

## **Exercise 1 – React JS**

### **(Components, Lists, Keys, prop validation and CSS)**

1. Develop a React-JS application using a functional component that renders a list of countries with their capitals using `.map()`. Each list item must have a unique key, and the list should be styled using an external CSS file.
2. Develop a class component in React-JS that displays a list of restaurants, where each restaurant contains a nested list of menu items and apply unique keys for each item.
3. Design and develop a functional component in React-JS that displays a 5 Vehicle Info Card in a row with details like Model, Manufacturer, Year, and Fuel Type. Use inline CSS to style the card layout and appearance.

## **Exercise 2 – React JS**

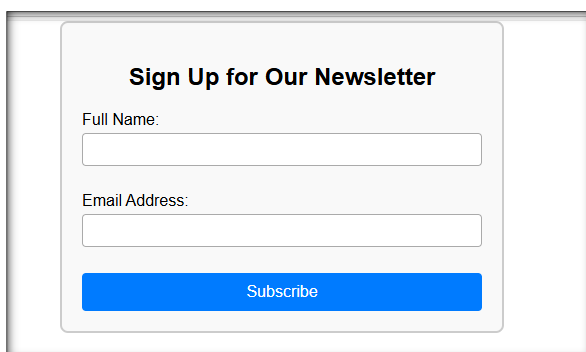
### **(Components, Lists, Keys, prop validation and CSS)**

4. Design a class-based React-JS component called CourseCard that receives props such as course title (string), duration in weeks (number), and instructor name (string). Use prop-types to validate that all props are required and of the correct type.
5. Design and develop a functional component in React-JS that renders a grid of famous landmarks (name, location, country). Use external CSS to apply styles like borders, spacing, and hover effects to each item.

## **Exercise 3 – React JS**

### **(Forms, Events, Conditional Rendering, CSS)**

6. Design and develop a functional component called NewsletterSignup. The form should include fields for full name and email address. Use `useState` to control the form inputs. On form submission, display a thank-you message using conditional rendering. Style the form using an external CSS file with padding, border, and hover effects.

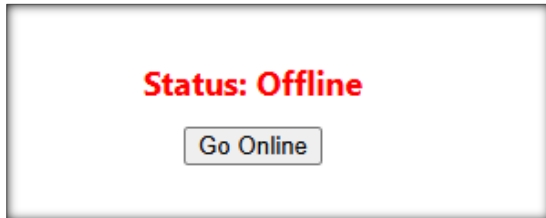


**Sign Up for Our Newsletter**

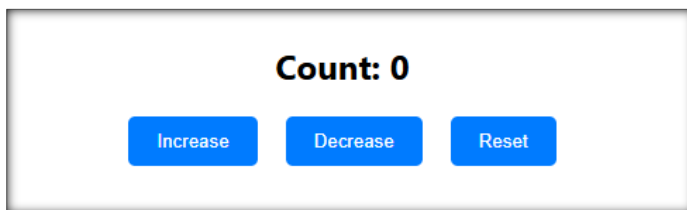
Full Name:

Email Address:

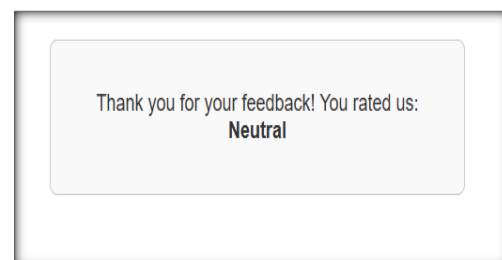
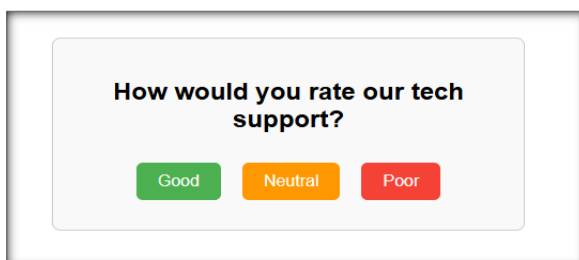
7. Design and develop a functional component named `UserStatusSwitcher` that toggles a user's status between “Online” and “Offline” using a button. Use `useState` to manage status and `onClick` to update it. Display the current status with conditional rendering. Style the status text with inline CSS (e.g., green for online, red for offline).



8. Design and develop a functional component named `Counter` that displays a number starting from 0. Add three buttons labeled "Increase", "Decrease", and "Reset". Clicking "Increase" should increment the number by 1 using `useState`. Clicking "Decrease" should decrement the number by 1. Clicking "Reset" should set the number back to 0. Use inline CSS to center the content on the page, apply padding, margin, and background color to the buttons, and make the displayed number bold with a slightly larger font size.



9. Design and develop a functional component named `FeedbackPoll`. Display a question such as “How would you rate our tech support?” with three buttons: Good, Neutral, Poor. When a user clicks one, use conditional rendering to show a thank-you message including their selected choice. Use external CSS to style the poll area, buttons, and feedback message.



## **Exercise 1 – Express JS**

### **(Routing, Parameters, Query Strings, HTTP Methods, Controllers)**

1. Develop a basic Express.js application that simulates a simple user management system. The application should define a `/users` route that allows interaction through different HTTP methods. Begin by initializing an in-memory array to hold user objects, where each object includes three properties: `id`, `name`, and `email`. Implement functionality to return all users using the GET method on `/users`. Allow new users to be added via the POST method to the same route. Each new user should receive a unique ID that increments sequentially. Additionally, implement the DELETE method on `/users/:id` to remove a specific user by ID. Ensure that the application uses `express.json()` middleware to parse incoming JSON request bodies, and return appropriate JSON responses for each action. Include basic error handling to display a clear message when attempting to delete a user that does not exist.
2. Create an Express.js application that manages a catalog of products. Start with a predefined array of products, each having the properties `id`, `name`, `price`, and `stock`. Implement a PUT route at `/products/:id` that updates an existing product's details. The request body may contain any combination of the three updatable fields: `name`, `price`, and `stock`. Your application should update only the fields provided in the request and leave the rest unchanged. If a product with the given ID is not found, return a 404 status with a message indicating that the product does not exist. Ensure the use of `express.json()` middleware to process JSON bodies and return structured JSON responses that confirm the update and show the latest state of the product. Keep the application implementation in a single JavaScript file, without using a database or external files.
3. Design and implement an Express.js application to manage a list of books. Each book must have an `id`, `title`, and `author`. Initialize the application with at least two predefined books stored in an in-memory array. The application must support the following functionalities:
  - a. The GET `/books` route should return the list of all books.
  - b. The POST `/books` route should accept a new book's title and author in the request body, assign it a unique id, add it to the array, and return a confirmation message.
  - c. The PUT `/books/:id` route should update an existing book's title and/or author based on the id provided in the URL. Only the fields present in the request body should be updated. If the book is not found, return a 404 status with an appropriate message.
  - d. The DELETE `/books/:id` route should remove a book by its id. If the book does not exist, return a 404 error. Otherwise, return a success message indicating which book was deleted.
  - e. Ensure proper use of `express.json()` middleware and return all responses in plain text format using `res.send()`.

## **Exercise 2 – Express JS**

### **(Templating with EJS & Pug, Static Files, Form Handling, Multer)**

4. Design a feedback form for a travel website using Express and multer:
  - The form should collect name, email, and a description of an issue.
  - On submission, display the input using EJS.
5. Build a recipe page using Pug templating:
  - Pass dynamic recipe data (name, ingredients, steps) to the Pug view.
  - Apply external CSS for layout and inline CSS for highlighting ingredients.
  - Include a virtual path for serving images used in the recipe view (/assets).

## **Exercise 3 – Express JS**

### **(Covers Cookies, Sessions, Authentication)**

6. Create a student portal login system using Express.js
  - Create a /register route to add a student with a rollNo, name, and password.
  - Use express-session to store the session after successful login from /login.
  - On login, also set a cookie studentPortalAccess with the student's roll number and an expiry of 3 minutes.
  - Use middleware (cookie-parser and express-session) to manage cookies and sessions.
7. Design a protected route /result that only logged-in students can access
  - Use session middleware to verify if the student is logged in.
  - If valid, show: "Hi [name], your results are available!".
  - If not, return: "Access denied: Please login to view results."
  - Add a /logout route to destroy the session and clear the cookie.
8. Build a course enrollment route /courses with GET and POST
  - Use GET /courses to return a list of available courses (only if logged in).
  - Use POST /courses to enroll the logged-in student into a course.
  - On successful enrollment, create a cookie named lastEnrolledCourse (valid for 2 mins).

## **Exercise 1 – Node JS**

### **(Callbacks, Event Loop and Event Emmitter)**

1. Write a Node.js program that defines a function `add(a, b, callback)` which adds two numbers and returns the result via a callback. Chain this with another callback to multiply the result by 10 and log it. Finally, use `fs.readFile()` to read and display the contents of a file named `info.txt`.
2. Create a countdown timer using `setTimeout()`, use `setTimeout()` and `console.log()` to demonstrate asynchronous behaviour, and add another `setTimeout` with 1000ms execute.
3. Create an event emitter that emits a `greet` event and logs a message, and emit a `login` event with a username and log "`<username>` has logged in".

## **Exercise 2 – Node JS**

### **(Buffers, Streams and File System)**

4. Create a buffer from the string "Node.js" and print it in hexadecimal form, then modify the first letter of the buffer from "N" to "C" and print the result.
5. Use a readable stream to read `data.txt` and log chunks to the console, and explain the benefit of using streams instead of `fs.readFile()`.
6. Write a program to write "Welcome to Node.js" into a file named `welcome.txt`, and read the content of `welcome.txt` and log it using a callback.

## **Exercise 1 - Git**

1. You are starting a personal website project, so initialize a Git repository in a folder named `my-website`, create a file called `index.html` with basic content, check the Git status, stage the file, commit it with the message "Initial commit", and view the commit history.
2. You want to build a new feature without disturbing the main branch, so create a new branch named `feature-navbar`, switch to it, add a file called `navbar.html` with a simple navigation bar, commit the changes with a suitable message, switch back to the main branch, and merge the `feature-navbar` branch into it.

## Extra Questions for Practice

1. Design a React class component with prop validation using PropTypes to ensure correct data types for props?

### Simple ReactJS Props validation example

Type	Value	Valid
Array	1, 2, 3, 4, 5	true
Boolean	False	true
Function	50	true
String	GFG	true
Number	100	true

### Simple ReactJS Props validation example

Type	Value	Valid
Array	1, 2, 3, 4, 5	true
Boolean	False	true
Function	50	true
String	GFG	true
Number	100	true

2. Design and develop a functional component called TechBugReportForm. The form should include the following fields:

- Bug Title (text)
- Description (textarea)
- Affected Module (dropdown: e.g., UI, API, Database, Network)

Validate that all fields are filled before submission. Use conditional rendering to show inline error messages if any field is empty and display a submission success message otherwise. Apply external CSS to organize the form and highlight input errors.

### Tech Bug Report

Bug Title:

Bug title is required.

Description:

Description is required.

Affected Module:

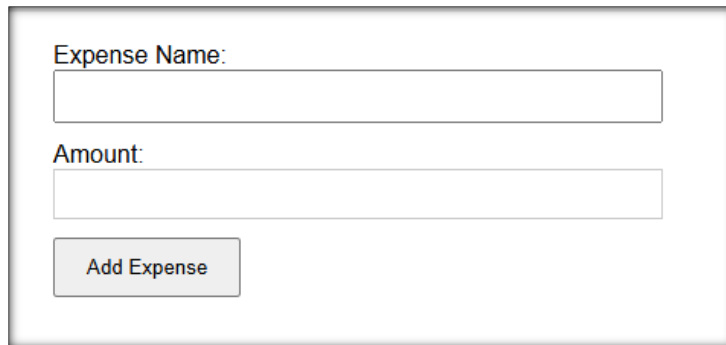
-- Select Module --

Please select a module.

Submit

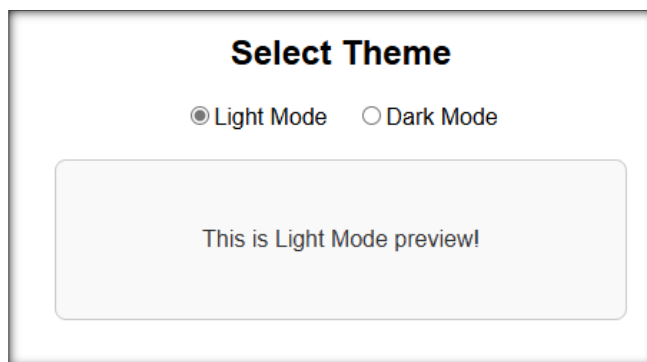
3. Design and develop a functional component called ExpenseTrackerInput. Include inputs for expense name and amount. Use useState for form control. Validate that the amount is a positive

number. On form submission, show a success message or an error message using conditional rendering. Use inline CSS to highlight the amount field in red if validation fails.



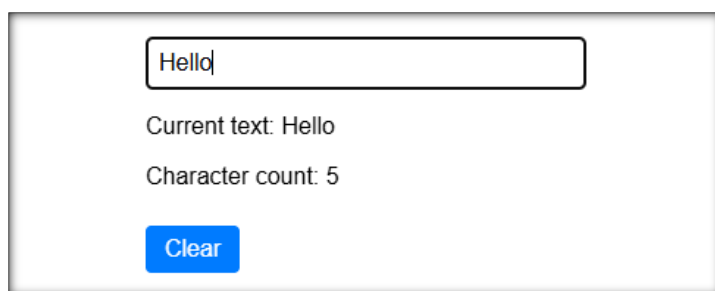
A form for adding an expense. It has two input fields: "Expense Name:" and "Amount:". Below the "Amount:" field is a button labeled "Add Expense".

4. Design and develop a functional component named `ThemeSelector`. Provide two radio buttons: Light Mode and Dark Mode. Use `useState` to track the selected theme. Conditionally render a preview box styled with the appropriate theme (dark or light). Use external CSS classes to apply theme-based background and text styling.



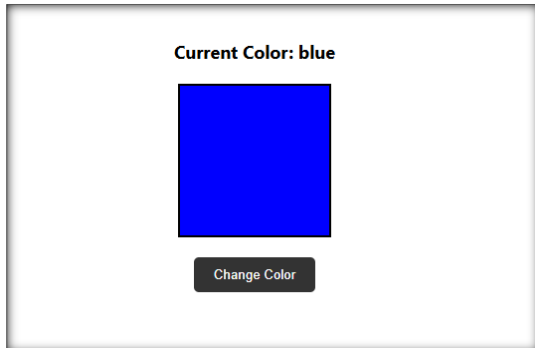
A "Select Theme" component. It has two radio buttons: "Light Mode" (selected) and "Dark Mode". Below the radio buttons is a preview box with the text "This is Light Mode preview!".

5. Design and develop a React functional component called `TextInputTracker` with a controlled text input. Display the current input text below the input as the user types, along with a character count. Use `useState` to track the input value. Restrict input to only letters and spaces—ignore any other characters. Include a clear button that resets the input and display. Keep the styling minimal: just some spacing and basic font styles to keep it clean and readable.



A `TextInputTracker` component. It has a controlled text input with the value "Hello". Below the input, it displays "Current text: Hello" and "Character count: 5". At the bottom is a blue "Clear" button.

6. Design and develop a functional component named `ColorChanger` with a button that cycles through a list of colors (`["red", "green", "blue"]`) each time it's clicked. Use `useState` to manage the current color. Display a `div` with a fixed height and width, and apply the current color as its background using inline CSS. Also, include a label above the `div` showing the current color name.



7. Create an Express.js app using EJS that renders a movie gallery:
- Render a list of movies with titles and ratings.
  - Use a layout file with a header/footer and external CSS to style the cards.
  - Highlight movies with a rating above 8 in a different color using conditional rendering.

