Integration of PUG with Express.js
- Using variables and conditionals
- Iteration in templates for dynamic content

15.4 Password Hashing using bcrypt

15.4.1 Module: Authentication using bcrypt and JWT in Express.js

Objective: Learn to hash and verify passwords using bcrypt in an Express.js application.

Theory: bcrypt hashes passwords securely with salt, protecting against rainbow table attacks.

Listing 60: hash.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

app.post('/register', async (req, res) => {
   const { password } = req.body;
   const salt = await bcrypt.genSalt(10);
   const hashedPassword = await bcrypt.hash(password, salt);
   res.send({ hashedPassword });
});

app.listen(3000, () => console.log('Server running on http://
    localhost:3000'));
```

Expected Output: Sends a hashed password when posting to `/register`.

15.4.2 User Authentication with bcrypt and JWT

Objective: Implement login with hashed password verification and JWT.

Theory: JWT (JSON Web Token) provides secure user authentication.

Listing 61: auth.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

const users = [];

app.post('/signup', async (req, res) => {
   const { username, password } = req.body;
   const hashedPassword = await bcrypt.hash(password, 10);
   users.push({ username, password: hashedPassword });
   res.send({ message: 'User registered successfully' });
});

app.post('/login', async (req, res) => {
   const { username, password } = req.body;
   const user = users.find(u => u.username === username);
   if (!user) return res.status(400).send('Invalid credentials');
```

```

  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) return res.status(400).send('Invalid credentials
      ');

  const token = jwt.sign({ username: user.username }, 'secretKey
      ', { expiresIn: '1h' });
  res.send({ token });
});

app.listen(3000, () => console.log('Auth server running on http://
     localhost:3000'));
```

Expected Output: User signs up and receives a JWT token on login.

15.5 Lab: Protecting Routes using JWT Middleware

Objective: Secure private routes with JWT verification middleware.

Listing 62: JWT Middleware

```
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.sendStatus(401);

  jwt.verify(token, 'secretKey', (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
}

app.get('/protected', authenticateToken, (req, res) => {
  res.send({ message: `Welcome ${req.user.username}, this is a
       protected route.` });
});
```

Expected Output: Accessing `/protected` requires a valid JWT in the Authorization header.

15.5.1 Conclusion

This lab demonstrated:

- Hashing passwords securely using bcrypt
- Authenticating users with JWT tokens
- Protecting routes using JWT middleware