



|| Jai Sri Gurudev ||
Sri Adichunchanagiri Shikshana Trust ®

SJB Institute of Technology

An Autonomous Institution under Visvesvaraya Technological University, Belagavi
No 67, BGS Health & Education City, Dr. Vishnuvardhan Road
Kengeri, Bangalore -560060

Department of Information Science & Engineering



React Lab

[BCSL657B]

VI SEMESTER – B.E

Academic Year: 2024-2025

(EVEN SEM)

By



FACULTY NAME:

Prof. Veeresh K M

Prof. Deeapk B N

SJB INTITUTE OF TECHNOLOGY**Institution's Vision**

To become a recognized technical education center with a global perspective.

Institution's Mission

To provide learning opportunities that foster students ethical values, intelligent development in science & technology and social responsibility so that they become sensible and contributing members of the society

Department of Information Science and Engineering**Department Vision**

We envision our department as a catalyst for developing educated, engaged and employable individuals whose collective energy will be the driving force for prosperity and quality of life in our diverse world.

Department Mission

To establish our mission is to provide quality technical education in the field of information technology and to strive for excellence in the education by developing and sharpening the intellectual and human potential for good industry and community.

RUBRICS FOR THE 2022 SCHEME

- Split-up of CIE for **record and test** are ratio **60:40** of total marks respectively.

Record		Lab Internal Assessment	
Rubrics per experiment	Marks distribution	Rubrics per experiment	Marks distribution
Observation, Write-up of Procedure/Algorithm/Program	10	Write-up of Procedure/Algorithm/Program	10
Execution of experiment	10	Execution of experiment	25
Viva-voce	10	Results & Viva-voce	15
Record Writing	20		
Total Marks	50	Total Marks	50
<ul style="list-style-type: none"> 50 marks for each experiment. Average of all experiments ---(a) 		<ul style="list-style-type: none"> 50 marks for Internal Assessment Sum/Avg is depended on Individual courses scheme 	

Laboratory courses / Laboratory with mini project / Theory integrated laboratory courses	
• Record Evaluation	• Record marks = Scaled down (a) as per individual course scheme given in the syllabus..... (b)
• Test Evaluation	<ul style="list-style-type: none"> • Number of tests= 2 tests to be conducted and compute sum or average of two tests (as given in the syllabus)..... (c) • Scale down (c) marks to maximum marks (as per individual course scheme given in the syllabus)----- (d)
Total CIE	• CIE = maximum marks= (b) + (d)

Weightage: Excellent=80-100%, Proficient=60-80%, Satisfactory=40-60%, Unsatisfactory=<40%, **Total Maximum Marks:** As prescribed by the University.

PROGRAM EDUCATIONAL OBJECTIVES(PEO'S)**Graduates will:-**

1. PEO1: Graduates will possess expertise in problem solving, design and analysis technical skills for a fruitful career accomplishing professional and social ethics with exposure to modern designing tools and technologies in information science and engineering.
2. PEO2: Graduates will excel in communication, teamwork, and multiple domains related to engineering issues accomplishing social responsibilities and management skills.
3. PEO3: Graduates will out class in competitive environment through certification courses, gaining leadership qualities and progressive research to become successful entrepreneurs.

PROGRAM SPECIFIC OUTCOMES(PSO'S)**Graduates will be able to:-**

1. **PSO1:** Graduates will be able to Analyze, apply knowledge of information science to develop software solutions in current research trends and technology.
2. **PSO2:** Graduates will be able to create social awareness and environmental wisdom along with ethical responsibility to lead a successful career and sustain passion using optimal resources to become an entrepreneur.

PROGRAM OUTCOMES-PO's

Engineering graduates will be able to:

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

General Instructions for the Laboratory***Do's***

- It is mandatory for all the students to attend all practical classes & complete the experiments as per syllabus.
- Students should strictly follow the lab timings, dress code with Apron & ID cards. Should maintain a neat observation book
- Study the theory and logic before executing the program.
- Submit the completed lab records of executed programs and update the index book in every lab session.
- Should prepare for viva questions regularly. Handle the computer systems carefully.
- Maintain discipline and silence in the lab.

Don't

- Should not take Bags and Mobile phones into the Laboratory.
- Do not wear footwear inside the Laboratory
- Systems & Components should be handled carefully failing to which penalty will be imposed.
- Do not switch off the system abruptly.
- Should not chew gum or eat in the lab.

REACT		Semester	6
Course Code	BCSL657B	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:1:0	SEE Marks	50
Credits	01	Exam Hours	100
Examination type (SEE)	Practical		

Course objectives:

- Enable students to develop React applications utilizing functional and class-based components, effectively managing state with hooks and lifecycle methods.
- Introduce, how to pass data dynamically between parent and child components using props, ensuring modular and reusable component design.
- Create dynamic and responsive applications, integrating forms, validation, task management systems, and styled components.
- Use React Router for navigation, external API integration for dynamic data handling, and CSS styling techniques for modern UI/UX design.

Sl. No	Experiments
1.	Use create-react-app to set up a new project. Edit the App.js file to include a stateful component with useState. Add an input field and a <h1> element that displays text based on the input. Dynamically update the <h1> content as the user types.
2.	Develop a React application that demonstrates the use of props to pass data from a parent component to child components. The application should include the parent component named App that serves as the central container for the application. Create two separate child components, Header: Displays the application title or heading. Footer: Displays additional information, such as copyright details or a tagline. Pass data (e.g., title, tagline, or copyright information) from the App component to the Header and Footer components using props. Ensure that the content displayed in the Header and Footer components is dynamically updated based on the data received from the parent component.
3.	Create a Counter Application using React that demonstrates state management with the useState hook. Display the current value of the counter prominently on the screen. Add buttons to increase and decrease the counter value. Ensure the counter updates dynamically when the buttons are clicked. Use the useState hook to manage the counter's state within the component. Prevent the counter from going below a specified minimum value (e.g., 0). Add a "Reset" button to set the counter back to its initial value. Include functionality to specify a custom increment or decrement step value.
4.	Develop a To-Do List Application using React functional components that demonstrates the use of the useState hook for state management. Create a functional component named ToDoFunction to manage and display the to- do list. Maintain a list of tasks using state. Provide an input field for users to add new tasks. Dynamically render the list of tasks below the input field. Ensure each task is displayed in a user-friendly manner. Allow users to delete tasks from the list. Mark tasks as completed or pending, and visually differentiate them.

5.	Develop a React application that demonstrates component composition and the use of props to pass data. Create two components: FigureList: A parent component responsible for rendering multiple child components. BasicFigure: A child component designed to display an image and its associated caption. Use the FigureList component to dynamically render multiple BasicFigure components. Pass image URLs and captions as props from the FigureList component to each BasicFigure component. Style the BasicFigure components to display the image and caption in an aesthetically pleasing manner. Arrange the BasicFigure components within the FigureList in a grid or list format. Allow users to add or remove images dynamically. Add hover effects or animations to the images for an interactive experience.
6.	Design and implement a React Form that collects user input for name, email, and password. Form Fields are Name, Email, Password. Ensure all fields are filled before allowing form submission. Validate the email field to
	ensure it follows the correct email format (e.g., example@domain.com). Optionally enforce a minimum password length or complexity. Display error messages for invalid or missing inputs. Provide visual cues (e.g., red borders) to highlight invalid fields. Prevent form submission until all fields pass validation. Log or display the entered data upon successful submission (optional). Add a "Show Password" toggle for the password field. Implement client- side sanitization to ensure clean input.
7.	Develop a React Application featuring a ProfileCard component to display a user's profile information, including their name, profile picture, and bio. The component should demonstrate flexibility by utilizing both external CSS and inline styling for its design. Display the following information: Profile picture, User's name, A short bio or description Use an external CSS file for overall structure and primary styles, such as layout, colors, and typography. Apply inline styles for dynamic or specific styling elements, such as background colors or alignment. Design the ProfileCard to be visually appealing and responsive. Ensure the profile picture is displayed as a circle, and the name and bio are appropriately styled. Add hover effects or animations to enhance interactivity. Allow the background color of the card to change dynamically based on a prop or state.
8.	Develop a Reminder Application that allows users to efficiently manage their tasks. The application should include the following functionalities: Provide a form where users can add tasks along with due dates. The form includes task name, Due date, An optional description. Display a list of tasks dynamically as they are added. Show relevant details like task name, due date, and completion status. Include a filter option to allow users to view all Tasks and Display all tasks regardless of status. Show only tasks marked as completed. Show only tasks that are not yet completed.
9.	Design a React application that demonstrates the implementation of routing using the react-router-dom library. The application should include the Navigation Menu: Create a navigation bar with links to three distinct pages, Home, About, Contact. Develop separate components for each page (Home, About, and Contact) with appropriate content to differentiate them. Configure routes using react-router-dom to render the corresponding page component based on the selected link. Use BrowserRouter and Route components for routing. Highlight the active link in the navigation menu to indicate the current page

10

Design a React application featuring a class-based component that demonstrates the use of lifecycle methods to interact with an external API. The component should fetch and update data dynamically based on user interactions or state changes. Use the `componentDidMount` lifecycle method to fetch data from an API when the component is initially rendered. Display the fetched data in a structured format, such as a table or list. Use the `componentDidUpdate` lifecycle method to detect changes in the component's state or props. Trigger additional API calls to update the displayed data based on user input or actions (e.g., filtering, searching, or pagination). Implement error handling to manage issues such as failed API requests or empty data responses. Display appropriate error messages to the user when necessary. Allow users to perform actions like filtering, searching, or refreshing the data. Reflect changes in the displayed data based on these interactions.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Illustrate React basics and state components.
- Develop React applications that utilize component composition, passing data through props.
- Use dynamic state updates, event handling, and custom logic to increment, decrement, and reset state values.
- Implement forms in React that collect and validate user input.
- Demonstrate interaction with external APIs, dynamic content generation and manage state in real-time applications.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.

- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in - 60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:**Books:**

1. Beginning React JS Foundations Building User Interfaces with ReactJS: An Approachable Guide, Chris Minnick, Wiley publications , 2022.
2. Learning React Functional Web Development with React and Redux , Alex Banks, Eve Porcello · 2017

Web links and Video Lectures (e-Resources):

- <https://www.youtube.com/watch?v=V9i3cGD-mts>
- <https://youtu.be/PHaECbrKgs0>
- <https://youtu.be/uvEAvxWvwOs>
- <https://www.geeksforgeeks.org/state-management-with-usestate-hook-in-react/>
- <https://youtu.be/KU-I2M9Jm68>
- https://youtu.be/H63Pd_IXkeQ
- <https://youtu.be/oTIJunBa6MA>
- <https://youtu.be/3EbYJrAOpUs>

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Apply java script to build dynamic and interactive web projects.
- Implement user interface components for javascript based web using React.js.
- Apply express/node to build web applications on the server side.
- Develop data model in an open source no sql database.
- Demonstrate modularization and packing of the front end modules.

CO-PO-PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	2	2	1	2				1	1				
CO2	1	2	3	1					1	1			2	
CO3	1	3	1	2	2				1	1				
CO4	2	1	1						1	1				
CO5	3	2	1	1			1						2	1

Mapping Justification:**CO-PO Mapping Justification**

CO1	Focuses on technical knowledge and analytical skills (PO1,PO2), Modern Tool Usage (PO5), , Collaboration and coordination to work in project (PO9, PO10).
CO2	Understanding of fundamentals of java script and analysis of problems to solve (PO2), Implementing solution to various problems(PO3),Collaboration and coordination to work in project (PO9, PO10).
CO3	Ability to grasp the required technical knowledge, analytical skills and investigating problem to interpret various aspects of (PO1,PO2,PO4) and , Modern Tool Usage (PO5),Collaboration and coordination to work in project (PO9, PO10).
CO4	Develop through understanding on technical knowledge, problem solving and probing aspect (PO1,PO2, PO3).
CO5	Emphasis on comprehending concepts of modularization and packaging, analyzing problems to find suitable use and design solutions (PO1,PO2,PO3), Development of sustainable solution (PO7).

PSO's Justifications

PSO1	Using the knowledge of concepts learnt, ability to investigate the problem and modern tool usage for developing software solutions and address many societal issues by conducting research(CO2,CO5).
PSO2	Adapting the knowledge of higher order skills obtained through critical thinking applied over many problems, consistent use of tools, packages and libraries, ability to function as a team thus facilitating entrepreneurship pursuit of student.(CO5)

Steps to install :

Step 1: Install Node.js and npm

React relies on Node.js and npm (Node Package Manager), so you'll need to install them first if you haven't already.

1. Go to the official [Node.js website](#).
2. Download the latest LTS version for your operating system (Windows/macOS/Linux).
3. Install it by following the instructions for your OS. This will also install npm, which comes bundled with Node.js.

Step 2: Install Visual Studio Code (VSCode)

If you haven't already installed VSCode, follow these steps:

1. Go to the [Visual Studio Code website](#).
2. Download and install VSCode for your operating system.

Step 3: Open VSCode and Install the Required Extensions

1. Launch VSCode.
2. **Install the following extensions** for better development experience:
 - **ES7 React, Redux/GraphQL, React-Native snippets:** This extension provides useful React snippets to speed up development.
 - **Prettier – Code formatter:** It helps in keeping your code clean and formatted automatically.
 - **Bracket Pair Colorizer:** Helps visually distinguish matching brackets.
 - **ESLint:** For JavaScript/React linting.

To install an extension:

- Press Ctrl+Shift+X (Windows/Linux) or Cmd+Shift+X (Mac) to open the Extensions panel.
- Search for the extension by name and click **Install**.

Step 4: Create a New React Project

You can use Create React App to set up a new React project easily. Here's how to do that:

1. **Open the terminal** in VSCode:
 - Press Ctrl+` (backtick) to open the terminal, or use the menu Terminal>New Terminal`.

Create a new React app using npx (no need to install globally):

```
npx create-react-app my-app
```

Replace my-app with the name of your project. This will create a new folder called my-app with all the necessary files and configurations.

Step 5: Navigate to Your Project Folder

Once the create-react-app process finishes, move into your project directory:

```
cd my-app
```

Step 7: Start the Development Server

To see your React app in action, run the following command in the VSCode terminal:

```
npm start
```

Step 8: Edit Your Code

Now that everything is set up, you can start modifying the code:

1. Open src/App.js in VSCode.
2. Make some changes to the code. For example, change the text inside the <h1> tag or modify the App component to see updates live in the browser.

```
import logo from './logo.svg';
import './App.css';

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

function App() {

  const [inputValue, setInputValue] = useState('jsdakjsd');
  const handleInputChange = (event) =>
  {
    console.log("event.target.value",event.target.value)

    setInputValue(event.target.value);};
  return (
    <div className="App">
      <h1>{inputValue}</h1>
      <input
        type="text"
        value={inputValue}
        onChange={handleInputChange}
        placeholder="Typeabc..."
      />
    </div>
  );
}

export default App;
```

1. Use create-react-app to set up a new project. Edit the App.js file to include a stateful component with `useState`. Add an input field and a `<h1>` element that displays text based on the input. Dynamically update the `<h1>` content as the user types

```
import logo from './logo.svg';
import './App.css';

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

function App() {

  const [inputValue, setInputValue] = useState('jsdakjsd');
  const handleInputChange = (event) =>
  {
    console.log("event.target.value",event.target.value)

    setInputValue(event.target.value);};
  return (
    <div className="App">
      <h1>{inputValue}</h1>
      <input
        type="text"
        value={inputValue}
        onChange={handleInputChange}
        placeholder="Typeabc..."
      />
    </div>
  );
}
export default App;
```

`npx create-react-app app.js`

Output

Dynamic Text Display

You typed: Hello vtucircle

2. Develop a React application that demonstrates the use of props to pass data from a parent component to child components. The application should include the parent component named App that serves as the central container for the application. Create two separate child components, Header: Displays the application title or heading. Footer: Displays additional information, such as copyright details or a tagline. Pass data (e.g., title, tagline, or copyright information) from the App component to the Header and Footer components using props. Ensure that the content displayed in the Header and Footer components is dynamically updated based on the data received from the parent component.

```
import React, { useState } from 'react';
import Header from './Header'; // Import the Header component
import Footer from './Footer'; // Import the Footer component
```

```
function App() {
  // Data to pass down as props
  // const title = "My React Application";
  // const tagline = "© 2025 All Rights Reserved";
  // Step 1: Declare a state variable to store the input value
  const [title, setTitle] = useState("");
  const [tagline, setTagline] = useState("");
  // © 2025 All Rights Reserved
  // Step 2: Handle the input change event
  const handleTitleChange = (event) => {
    setTitle(event.target.value);
  };
  // Step 2: Handle the input change event
  const handleTaglineChange = (event) => {
    setTagline(event.target.value);
  };
  return (
    <div className="App">
      {/* Pass the data to child components using props */}
      <Header
        title={title}
        onChange={handleTitleChange}
      />
      {/* Step 4: Add an input field to capture user input */}
      <input
        type="text"
        value={title}
        onChange={handleTitleChange}
        placeholder="Type Header..."
      />
      {/* Step 4: Add an input field to capture user input */}
      <input
```

```
type="text"
value={tagline}
onChange={handleTaglineChange}
placeholder="Type Footer..."
/>
<Footer
  tagline={tagline}
  onChange={handleTaglineChange}
/>
</div>
}
export default App;
create a Footer.js
import React from 'react';
// Footer component receives 'tagline' as a prop
function Footer(props) {
  return (
    <footer>
      <p>{props.tagline}</p>
    </footer>
  );
}
```

```
export default Footer;
```

create a Header.js

```
import React from 'react';
// Header component receives 'title' as a prop
function Header(props) {
  return (
    <header>
      <h1>{props.title}</h1>
    </header>
  );
}
```

```
export default Header;
```

npx create-react-app app2.js

create a Footer.js

create a Header.js

Npm Start

OUTPUT



3. Create a Counter Application using React that demonstrates state management with the `useState` hook. Display the current value of the counter prominently on the screen. Add buttons to increase and decrease the counter value. Ensure the counter updates dynamically when the buttons are clicked. Use the `useState` hook to manage the counter's state within the component. Prevent the counter from going below a specified minimum value (e.g., 0). Add a "Reset" button to set the counter back to its initial value. Include functionality to specify a custom increment or decrement step value.

```
import React, { useState } from 'react';
function App() {
  // Initialize the state for the counter, step, and minimum value
  const [counter, setCounter] = useState(0); // Counter value
  const [step, setStep] = useState(1);      // Step value (custom increment or decrement)
  const minVal = 0;                         // Minimum allowed counter value

  // Function to increase the counter
  const increaseCounter = () => {
    setCounter(prevCounter => prevCounter + step);
  };

  // Function to decrease the counter
  const decreaseCounter = () => {
    setCounter(prevCounter => Math.max(prevCounter - step, minVal)); // Prevent going below 0
  };

  // Function to reset the counter to the initial value (0)
  const resetCounter = () => {
    setCounter(0);
  };

  // Function to handle changes in the step value
  const handleStepChange = (event) => {
    const value = Number(event.target.value);
    if (value > 0) { // Ensure the step is positive
      setStep(value);
    }
  };

  return (
    <div className="App">
      <h1>Counter Application</h1>

      {/* Display the current counter value */}
      <div>
        <h2>{counter}</h2>
      </div>

      {/* Buttons for increasing and decreasing the counter */}
    </div>
  );
}
```

```
<div>
  <button onClick={increaseCounter}>Increase</button>
  <button onClick={decreaseCounter}>Decrease</button>
</div>

{/* Reset button to reset the counter */}
<div>
  <button onClick={resetCounter}>Reset</button>
</div>

{/* Input to set the custom step value */}
<div>
  <label>Set Step: </label>
  <input
    type="number"
    value={step}
    onChange={handleStepChange}
    min="1" // Ensure that the step is always positive
  />
</div>
</div>
);
}

export default App;
npx create-react-app app3.js
Npm Start
```

OUTPUT

Counter Application

0

Increase by 1

Decrease by 1

Reset

Set Increment/Decrement Step:

4. Develop a To-Do List Application using React functional components that demonstrates the use of the `useState` hook for state management. Create a functional component named `ToDoFunction` to manage and display the to-do list. Maintain a list of tasks using state. Provide an input field for users to add new tasks. Dynamically render the list of tasks below the input field. Ensure each task is displayed in a user-friendly manner. Allow users to delete tasks from the list. Mark tasks as completed or pending, and visually differentiate them.

```
import React, { useState } from 'react';
function ToDoFunction() {
  // State for tasks list and the input value
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState("");

  // Function to handle the input change
  const handleInputChange = (e) => {
    setNewTask(e.target.value);
  };

  // Function to handle adding a new task
  const addTask = () => {
    if (newTask.trim()) {
      const newTaskObj = {
        id: Date.now(), // Unique id based on timestamp
        text: newTask,
        completed: false,
      };
      setTasks([...tasks, newTaskObj]);
      setNewTask(""); // Clear input after adding
    }
  };

  // Function to toggle task completion status
  const toggleTaskCompletion = (taskId) => {
    setTasks(tasks.map((task) =>
      task.id === taskId ? { ...task, completed: !task.completed } : task
    ));
  };

  // Function to delete a task
  const deleteTask = (taskId) => {
    setTasks(tasks.filter((task) => task.id !== taskId));
  };

  return (
    <div>
      <h1>To-Do List</h1>

      { /* Input field for new task */ }
```

```

<input
  type="text"
  value={newTask}
  onChange={handleInputChange}
  placeholder="Add a new task"
/>

<button onClick={addTask}>Add Task</button>
{/* List of tasks */}
<ul>
  {tasks.map((task) => (
    <li key={task.id} style={{ margin: '10px 0' }}>
      {/* Task text */}
      <span
        onClick={() => toggleTaskCompletion(task.id)}
        style={{
          textDecoration: task.completed ? 'line-through' : 'none',
          cursor: 'pointer',
        }}
      >
        {task.text}
      </span>

      {/* Delete button */}
      <button
        onClick={() => deleteTask(task.id)}
        style={{ marginLeft: '10px', color: 'red' }}
      >
        Delete
      </button>
    </li>
  ))}
</ul>
</div>
);
}

export default TodoFunction;
create a TodoFunction
import React from 'react';
import TodoFunction from './TodoFunction';
import './App4.css';
function App() {
  return (

```

```
<div className="App">  
  <ToDoFunction />  
</div>  
);  
}
```

export default App;

npx create-react-app app4.js

create a ToDoFunction.js

Npm Start

OUTPUT

To-Do List

Add Task

Hello vtucircle upload react lab program

react lab programs has been uploaded

5. Develop a React application that demonstrates component composition and the use of props to pass data. Create two components: **FigureList**: A parent component responsible for rendering multiple child components. **BasicFigure**: A child component designed to display an image and its associated caption. Use the **FigureList** component to dynamically render multiple **BasicFigure** components. Pass image URLs and captions as props from the **FigureList** component to each **BasicFigure** component. Style the **BasicFigure** components to display the image and caption in an aesthetically pleasing manner. Arrange the **BasicFigure** components within the **FigureList** in a grid or list format. Allow users to add or remove images dynamically. Add hover effects or animations to the images for an interactive experience.

```
import React, { useState } from 'react';
import FigureList from './FigureList';
import './App.css'; // Importing the CSS file
function App() {

  const [figures, setFigures] = useState([
    { id: 1, imageUrl: 'https://images.pexels.com/photos/20787/pexels-photo.jpg?auto=compress&cs=tinysrgb&h=350', caption: 'Image 1' }, // From public/images folder
    { id: 2, imageUrl: 'https://images.unsplash.com/photo-1457305237443-44c3d5a30b89?q=80&w=2948&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D', caption: 'Image 2' }, // From public/images folder
    // { id: 3, imageUrl: 'https://images.unsplash.com/photo-1498050108023-c5249f4df085?q=80&w=1172&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D', caption: 'Image 3' },
  ]);

  const addFigure = () => {
    const newFigure = {
      id: Date.now(), // Unique ID based on timestamp
      imageUrl: 'https://images.unsplash.com/photo-1498050108023-c5249f4df085?q=80&w=1172&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D', // New placeholder image
      caption: `New Image ${figures.length + 1}`, // Generate a new caption
    };
    setFigures([...figures, newFigure]); // Add new figure to the state
    console.log("addfigures", figures);
  };

  // console.log("111111111111111111")

  const removeFigure = (id) => {
    console.log("removeFigure", id);
    setFigures(figures.filter((figure) => figure.id !== id)); // Remove figure by ID
  };
}
```

```
    console.log("figures",figures)
  };
// If the id of the current figure is not equal to the id passed to the function, the figure will be included in
the new array.
// If the id of the current figure matches the id passed in, the figure will not be included in the new array.
return (
  <div className="App">
    <h1>Image Gallery</h1>
    <FigureList figures={figures} removeFigure={removeFigure} />
    <button className="add-button" onClick={addFigure}>Add Image</button>
  </div>
);
}
```

export default App;

create a BasicFigure.js

import React from 'react';

function BasicFigure({ imageUrl, caption, onRemove }) {

```
  return (
    <div className="figure-container">
      <img src={imageUrl} alt={caption} className="figure-image" />

      <div className="figure-caption">
        <p>{caption}</p>
        <button onClick={onRemove} className="remove-button">Remove</button>
      </div>
    </div>
  );
}
```

export default BasicFigure;

create a Figurelist.js

import React from 'react';

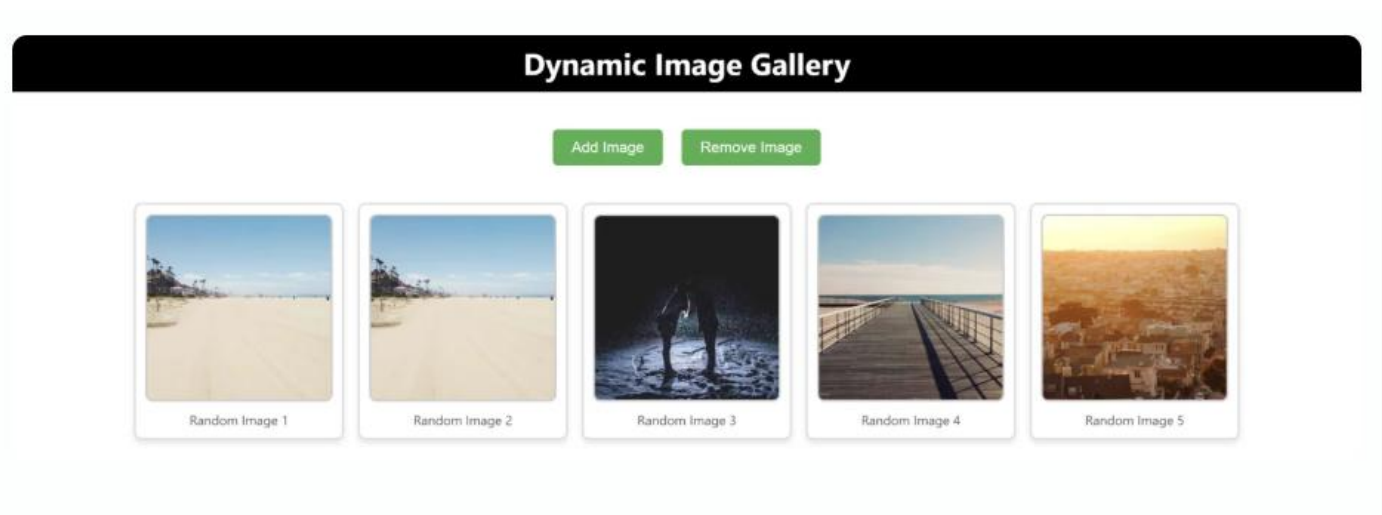
import BasicFigure from './BasicFigure';

function FigureList({ figures, removeFigure }) {

```
  return (
    <div className="figure-list">
      {figures.map(figure => (
        <BasicFigure
          key={figure.id}
          imageUrl={figure.imageUrl}
          caption={figure.caption}
          onRemove={() => removeFigure(figure.id)} // Passing remove function
        />
      )
    </div>
  );
}
```

```
    )})  
  </div>  
);  
}  
export default FigureList;  
npx create-react-app app5.js  
add a BasicFigure.js  
create a FigureList.js  
Npm Start
```

OUTPUT



6. Design and implement a React Form that collects user input for name, email, and password. Form Fields are Name, Email, Password. Ensure all fields are filled before allowing form submission. Validate the email field to ensure it follows the correct email format (e.g., example@domain.com). Optionally enforce a minimum password length or complexity. Display error messages for invalid or missing inputs. Provide visual cues (e.g., red borders) to highlight invalid fields. Prevent form submission until all fields pass validation. Log or display the entered data upon successful submission (optional). Add a “Show Password” toggle for the password field. Implement client-side sanitization to ensure clean input.

```
import React, { useState } from 'react';
import './App.css';

const Form = () => {
  // State to manage form fields and validation errors
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
  });

  const [errors, setErrors] = useState({
    name: "",
    email: "",
    password: "",
  });

  const [showPassword, setShowPassword] = useState(false);

  // Function to handle form data change
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };

  // Function to handle password visibility toggle
  const togglePasswordVisibility = () => {
    setShowPassword(!showPassword);
  };

  // Function to validate email format
  const isValidEmail = (email) => {
    const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
```

```
    return emailRegex.test(email);
  };

  // Function to validate password length
  const isValidPassword = (password) => {
    return password.length >= 6; // For example, minimum 6 characters
  };

  // Function to handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();

    // Reset errors
    let validationErrors = { name: "", email: "", password: "" };
    let isValid = true;

    // Validate name
    if (formData.name === "") {
      validationErrors.name = 'Name is required';
      isValid = false;
    }

    // Validate email
    if (formData.email === "") {
      validationErrors.email = 'Email is required';
      isValid = false;
    } else if (!isValidEmail(formData.email)) {
      validationErrors.email = 'Invalid email format';
      isValid = false;
    }

    // Validate password
    if (formData.password === "") {
      validationErrors.password = 'Password is required';
      isValid = false;
    } else if (!isValidPassword(formData.password)) {
      validationErrors.password = 'Password must be at least 6 characters';
      isValid = false;
    }

    // Set errors if validation fails
    setErrors(validationErrors);

    // If form is valid, log the data (or submit it)
```

```
if (isValid) {
  console.log('Form submitted successfully:', formData);
  alert('Form submitted successfully');
  // Reset form
  setFormData({
    name: "",
    email: "",
    password: "",
  });
}
};

return (
  <div className="form-container" style={{ maxWidth: '400px', margin: '0 auto' }}>
    <h2>Registration Form</h2>
    <form onSubmit={handleSubmit} noValidate>
      {/* Name Field */}
      <div className="form-group">
        <label>Name:</label>
        <input
          type="text"
          name="name"
          value={formData.name}
          onChange={handleChange}
          className={`form-input ${errors.name ? 'invalid' : ''}`}
          placeholder="Enter your name"
        />
        {errors.name && <p className="error">{errors.name}</p>}
      </div>

      {/* Email Field */}
      <div className="form-group">
        <label>Email:</label>
        <input
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          className={`form-input ${errors.email ? 'invalid' : ''}`}
          placeholder="Enter your email"
        />
        {errors.email && <p className="error">{errors.email}</p>}
      </div>

      {/* Password Field */}
```

```

<div className="form-group">
  <label>Password:</label>
  <input
    type={showPassword ? 'text' : 'password'}
    name="password"
    value={formData.password}
    onChange={handleChange}
    className={`form-input ${errors.password ? 'invalid' : ''}`}
    placeholder="Enter your password"
  />
  <button type="button" onClick={togglePasswordVisibility}>
    {showPassword ? 'Hide Password' : 'Show Password'}
  </button>
  {errors.password && <p className="error">{errors.password}</p>}
</div>

{/* Submit Button */}
<button type="submit" disabled={Object.values(errors).some((error) => error !== "")}>
  Submit
</button>
</form>
</div>
);
};

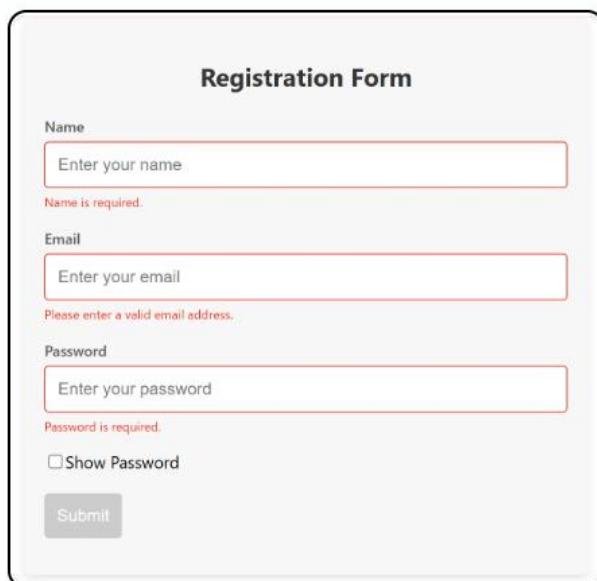
```

export default Form;

npx create-react-app app6.js

Npm Start

OUTPUT



Registration Form

Name

Name is required.

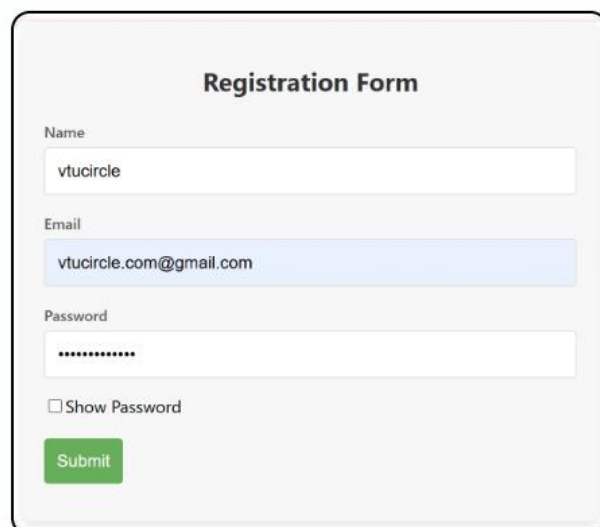
Email

Please enter a valid email address.

Password

Password is required.

☐ Show Password



Registration Form

Name

Email

Password

☐ Show Password

7. Develop a React Application featuring a ProfileCard component to display a user's profile information, including their name, profile picture, and bio. The component should demonstrate flexibility by utilizing both external CSS and inline styling for its design. Display the following information: Profile picture, User's name, A short bio or description Use an external CSS file for overall structure and primary styles, such as layout, colors, and typography. Apply inline styles for dynamic or specific styling elements, such as background colors or alignment. Design the ProfileCard to be visually appealing and responsive. Ensure the profile picture is displayed as a circle, and the name and bio are appropriately styled. Add hover effects or animations to enhance interactivity. Allow the background color of the card to change dynamically based on a prop or state.

```
import React, { useState } from 'react';
import './ProfileCard.css'; // External CSS

const ProfileCard = ({ name, bio, profilePicture, initialBgColor }) => {
  // State for dynamic background color
  const [bgColor, setBgColor] = useState(initialBgColor);

  // Function to change background color on hover
  const handleMouseEnter = () => {
    setBgColor('lightblue'); // Change background color on hover
    // setBgColor('#666666');
  };

  const handleMouseLeave = () => {
    setBgColor(initialBgColor); // Revert to initial background color
  };

  // Inline styles for specific elements
  const profileCardStyles = {
    backgroundColor: bgColor,
  };
  return (
    <div
      className="profile-card"
      style={profileCardStyles}
      // style={{backgroundColor: bgColor}}
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    >
      <img
        src={profilePicture}
        alt="Profile"
        className="profile-picture"
      />
    </div>
  );
};
```

```

    />
    <h2 className="name">{name}</h2>
    <p className="bio">{bio}</p>
  </div>
);
};
export default ProfileCard;
Creat a ProfileCard
import React from 'react';
import ProfileCard from './ProfileCard';
function App() {
  return (
    <div className="App">
      <ProfileCard
        name="John Doe"
        bio="Software engineer with a passion for building great user experiences."
        profilePicture="https://images.unsplash.com/photo-1498050108023-
c5249f4df085?q=80&w=1172&auto=format&fit=crop&ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D"
        initialBgColor="#ffffff" // Initial background color
      />
      <ProfileCard
        name="Jane Smith"
        bio="Graphic designer and artist, loves creating visually engaging designs."
        profilePicture="https://images.unsplash.com/photo-1457305237443-
44c3d5a30b89?q=80&w=2948&auto=format&fit=crop&ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D"
        initialBgColor="#f7f7f7" // Initial background color
      />
    </div>
  );
}
export default App;
npx create-react-app app7.js
Create a ProfileCard.js
Npm Start

```

OUTPUT



8. Develop a Reminder Application that allows users to efficiently manage their tasks. The application should include the following functionalities: Provide a form where users can add tasks along with due dates. The form includes task name, Due date, An optional description. Display a list of tasks dynamically as they are added. Show relevant details like task name, due date, and completion status. Include a filter option to allow users to view all Tasks and Display all tasks regardless of status. Show only tasks marked as completed. Show only tasks that are not yet completed.

```
import React, { useState } from 'react';
import './ReminderApp.css'; // External CSS (optional, for styling)

const ReminderApp = () => {
  const [tasks, setTasks] = useState([]); // To hold tasks
  const [taskName, setTaskName] = useState("");
  const [dueDate, setDueDate] = useState("");
  const [description, setDescription] = useState("");
  const [filter, setFilter] = useState('all'); // Filter state: 'all', 'completed', 'incomplete'

  // Add Task function
  const addTask = (e) => {
    e.preventDefault();
    if (taskName && dueDate) {
      const newTask = {
        id: Date.now(),
        name: taskName,
        dueDate: dueDate,
        description: description || 'No description provided',
        isCompleted: false,
      };
      setTasks([...tasks, newTask]);
      setTaskName("");
      setDueDate("");
      setDescription("");
    } else {
      alert('Please fill out both task name and due date!');
    }
  };

  // Toggle task completion status
  const toggleCompletion = (id) => {
    setTasks(
      tasks.map((task) =>
        task.id === id ? { ...task, isCompleted: !task.isCompleted } : task
      )
    );
  };
};
```

```

    );
  };
  // Filter tasks based on status
  const filteredTasks = tasks.filter((task) => {
    if (filter === 'completed') return task.isCompleted;
    if (filter === 'incomplete') return !task.isCompleted;
    return true; // Show all tasks
  });
  return (
    <div className="reminder-app">
      <h1>Reminder Application</h1>

      {/* Task Form */}
      <form onSubmit={addTask} className="task-form">
        <input
          type="text"
          placeholder="Task Name"
          value={taskName}
          onChange={(e) => setTaskName(e.target.value)}
        />
        <input
          type="date"
          value={dueDate}
          onChange={(e) => setDueDate(e.target.value)}
        />
        <textarea
          placeholder="Description (optional)"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
        />
        <button type="submit">Add Task</button>
      </form>
      {/* Filter Options */}
      <div className="filter-options">
        <button onClick={() => setFilter('all')}>All Tasks</button>
        <button onClick={() => setFilter('completed')}>Completed</button>
        <button onClick={() => setFilter('incomplete')}>Incomplete</button>
      </div>
      {/* Task List */}
      <ul className="task-list">
        {filteredTasks.length === 0 ? (
          <p>No tasks found</p>
        ) : (
          filteredTasks.map((task) => (

```



```

<li
  key={task.id}
  className={`task-item ${task.isCompleted ? 'completed' : ''}`}
>
  <div>
    <h3>{task.name}</h3>
    <p>Due Date: {task.dueDate}</p>
    <p>{task.description}</p>
  </div>
  <div>
    <button onClick={() => toggleCompletion(task.id)}>
      {task.isCompleted ? 'Mark Incomplete' : 'Mark Completed'}
    </button>
  </div>
</li>
))
}}
</ul>
</div>
);
};

```

```

export default ReminderApp;
npx create-react-app app8.js
npm start

```

OUTPUT

Task Reminder

Task Name

mm/dd/yyyy

Description (optional)

Add Task

All Tasks

Completed Tasks

Pending Tasks

No tasks available!

Task Reminder

Task Name

mm/dd/yyyy

Description (optional)

Add Task

All Tasks

Completed Tasks

Pending Tasks

Upload React Lab Program

Due Date: 2025-02-02

Description: Upload the completed React lab program.

Status: Not Completed

Mark as Completed **Delete**

9. Design a React application that demonstrates the implementation of routing using the react-router-dom library. The application should include the Navigation Menu: Create a navigation bar with links to three distinct pages, Home, About, Contact. Develop separate components for each page (Home, About, and Contact) with appropriate content to differentiate them. Configure routes using react-router-dom to render the corresponding page component based on the selected link. Use BrowserRouter and Route components for routing. Highlight the active link in the navigation menu to indicate the current page.

APP.JS

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes, NavLink } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
import './App.css'; // You can use this for styling

function App() {
  return (
    <Router>
      <div className="app">
        { /* Navigation Bar */ }
        <nav className="navbar">
          <ul>
            <li>
              <NavLink
                exact
                to="/"
                className={({ isActive }) => (isActive ? 'nav-link active-link' : 'nav-link')}
              >
                Home
              </NavLink>
            </li>
            <li>
              <NavLink
                to="/about"
                className={({ isActive }) => (isActive ? 'nav-link active-link' : 'nav-link')}
              >
                About
              </NavLink>
            </li>
            <li>
              <NavLink
```

```
      to="/contact"
      className={({ isActive }) => (isActive ? 'nav-link active-link' : 'nav-link')}
    >
      Contact
    </NavLink>
  </li>
</ul>
</nav>

{/* Routing */}
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
</Routes>
</div>
</Router>
);
}
```

export default App;

contact.js

import React from 'react';

const Contact = () => {

return (

<div className="page-content">

<h1>Contact Us</h1>

<p>If you have any questions, feel free to reach out to us.</p>

</div>

);

};

Home.js

export default Contact;

import React from 'react';

const Home = () => {

return (

<div className="page-content">

<h1>Welcome to the Home Page</h1>

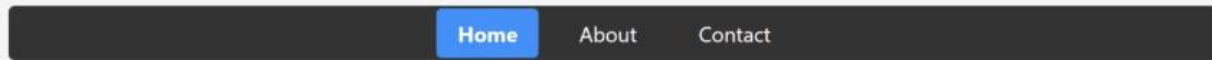
<p>This is the home page of our website.</p>

</div>

);

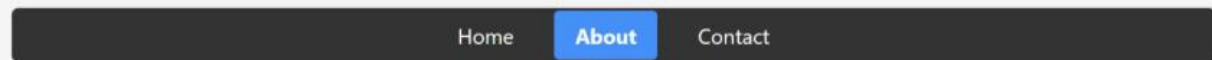
```
};  
export default Home;  
npx create-react-app app9.js  
npm start
```

OUTPUT



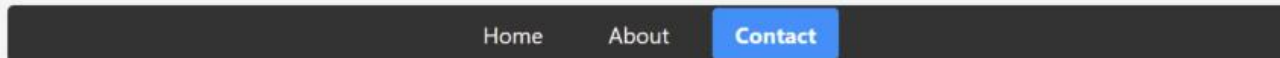
Home Page

Welcome to the Home Page!



About Page

Learn more about us on the About Page!



Contact Page

Get in touch with us through the Contact Page!

10. Design a React application featuring a class-based component that demonstrates the use of lifecycle methods to interact with an external API. The component should fetch and update data dynamically based on user interactions or state changes. Use the `componentDidMount` lifecycle method to fetch data from an API when the component is initially rendered. Display the fetched data in a structured format, such as a table or list. Use the `componentDidUpdate` lifecycle method to detect changes in the component's state or props. Trigger additional API calls to update the displayed data based on user input or actions (e.g., filtering, searching, or pagination). Implement error handling to manage issues such as failed API requests or empty data responses. Display appropriate error messages to the user when necessary. Allow users to perform actions like filtering, searching, or refreshing the data. Reflect changes in the displayed data based on these interactions.

APP.JS

```
import React, { Component } from 'react';
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      users: [],
      filteredUsers: [],
      isLoading: false,
      error: null,
      searchQuery: "",
    };
  }

  // Fetch data when the component mounts
  componentDidMount() {
    this.fetchUsers();
  }

  // Fetch users from the API
  fetchUsers = async () => {
    this.setState({ isLoading: true });
    try {
      const response = await fetch('https://jsonplaceholder.typicode.com/users');
      console.log("response of api", response);
      if (!response.ok) {
        throw new Error('Failed to fetch data');
      }
      const data = await response.json();
      this.setState({ users: data, filteredUsers: data, isLoading: false });
    } catch (error) {
      this.setState({ error: error.message, isLoading: false });
    }
  }
}
```

```
};

// Detect changes in searchQuery and filter the users accordingly
componentDidUpdate(prevProps, prevState) {
  if (prevState.searchQuery !== this.state.searchQuery) {
    this.filterUsers(this.state.searchQuery);
  }
}

// Filter users by search query
filterUsers = (query) => {
  const filteredUsers = this.state.users.filter((user) =>
    user.name.toLowerCase().includes(query.toLowerCase())
  );
  this.setState({ filteredUsers });
};

// Handle search input change
handleSearchChange = (event) => {
  this.setState({ searchQuery: event.target.value });
};

// Handle refresh button click
handleRefresh = () => {
  this.fetchUsers();
};

render() {
  const { filteredUsers, isLoading, error, searchQuery } = this.state;

  return (
    <div className="App">
      <h1>User List</h1>

      {/* Search and Refresh Controls */}
      <div>
        <input
          type="text"
          placeholder="Search by name"
          value={searchQuery}
          onChange={this.handleSearchChange}
        />
        <button onClick={this.handleRefresh} disabled={isLoading}>
          {isLoading ? 'Refreshing...' : 'Refresh Data'}
        </button>
      </div>
    </div>
  );
}
```

```

</div>
{/* Display Errors */}
{error && <p style={{ color: 'red' }}>{error}</p>}

{/* Display Data */}
{isLoading ? (
  <p>Loading...</p>
) : (
  <ul>
    {filteredUsers.length === 0 ? (
      <p>No users found</p>
    ) : (
      filteredUsers.map((user) => (
        <li key={user.id}>
          <h2>{user.name}</h2>

          <p>{user.email}</p>
          <p>{user.phone}</p>
        </li>
      ))
    )}
  </ul>
)}
</div>
);
}
}
export default App;
npx create-react-app app10.js
npm start

```

OUTPUT

User Data		
<input type="text" value="Search by name"/>		
Name	Email	City
Leanne Graham	Sincere@april.biz	Gwenborough
Ervin Howell	Shanna@melissa.tv	Wisokyburgh
Clementine Bauch	Nathan@yesenia.net	McKenziehaven
Patricia Lebsack	Julianne.OConner@kory.org	South Elvis
Chelsey Dietrich	Lucio_Hettinger@annie.ca	Roscoeview
Mrs. Dennis Schulist	Karley_Dach@jasper.info	South Christy
Kurtis Weissnat	Telly.Hoeger@billy.biz	Howemouth
Nicholas Runolfsdottir V	Sherwood@rosamond.me	Aliyaview
Glenna Reichert	Chaim_McDermott@dana.io	Bartholomebury
Clementina DuBuque	Rey.Padberg@karina.biz	Lebsackbury
<input type="button" value="Refresh Data"/>		

VIVA-VOCE**1. What is `create-react-app`?**

`create-react-app` is a command-line tool provided by Facebook to quickly set up a new React project with a pre- configured development environment, including Webpack, Babel, and ESLint.

2. How do you create a new React project using `create-react-app`?

You can create a new React project by running:

```
npx create-react-app my-app
```

Then navigate into the project folder:

```
cd my-app
```

And start the development server with:

```
npm start
```

3. What is the purpose of `useState` in React?

`useState` is a React Hook that allows functional components to have local state. It returns a stateful value and a function to update it.

4. How do you import and use `useState` in a component?

```
js
import React, { useState } from 'react';
function App() {
  const [text, setText] = useState("");
}
```

5. What happens when you type in the input field in your project?

As the user types in the input field, the `onChange` event handler updates the state using `setText`. The `

` element re-renders automatically to display the updated text in real-time.

6. What would happen if you didn't use `setText`?

If we didn't use `setText`, the state wouldn't change, and React wouldn't re-render the component with the new value. The `

` would stay the same regardless of what the user types.

7. What are props in React?

Props (short for "properties") are used to pass data from one component to another in React, typically from a parent to a child component. They are read-only and help in creating reusable components.

8. What is the difference between props and state?

- Props are used to pass data from parent to child and are read-only.
- State is local to a component and can be changed internally by the component.

9. Can you modify props inside a child component?

No, props are immutable in the child component. If a child component needs to modify data, it should either request the parent to update it via callbacks or manage it as state.

10. What will happen if you change the data in the `App` component?

If we update the values of the props in the `App` component (like the `title` or `tagline`), the child components (`Header` and `Footer`) will automatically re-render to reflect the new values.

11. Can a child component pass data back to the parent?

Yes, but not directly through props. You can pass a callback function as a prop from the parent to the child, and the child can call it to send data back.

12. What is `useState` in React?

`useState` is a Hook that allows you to add state to functional components in React. It returns an array with two elements: the current state and a function to update it.

13. Why is `useState` used in the counter app?

It is used to manage the `counter` value, so that every time the state updates, the component re-renders and displays the new counter value.

14. Can you explain your Counter App functionality?

Yes. My Counter App includes:

- Display of the current counter value.
- Buttons to increase, decrease, and reset the counter.
- A step input to specify how much to increase or decrease.
- Prevention of the counter from going below 0.

15. What is the purpose of the `step` state?

The `step` state allows users to define how much they want to increase or decrease the counter at a time. It adds flexibility to the counter.

16. Why do you use `Number(step)` or `parseInt(step)`?

Since the input from `e.target.value` is always a string, using `Number()` converts it to a number for arithmetic operations.

17. How does the reset button work?

It calls the `setCounter(0)` function, which sets the counter back to its initial state, 0.

18. Can you make this counter reusable across other components?

Yes, by converting it into a reusable component and passing initial value and step as props.

19. How does React know to re-render when a button is clicked?

When we call `setCounter`, it changes the state. React detects this change and re-renders the component to reflect the updated state in the UI.

20. How can you improve the UX of this counter app?

Some improvements:

- Disable the "Decrease" button when counter is 0.
- Add min/max validation for the step.
- Use PropTypes or TypeScript for type checking.
- Add animations or sound feedback on click.

21. Is the counter app a controlled or uncontrolled component?

The input field for the step is a controlled component because its value is bound to React state (`step`) and is updated via `onChange`.

22. Why use functional components instead of class components in modern React?

Functional components are simpler and easier to read. With the introduction of Hooks like `useState`, functional components can now manage state and side effects, which were previously only possible in class components.

23. Can you describe the structure and functionality of your To-Do List App?

Yes. The app includes:

- A functional component named `ToDoFunction`.
- An input field to add new tasks.
- A state variable to store the list of tasks.
- Ability to delete tasks.
- Ability to toggle a task between completed and pending.
- Tasks are styled differently based on their completion status.

24. How does the input field add a new task to the list?

The input is tied to a state variable. On clicking "Add Task" or pressing Enter:

1. A new task object is created.
2. It is added to the existing task array using the spread operator.
3. The list updates automatically due to state change.

25. Why did you use `Date.now()` for generating task IDs?

It provides a quick and mostly unique number based on the current timestamp. For production apps, using UUIDs or libraries like `nanoid` is better for guaranteed uniqueness.

26. How can you improve this app further?

- Add task editing functionality.
- Persist tasks in `localStorage`.
- Add categories or deadlines.
- Implement filtering (e.g., All / Completed / Pending).

27. Is your input a controlled component?

Yes, the input value is tied to React state (`newTask`), and its updates are handled via `onChange`.

28. What will happen if you try to add an empty task?

The ``addTask`` function has a condition to check if the task is empty using ``trim()``, and if so, it doesn't add it to the list.

29. What is component composition in React?

Component composition is the practice of building complex UIs by combining smaller, reusable components. It promotes modularity, reusability, and easier maintenance.

30. What are props in React?

Props (short for "properties") are read-only data passed from a parent component to a child component. They allow components to be dynamic and reusable.

31. Why do we use props instead of global variables or state in child components?

Props allow data flow in a unidirectional way (from parent to child), keeping components predictable and making state management easier.

32. Can you describe the purpose of your application?

Yes. The app:

- Uses a parent component ``FigureList`` to manage and render multiple image components.
- Each image is displayed using the ``BasicFigure`` child component.
- Image URLs and captions are passed as props from ``FigureList`` to ``BasicFigure``.
- Users can add or remove images.
- Images are styled and arranged in a grid with hover effects.

33. How do you handle dynamic addition and deletion of figures?

- Addition: Inputs capture the new URL and caption, and on button click, a new object is pushed to the array using ``setFigures``.
- Deletion: The ``onRemove`` function passed to each ``BasicFigure`` filters out the figure with the matching ID.

34. How do you ensure each ``BasicFigure`` component is unique in the list?

Each figure object has a unique ``id`` (generated using ``Date.now()``), which is used as the ``key`` in ``map()`` and to identify it for deletion.

35. Why do you lift state to the ``FigureList`` component?

Because the ``FigureList`` component manages the data and needs to pass props to multiple ``BasicFigure`` components. Keeping state in the parent enables consistent and centralized state management.

36. What kind of hover or animation effects did you use?

I used a simple scale transform and box-shadow to make images "pop" on hover. These transitions are handled using CSS.

37. How can you improve the app user experience further?

- Add animations when images are added or removed.
- Validate URLs before adding.
- Use a modal or lightbox to enlarge images on click.
- Store the list in localStorage or backend

38. Is `BasicFigure` a presentational or container component?

`BasicFigure` is a ****presentational**** component. It only displays data passed via props and doesn't manage any internal logic or state.

39. What is the purpose of this React Form application?

The form collects user input for Name, Email, and Password, validates the fields, and only allows submission when all inputs are valid. It also provides feedback through error messages and visual cues.

40. What are controlled components in React and how are they used in forms?

Controlled components are input elements whose value is controlled by React state. The form fields are tied to state variables and updated using `onChange` handlers.

41. What is client-side validation?

Client-side validation is the process of checking user inputs in the browser before sending them to a server. It improves UX by providing immediate feedback and reducing unnecessary server requests.

Project-Specific Questions

42. How did you manage form state in your component?

I used the `useState` hook to manage form field values (`name`, `email`, `password`) and separate state for error messages or validation flags.

43. What validations did you apply to the form fields?

- Name: Required.
- Email: Must match a regex pattern (e.g., `example@domain.com`).
- Password: Required with minimum length (e.g., 6 characters).

44. How did you prevent form submission if validation fails?

In the `handleSubmit` function, I checked for:

- Non-empty fields
- Valid email pattern
- Password length

If any validation failed, the form did ****not**** submit, and error messages were shown.

45. What happens after the form is successfully submitted?

If all validations pass, the entered data is either:

- Logged to the console (`console.log(formData)`)
- Or shown on screen in a confirmation message.

46. How would you improve this form for real-world production use?

- Add server-side validation.
- Add password strength indicators.
- Use more robust form libraries (e.g., `Formik` or `React Hook Form`).
- Store form data securely if needed.

47. What is the purpose of the `ProfileCard` component in your application?

The `ProfileCard` component displays a user's profile details, including their name, profile picture, and bio. It's styled using both external CSS and inline styles to demonstrate flexibility and dynamic behavior.

48. What are the advantages of using both external CSS and inline styles?

- External CSS is great for global styles and reusability (e.g., layout, fonts, spacing).
 - Inline styles are useful for dynamic styling (e.g., changing background color based on props or state).
- Combining both offers flexibility and control over styling.

49. How did you make the component responsive?

I used media queries in the external CSS and flexible units like `em`, `%`, and `vw/vh`. The card layout adjusts automatically on smaller screens using CSS Grid or Flexbox.

50. Why use `object-fit: cover` for the profile image?

It ensures that the image fills the circle without distortion, cropping excess parts while keeping the image centered.

51. Could you integrate animations using a library instead of CSS?

Yes. Libraries like **Framer Motion** or **React Spring** can add rich animations to the card with more control.

52. What is the goal of your Reminder Application?

The Reminder App helps users manage tasks by allowing them to:

- Add tasks with a due date and optional description.
- View a list of tasks with their completion status.
- Filter tasks by all, completed, or pending status.

53. Which React hooks did you use in this project?

I used:

- `useState` to manage the form inputs and the task list.
- Optionally, `useEffect` if I wanted to persist data (e.g., to localStorage).

54. What fields are included in your task form?

- Task name (required)
- Due date (required)
- Description (optional)

These are controlled inputs using React state.

55. What kind of validation did you use for the task form?

- Task name and due date are required.
- I check this before adding the task and show error messages if missing.

56. What happens if a user tries to add a task without a name or date?

The form won't submit, and an error message is shown next to the empty input fields.

1. What is the purpose of the `react-router-dom` library in your application?

The `react-router-dom` library is used to enable routing in a React application, allowing users to navigate between different pages (or views) without triggering a full page reload. It handles URL changes, rendering the appropriate components for each route.

57. What are the main components provided by `react-router-dom` for routing?

The main components are:

- `BrowserRouter`: Wraps the entire app and manages the browser's history.
- `Route`: Defines a mapping between a URL and a component.
- `Link`: Creates navigational links to different routes.
- `Switch`: (Optional in later versions) Renders the first `Route` that matches the current location.

58. How do you ensure that your application is user-friendly and intuitive?

- The navigation menu is clear and easily accessible, making it simple for users to switch between pages.
- Active links are highlighted to inform users about the current page.
- The layout is clean and the content is appropriately structured.

59. How do you prevent page reloads when navigating to a different route?

`react-router-dom` automatically prevents page reloads by using client-side routing. The `Link` and `NavLink` components handle navigation without triggering a full page reload, providing a smoother user experience.

60. How do you prevent duplicate tasks?

I can prevent duplicates by checking if a task with the same name and date already exists before adding it.

61. What is the purpose of lifecycle methods in React class components?

Lifecycle methods in React class components allow developers to hook into specific points of a component's lifecycle, such as when it is created, updated, or destroyed. These methods enable you to perform actions like fetching data, updating the UI, or cleaning up resources. The key lifecycle methods include `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

62. Why did you use a class-based component for this application?

I used a class-based component to demonstrate the use of lifecycle methods. While functional components with hooks (like `useEffect`) are common today, class components still provide a good understanding of React's lifecycle and how it manages state and updates over time.

63. What error messages do you display to the user when the API request fails?

I display a generic error message like "Failed to fetch data" along with a button to retry the action. The user can click the button to attempt to fetch the data again.

64. How would you handle API rate limiting or very large data sets in this application?

To handle API rate limiting, I would implement debouncing for searches or filters to minimize the number of API calls. For large data sets, I would implement pagination or infinite scrolling to load data in chunks, reducing the amount of data fetched at once.

65. How would you optimize the performance of this class-based component?*

To optimize performance, I would:

- Use `shouldComponentUpdate` to prevent unnecessary re-renders when the state or props haven't changed.
- Avoid fetching data on every state change unless it's necessary (e.g., debounce user input).
- Consider using React.memo or PureComponent for any child components to optimize rendering.