AUTHORITY ENTREPRENEURS

# Javascript Developer Task:

Task: Build a Single Page Application (SPA) with a Simulated RESTful API Using JavaScript Only

**Objective**
Develop a fully functional Single Page Application (SPA) that performs CRUD (Create, Read, Update, Delete) operations on a list of tasks using only vanilla JavaScript. The application should simulate a RESTful API using JavaScript objects and local storage to store data.

**Requirements**

**1. Project Overview**
- Frontend: Build the SPA using vanilla JavaScript (no libraries or frameworks).
- Backend Simulation: Use JavaScript objects and arrays to simulate RESTful API behavior. Store the data in the browser's `localStorage` to persist it across page reloads.

**2. Core Features of the SPA**
- Task List Page: Display a list of tasks fetched from the simulated RESTful API.
- Add Task: A form to create a new task with fields for `title`, `description`, and `due date`.
- Edit Task: Ability to update an existing task directly from the SPA interface.
- Delete Task: Option to delete a task from the list.
- Task Detail View: Display a detailed view of a task when a user clicks on a task from the list.
- Search/Filter Tasks: Feature to search or filter tasks based on status, due date, or keyword.

**3. Technical Requirements**

- Frontend (SPA):
  - Use modern JavaScript (ES6+ features) such as `const`/`let`, template literals, arrow functions, and the Fetch API for simulated API requests.
  - Implement routing using the browser's History API (`pushState` and `popstate` events) to manage different views within the SPA (e.g., Task List, Task Details, Add/Edit Task forms).
  - Use JavaScript to dynamically update the DOM without page reloads.
  - Implement form validation using JavaScript before storing or updating data.
  - Use CSS for styling and ensure the application is fully responsive and works across all modern browsers.
  - Implement client-side error handling and user-friendly notifications (e.g., using `alert` or custom modals).

- Simulated RESTful API:

- Data Storage: Use JavaScript objects and arrays to represent the data model (e.g., an array of task objects).
- CRUD Operations:
  - `GET /tasks`: Simulate fetching all tasks by reading from the JavaScript array or local storage.
  - `POST /tasks`: Simulate creating a new task by pushing a new task object to the array and saving it to local storage.
  - `GET /tasks/:id`: Simulate fetching a single task by ID by finding the task object in the array.
  - `PUT /tasks/:id`: Simulate updating a task by modifying the task object in the array and updating local storage.
  - `DELETE /tasks/:id`: Simulate deleting a task by removing the task object from the array and updating local storage.
- Data Persistence: Use the browser's `localStorage` to persist data between page reloads. Store the tasks as a JSON string and parse it back into JavaScript objects when needed.

**4. Additional Features (Bonus)**
- Task Prioritization: Add the ability to prioritize tasks (e.g., High, Medium, Low) and filter tasks by priority.
- Drag-and-Drop Interface: Implement a drag-and-drop interface to reorder tasks within the list.
- Task Reminders: Add a feature to set reminders for tasks and display notifications using the browser's `Notification` API.

**5. Testing and Validation**
- Test the application across different browsers to ensure compatibility.
- Ensure that all CRUD operations work correctly and data is properly persisted in `localStorage`.
- Validate that the routing system correctly handles back and forward navigation without page reloads.

Deliverables

- A zip file containing the complete source code for the SPA.
- A README file with instructions on how to run the application locally.
- A short write-up and VIDEO within the code explaining key parts of the implementation.