1 Assignment: Interactive Web Dashboard

1.1 Task 1: Persistent Task Manager

Goal: Create a simple to-do list application that saves tasks in the browser so they don't disappear when the page is reloaded.

Starter HTML (add this to your `index.html` file):

```
<div id="task-manager">
    <h2>My Tasks</h2>
    <input type="text" id="task-input" placeholder="Enter a new task..." />
    <input type="date" id="task-due-date" />
    <button id="add-task-btn">Add Task</button>
    <ul id="task-list"></ul>
</div>
```

1.1.1 Instructions:

1. **Create a Task Class:**
   - In your `script.js` file, define a class named `Task`.
   - The `constructor` should accept a `description` and a `dueDate`. It should also initialize a property `isCompleted` to `false`.
2. **Add a New Task:**
   - When the "Add Task" button is clicked, get the values from the text input and the date input.
   - Create a new instance of your `Task` class with these values.
   - Store all your task objects in a global array (e.g., `var tasks = [];`).
3. **Display Tasks:**
   - Create a function called `renderTasks()`. This function should:
     - Clear the current content of the `<ul id="task-list">`.
     - Loop through your `tasks` array.
     - For each task object, create an `<li>` element. The `<li>` should display the task's description and its due date.
     - If a task's `isCompleted` property is `true`, add a CSS class to the `<li>` that applies a `text-decoration: line-through`.
     - Append the `<li>` to the task list `<ul>`.
4. **Local Storage Persistence:**
   - Create two functions: `saveTasks()` and `loadTasks()`.
   - `saveTasks()`: This function should convert the `tasks` array into a JSON string using `JSON.stringify()` and save it to `localStorage` under a key like `'myTasks'`.
   - `loadTasks()`: This function should be called when the page first loads. It should get the JSON string from `localStorage`, parse it back into an array using `JSON.parse()`, and then call `renderTasks()` to display them.

> **Important**: The objects parsed from JSON won't be instances of your `Task` class. You'll need to re-instantiate them.

5. **Toggling Completion:**
   - Modify your `renderTasks()` function so that each `<li>` is clickable.
   - When an `<li>` is clicked, find the corresponding task object in your `tasks` array, toggle its `isCompleted` property (from `true` to `false` or vice-versa), re-save the tasks to `localStorage`, and re-render the list.

1.2 Task 2: Asynchronous User Profile Card

Goal: Get user data from a public API and display it. The data should be cached for the current session to avoid unnecessary API calls.

Starter HTML (add this to your `index.html` file):

```
<div id="profile-section">
    <h2>User Profile</h2>
    <button id="get-user-btn">Get Random User</button>
    <div id="profile-card"></div>
</div>
```

1.2.1 Instructions

1. **Get User Data:**
   - Add an event listener to the "Get Random User" button.
   - When clicked, use `XMLHttpRequest` to make a `GET` request to the API endpoint: `https://jsonplaceholder.typicode.com/users/ID`.
     - Replace `ID` with a random user ID (e.g., 1, 2, ..., 10).
2. **Handle the Response:**
   - Inside your `XMLHttpRequest`'s `onload` event handler, check if the request was successful.
   - If successful, parse the JSON response text into a JavaScript object.
   - Store this user object as a JSON string in `sessionStorage` under a key like `'currentUser'`.
   - Call a function `displayUser()` and pass the parsed user object to it.
3. **Display User Data with `for...in`:**
   - Create the `displayUser(userObject)` function.
   - This function should clear any existing content inside the `<div id="profile-card">`.
   - Use a `for...in` loop to iterate over all the properties of the `userObject`.
   - For each `key` in the object, create and append a `<p>` element to the profile card. The paragraph should be formatted like: `<strong>key:</strong> value`.
   - **Note:** Skip displaying properties that are themselves objects (like the `address` and `company` properties in the API response).
4. **Implement Session Caching:**

- Write a function that runs when the page first loads.
- This function should check if a user object already exists in `sessionStorage`.
- If it does, parse it and call `displayUser()` immediately, so the user data appears without needing to click the button.

1.3 Task 3: Timers

Goal: Use higher-order functions to manipulate data and use BOM methods to create timed events.

Starter Code (add this to your `script.js` file):

```
const transactions = [
    { id: "t1", type: "debit", amount: 150 },
    { id: "t2", type: "credit", amount: 200 },
    { id: "t3", type: "debit", amount: 50 },
    { id: "t4", type: "credit", amount: 300 },
];
```

Starter HTML (add this to your `index.html` file):

```
<div id="timed-events">
    <h2>Notifications & Timer</h2>
    <p>A special welcome message will appear in 3 seconds!</p>
    <div id="timer-display">10</div>
    <button id="start-timer-btn">Start Countdown</button>
</div>
```

1.3.1 Instructions:

1. **Higher-Order Functions:**
   - Using the `transactions` array provided:
     - Use the `filter()` method to create a new array containing only the 'credit' transactions.
     - Use the `map()` method to create a new array of just the amounts from all transactions (e.g., `[150, 200, 50, 300]`).
     - Use the `reduce()` method to calculate the total amount of all 'debit' transactions.
   - Log the results of all three operations to the console.
2. **One-Time Notification:**
   - Use `setTimeout` to display a browser `alert()` with the message "Welcome to your dashboard!" exactly 3 seconds after the page loads.
3. **Countdown Timer:**
   - Add a click event listener to the "Start Countdown" button.
   - When clicked, it should use `setInterval` to decrease the number in the `<div id="timer-display">` by 1 every second, starting from 10.
   - When the countdown reaches 0, the text should change to "Done!", and you must stop the interval from running further (hint: use `clearInterval`).

4. **Saving User Preferences with Cookies:**
    - Create a simple function `setTheme(themeName)`. This function should set the `document.body.style.backgroundColor`. For `dark` theme use `#333` and for `light` theme use `#FFF`.
    - Create another function `saveThemePreference(themeName)` that saves the chosen theme to `document.cookie`. For example: `"theme=dark"`.
    - Create a function `loadThemePreference()` that runs on page load. It should read `document.cookie`, find the theme preference, and call `setTheme()` to apply it.
    - Add two buttons to your HTML, "Dark Mode" and "Light Mode," which call both `setTheme()` and `saveThemePreference()` when clicked.

1.4 Deliverables

- `index.html`: An HTML file containing the structure for all three parts.
- `task-1.js`, `task-2.js`, `task-3.js`: A JavaScript file for each task containing the required functionality.