Prerequisites for Developing Task Management with Categories

To develop the provided task management system effectively, the following technical prerequisites should be fulfilled:

---

1. Technical Skills

JavaScript Concepts

1. **Data Management**:
    - Object manipulation (`Object.assign`) to handle categories.
    - Array operations (`push`, `splice`, `map`, `forEach`).
2. **DOM Manipulation**:
    - Dynamically updating DOM elements using `innerHTML`.
    - Accessing DOM elements via `getElementById` and adding event listeners.
3. **LocalStorage API**:
    - Saving and retrieving data using `localStorage.setItem` and `localStorage.getItem`.
4. **Async Programming**:
    - Fetching tasks from an external API using `fetch` and `async/await`.
    - Handling API responses and errors gracefully.
5. **Event Handling**:
    - Listening for button clicks to add tasks or move them between categories.

---

2. Functional Requirements

Task Management

- **Categories**:
    - Predefined categories: `pending`, `inProgress`, and `completed`.
    - Tasks can be dynamically moved between categories.
- **Task Addition**:

    - Allow users to add a new task to the `pending` category.
- **Task Movement**:
    - Move tasks from:
        - `pending` → `inProgress`.
        - `inProgress` → `completed`.

Persistent Storage

- Save tasks to LocalStorage whenever they are added or moved.
- Load tasks from LocalStorage during initialization.

Dynamic Rendering

- Render tasks dynamically for each category:
    - `pending` tasks with a "Start" button.
    - `inProgress` tasks with a "Complete" button.
    - `completed` tasks as static items.

API Integration

- Fetch tasks from an external API (`https://jsonplaceholder.typicode.com/todos`) and add them to the `pending` category.

Error Handling

- Handle invalid inputs (e.g., empty task names).
- Handle errors during API fetching with meaningful error messages.

---

3. Frontend Design Requirements

UI Layout

- **Input Field**: For adding new tasks.
- **Category Containers**:
    - Separate sections for `pending`, `inProgress`, and `completed` tasks.
- **Buttons**:
    - "Start" and "Complete" buttons for task transitions.

Styling

- Basic CSS or frameworks like **Tailwind CSS** for:

- Spacing (`p-2`, `bg-gray-100`, etc.).
- Flexbox alignment (`flex`, `justify-between`).
- Button styling (`text-blue-500`, `text-green-500`).

---

4. Code Structure Requirements

Encapsulation

- Modularize functionality:
  - LocalStorage operations (`saveToLocalStorage`, `loadFromLocalStorage`).
  - DOM rendering (`renderTasks`).
  - Task transitions (`moveTask`).

Reusable Components

- Separate rendering logic for each category.
- Create a utility function for updating the UI dynamically.

Separation of Concerns

- Keep business logic (task management) separate from rendering logic.

---

5. Development Workflow

1. **Setup Task Categories**:
   - Define `categories` with `pending`, `inProgress`, and `completed`.
2. **LocalStorage Integration**:
   - Implement functions to save and load tasks from LocalStorage.
3. **Dynamic Rendering**:
   - Write `renderTasks` to display tasks dynamically for each category.
4. **Task Operations**:
   - Implement task addition and movement logic.
5. **Fetch Tasks from API**:
   - Retrieve tasks from `jsonplaceholder.typicode.com` and populate the `pending` category.

6. **Test & Debug**:
    - Verify:
        - Tasks are correctly added and moved between categories.
        - State is persisted in LocalStorage.
        - External tasks are fetched and displayed correctly.

---

6. Testing Scenarios

Functional Tests

- Add a new task and verify it appears in the `pending` category.
- Move tasks between categories and ensure they are correctly updated.
- Fetch tasks from the API and display them dynamically.

Edge Case Handling

- Handle empty input when adding tasks.
- Prevent duplicate task names (if required).

Error Handling

- Test API failure scenarios (e.g., network errors).
- Ensure LocalStorage operations do not throw errors.

By meeting these prerequisites and following the development workflow, you can build a robust and scalable task management system with dynamic categories. Let me know if you need help implementing specific features!