MERN Stack Training

Weekly Tasks

Week 3 & 4:

1. **Recursion and stack**:

    - Task 1: Implement a function to calculate the factorial of a number using recursion.
    - Task 2: Write a recursive function to find the nth Fibonacci number.
    - Task 3: Create a function to determine the total number of ways one can climb a staircase with 1, 2, or 3 steps at a time using recursion.
    - Task 4: Write a recursive function to flatten a nested array structure.
    - Task 5: Implement the recursive Tower of Hanoi solution.

2. **JSON and variable length arguments/spread syntax**:

    - Task 1: Write a function that takes an arbitrary number of arguments and returns their sum.
    - Task 2: Modify a function to accept an array of numbers and return their sum using the spread syntax.
    - Task 3: Create a deep clone of an object using JSON methods.
    - Task 4: Write a function that returns a new object, merging two provided objects using the spread syntax.
    - Task 5: Serialize a JavaScript object into a JSON string and then parse it back into an object.

3. **Closure**:

    - Task 1: Create a function that returns another function, capturing a local variable.
    - Task 2: Implement a basic counter function using closure, allowing incrementing and displaying the current count.
    - Task 3: Write a function to create multiple counters, each with its own separate count.
    - Task 4: Use closures to create private variables within a function.
    - Task 5: Build a function factory that generates functions based on some input using closures.

4. **Promise, Promises chaining**:

- Task 1: Create a new promise that resolves after a set number of seconds and returns a greeting.
- Task 2: Fetch data from an API using promises, and then chain another promise to process this data.
- Task 3: Create a promise that either resolves or rejects based on a random number.
- Task 4: Use Promise.all to fetch multiple resources in parallel from an API.
- Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

5. **Async/await**:

   - Task 1: Rewrite a promise-based function using async/await.
   - Task 2: Create an async function that fetches data from an API and processes it.
   - Task 3: Implement error handling in an async function using try/catch.
   - Task 4: Use async/await in combination with Promise.all.
   - Task 5: Create an async function that waits for multiple asynchronous operations to complete before proceeding.

6. **Modules introduction, Export and Import**:

   - Task 1: Create a module that exports a function, a class, and a variable.
   - Task 2: Import the module in another JavaScript file and use the exported entities.
   - Task 3: Use named exports to export multiple functions from a module.
   - Task 4: Use named imports to import specific functions from a module.
   - Task 5: Use default export and import for a primary function of a module.

7. **Browser: DOM Basics**:

   - Task 1: Select an HTML element by its ID and change its content using JavaScript.
   - Task 2: Attach an event listener to a button, making it perform an action when clicked.
   - Task 3: Create a new HTML element and append it to the DOM.
   - Task 4: Implement a function to toggle the visibility of an element.
   - Task 5: Use the DOM API to retrieve and modify the attributes of an element.

Mini Project: **"Task Scheduler"**

Objective:
Develop a web-based task management application where users can add, delete, modify, and

save tasks to local storage. The application will incorporate asynchronous programming, DOM operations, and JSON serialization for storing and retrieving tasks.

---

1. **Task Object Definition**:
    - **Properties**: taskName, dueDate, priority, completed
    - **Methods**:
        - **getTaskDetail**: Returns task details in a string format.
        - **toggleCompletion**: Toggles the completion status of the task.
2. **Tasks Collection (Array)**:
    - Initialize an empty array taskList to store task objects.
3. **Recursion**:
    - Implement a function to display tasks in a hierarchical or nested manner if tasks are dependent on one another.
4. **Array Methods & Spread Syntax**:
    - **Functions**:
        - **addTask(task...)**: Adds multiple tasks to the taskList using push().
        - **deleteLastTask()**: Deletes the last task using pop().
        - **addTaskToFront(task...)**: Adds tasks to the beginning using unshift().
        - **deleteFirstTask()**: Deletes the first task using shift().
5. **Closure & Modules**:
    - Each task operation (like add, delete, update) can be a separate module. A counter for the total number of tasks can use closures to ensure privacy.
6. **Promise, Async/Await & JSON**:
    - **Functions**:

        - **saveTasks()**: Emulates saving of tasks to local storage. Use JSON.stringify to serialize the taskList before storing.
        - **loadTasks()**: Emulates loading tasks from local storage using async/await. Deserialize using JSON.parse.

7. **DOM Basics**:

    - Create an HTML structure for task display and manipulation.
    - **Functions**:
        - **renderTasks()**: Displays all tasks on the page using DOM operations.
        - **addTaskUI()**: Provides task input fields and buttons to add tasks.
        - **deleteTaskUI(taskName)**: Deletes the task with the specified name from the UI.

Project Workflow:

1. **Initialization**:

    - Start by defining the task object.
    - Initialize the taskList array.

2. **Task Operations**:

    - Implement array methods to manipulate tasks in the array.

3. **DOM Integration**:

    - Design the HTML structure to display and manage tasks.
    - Integrate task functions to respond to button presses and other UI events.

4. **Asynchronous & JSON Operations**:

    - Implement the save and load functions.
    - Serialize the taskList into JSON when saving and parse JSON data when loading tasks from local storage.
    - Connect these functions to appropriate buttons or events (e.g., page load).

5. **Enhancements**:

    - Improve the user interface.
    - Add error handling for async operations and handle potential JSON parsing errors.
    - Modularize the code for better maintainability.

Bonus: Add filtering and sorting functionalities for tasks. Allow users to filter tasks based on priority, completion status, or due date. Sorting tasks by due date or priority can also enhance usability.