Create a simple HTML webpage that displays a heading, a paragraph, and an image. Apply basic CSS to change the background color, text color, and font style.
Design a webpage using HTML lists to display a navigation menu. Use CSS to style the menu horizontally with hover effects.
Create an HTML table showing student details (Roll No, Name, Class, Marks). Use CSS to add borders, padding, and alternate row colors.
Design a registration form using HTML form elements (text box, radio button, checkbox, submit button). Style the form using CSS for proper alignment.
Create a webpage that demonstrates the use of CSS box model (margin, border, padding, and content). Clearly show the effect of each property.
Develop a webpage using div elements and apply CSS to create a two-column layout.
Create an HTML page using semantic elements (header, nav, section, footer) and style them using CSS.
Design a webpage with an image gallery using HTML and CSS. Add hover effects to images.
Create a webpage that uses different CSS text properties such as text alignment, text decoration, text transformation, and letter spacing.
Design a login page using HTML and CSS. Apply proper styling to input fields and buttons.
Create a webpage using CSS positioning (relative, absolute, fixed). Demonstrate the effect of each positioning method.
Develop a webpage that uses CSS flexbox to align items horizontally and vertically.
Create an HTML page that uses external CSS. Apply styles such as background color, font size, and border.
Design a responsive webpage using CSS media queries for mobile and desktop view.
Create a webpage with a header, footer, and main content area. Use CSS to apply layout, colors, and spacing properly.
Create an HTML page that displays a card layout using div elements. Apply CSS to add shadow effects, border radius, and spacing.
Create an HTML page with a dropdown navigation menu using HTML and CSS
Create a student registration form using HTML that includes text fields, email, password, radio buttons, checkboxes, a dropdown list, and a submit button. Style the form using CSS.
Design a feedback form with name, email, rating (radio buttons), comments (textarea), and submit button. Use CSS to improve layout and appearance.

Design a contact us form with name, email, subject, message, and submit button. Use CSS to add spacing, borders, and background color.

Design a survey form using HTML forms with multiple choice questions, checkboxes, and dropdown lists. Style it neatly using CSS.

Create a login/registration form and write a JavaScript validation program to ensure that:

- The username field is not empty
- The password field is not empty
- The password and confirm password fields match
  Display appropriate error messages and prevent form submission if validation fails.

Create a student login form using HTML. Write a JavaScript program to check that the username and password fields are not left empty. Display suitable error messages and stop the form from submitting if any field is empty.

Create a contact form that includes name and email fields. Write a JavaScript program to check whether both fields are filled and validated where correct syntax of email ID is entered and name is alphabet only before submitting the form.

Design a form that includes a mobile number input field and a Terms and Conditions checkbox. Write a JavaScript program to validate that:

- The mobile number contains exactly 10 digits, and
- The Terms and Conditions checkbox is selected before submitting the form.

Display appropriate error messages and prevent form submission if any validation fails.

Design a Student Registration Form using HTML that contains the following fields:

- Name
- Gender (using radio buttons)
- Hobbies (using checkboxes)
- Course (using a dropdown list)
- Submit button

Write a JavaScript program to perform the following validations when the form is submitted:

1. Ensure that the name field is not left empty.
2. Check that one gender option is selected.
3. Verify that at least one hobby is selected.
4. Ensure that the user selects a course other than the default option from the dropdown list.

Create a login form with:

- Username (text input)
- Password

Validation using JavaScript:

- Both fields must not be empty.
- Password must be at least 8 characters.
  Display error messages if validation fails.

Design a contact form with:

- Name
- Email
- Mobile Number
- Message (textarea)

Validation using JavaScript:

- Name and message cannot be empty.
- Email must be valid (contain @ and .).
- Mobile number must be exactly 10 digits.
  Prevent submission until all validations are correct.

Create a form with:

- Full Name
- Email
- Contact Number
- Gender (radio buttons)
- Events to Participate (checkboxes: Quiz, Debate, Workshop)
- Event Type (dropdown: select one)

Validation using JavaScript:

- Name, email, and contact number must not be empty.
- Email must be valid.
- Contact number must be 10 digits.
- One gender must be selected.
- At least one event must be selected.
- A valid event type must be selected.

Create a form with: • Name • Email • Interests (checkboxes: Technology, Sports, Music, Fashion) Validation using JavaScript: • Name and email cannot be empty. • Email must be valid. • At least one interest must be selected.
Create a form with: • Name • Email • Rating (radio buttons: Poor, Good, Excellent) • Suggestions (textarea) • Subscription (checkbox: Yes, I want updates) Validation using JavaScript: • Name, email, and suggestions must not be empty. • Email must be valid. • One rating must be selected. • Subscription checkbox is optional.

Create a functional component Greeting that accepts a name prop and displays: • "Hello, [name]! Welcome to React." Then create a parent component that passes different names to Greeting and renders three greetings dynamically.
Create a component StudentCard that accepts props: name, rollNumber, and marks. • Display the student information inside a styled card. • Use inline styles or CSS modules. • Render at least three different student cards in the parent component.
Create a Product component that accepts name, price, and inStock props.

• Display product name and price. • If inStock is true, show “Available”, else “Out of Stock”. • Render a list of at least 5 products using the same component.
Create a Parent component that renders a Child component. • Pass at least 4 different props (string, number, boolean, array) to Child. • In Child, display all props properly. • Style the output using CSS.
Create a CustomButton component that accepts a label prop and an onClick prop. • When clicked, it should call the onClick function from the parent. • Parent component should render three buttons with different messages in the console.
Create a TableRow component that takes props: id, name, age, course. • Parent component passes an array of student objects to render multiple rows. • Dynamically create table rows.
Create a Card component that accepts title, subtitle props and children. • Render children inside the card body. • Use the Card component in the parent with different content for each card.
Create a Task component that accepts taskName and completed props. • Display task name. • If completed is true, show the text in green, else red. • Parent component should render a list of tasks using this component.
Create a Profile component that accepts name, age, city props. • Set default props for age and city. • Render the component with and without passing optional props to test default behavior.
Create a Category component that accepts name and items props (items is an array). • Render the category name and a list of items under it.

- Parent should render at least 3 categories with multiple items.

Create a MovieList component that accepts a movies prop (array of objects with title, year, rating).

- Render each movie inside a styled card.
- Display all properties for each movie.

Create a ParentCounter component and a CounterButton component.

- Pass a function from parent to child via props to increment a counter.
- Display updated counter in parent each time child button is clicked.

Create an ImageCard component with props: imageUrl, title, description.

- Parent component should render at least 5 ImageCard components.
- Use flexbox or grid to display cards in a responsive layout.

Create a Message component with props: text, type (success, error, info).

- Apply different CSS styles based on type.
- Render at least three messages of different types in the parent component.

Create a CalculatorButton component that accepts label and onClick props.

- Parent component should pass a function to calculate double, square, and cube of a number.
- Render three buttons, each calling the respective calculation function and showing the result in the parent.

Create a counter component that allows the user to:

- Increment or decrement the count by a dynamic step entered in an input field.
- Reset the count to zero.
- Display an alert if the count goes negative.

Create a signup form with fields: name, email, and password.

- Use state for controlled inputs.
- Display real-time error messages using state.

Create an input field where: • The typed value is shown in real-time below the input. • Use state to store and display the input value.
Create a component with a “Show/Hide” button: • Clicking the button toggles the visibility of some text. • Use state to track whether the text is visible or hidden.
Create a form with name and email fields: • On submitting, display the entered data in an alert box. • Use state to manage form values.
Create a button that increments the number of likes: • Display the current number of likes using state. • Clicking multiple times updates the count immediately.
Build a component with two tabs: Tab 1 and Tab 2: • Clicking a tab displays different content. • Use state to track the active tab.
Create a dropdown with 3 options: • Display the selected option below the dropdown. • Use state to store and display the selection.
Create a text input field that shows: • The number of characters typed in real-time. • Use state to track input length dynamically.
Create a counter where the user can enter a number as a step: • Increment and decrement the count by that step. • Display the current count using state.

Password Show/Hide Toggle

- Create a password input field with a “Show/Hide” button.
- Clicking the button toggles between text and password type.
- Use state to manage the input type.

Simple Rating Component

- Display 5 stars.
- Clicking a star highlights that many stars.
- Use state to track the current rating.

Conditional Rendering Based on Input

- Create an input field for age.
- Display “You are a minor” if age < 18, else display “You are an adult”.
- Use state to track the input value and render dynamically.

Multi-Page Navigation

- Create a 3-page app: Home, About, Contact.
- Use React Router to navigate between pages.
- Highlight the active link in the navigation bar.

Redirect after Action

- Create a Login form.
- On successful login, redirect to /dashboard.
- Use useNavigate to perform the redirect programmatically.

Home and About Page Navigation

Create a React app with Home and About pages.

- Include a navigation menu on both pages.
- Users should be able to move between Home and About easily.

Button-Based Redirect

On the Home page, add a button labeled “Go to About”.

- When clicked, the user should be redirected to the About page.

Passing Data Between Pages

Create two pages: User Input and User Profile.

- Allow the user to enter a name on the first page.
- On the second page, display the entered name.

Create a simple form (e.g., Contact Form).

- After submission, redirect the user to a Thank You page.

Navigate Back to Previous Page

After performing an action (e.g., clicking a “Cancel” button), the user should return to the previous page.

Create a REST API and Display Data

Build an Express REST API that returns a list of users.

- Fetch this data in React and display it in a table.

Create a New Resource

Build an API endpoint to add a new user with fields like name and email.

Update a Resource

Create an API endpoint to update a user’s details.

Delete a Resource

Build an API endpoint to delete a user by ID.

Get User by ID

Create an API endpoint to fetch a single user by ID.

Update Specific Field

Create an API endpoint to update only the email of a user.

Library Book Catalog

Create a system to manage books:

- Add new books (title, author, ISBN)
- Display all books in a list
-

Library Book Catalog

Create a system to manage books:

- Display all books in a list
- Edit book details

Library Book Catalog

Create a system to manage books:

- Display all books in a list
- Delete books

Student Management System – Add Students

Create a system to manage students:

- Add new students (name, age, grade)
- Display all students in the list

Student Management System – Display Students

Create a system to manage students:

- Display all students in a table based on their roll no
- Display all students in the list

Student Management System – Edit Students

Create a system to manage students:

- Edit student information (name, age, grade)
- Display all students in the list

Student Management System – Delete Students

Create a system to manage students:

- Delete students from the list
- Display all students in the list