Prerequisites for Developing the Expense Tracker

Here's a breakdown of the technical and functional prerequisites to develop the expense tracker application:

---

1. Core JavaScript Skills

Data Management

1. **Arrays**:
    - Knowledge of array operations:
        - `map`: To create proxies for expenses.
        - `filter`: For deleting expenses by `id`.
        - `reduce`: To calculate the total expenses.
2. **Proxies**:
    - Understanding of the `Proxy` object to validate expense properties dynamically.
    - Example: Prevent negative or invalid expense amounts.
3. **DOM Manipulation**:
    - Ability to dynamically create and update DOM elements using:
        - `document.createElement`.
        - `innerHTML` for rendering expense items.
4. **Event Handling**:
    - Adding event listeners for user actions like:
        - Adding a new expense.
        - Deleting an existing expense.
    - Listening for `DOMContentLoaded` to initialize the app.
5. **LocalStorage**:
    - Familiarity with `localStorage` to persist expenses data:
        - `setItem`: Save expenses to `localStorage`.
        - `getItem`: Load expenses from `localStorage` on page load.
6. **Validation**:
    - Ensuring user inputs are valid:
        - Non-empty description.
        - Valid and non-negative expense amounts.

---

2. Frontend Development Skills

HTML

- Create the structure of the application:
    - Input fields for `description`, `amount`, and `category`.
    - A button to add expenses.
    - A container to display the list of expenses dynamically.
    - A section to display the total expenses.

CSS

- Style the application using:
    - Layout techniques (`flexbox`, `grid`) for arranging expense items.
    - Button styles (`hover`, `rounded`, `background-color`).
    - Responsive design to make the app mobile-friendly.

Frameworks (Optional)

- Use **Tailwind CSS** or similar utility frameworks for faster styling.

---

3. Backend/API Knowledge

API Integration

- Basic knowledge of REST APIs:
    - Use `fetch` to retrieve expense data from the backend (if applicable).
    - Handle API errors gracefully using `try-catch`.

---

4. Tools and Environment

Development Tools

- **Code Editor**: VS Code, Sublime Text, or similar.
- **Browser**: Chrome/Firefox with DevTools for debugging.

Version Control

- Use Git/GitHub for:

- Version control of the codebase.
- Collaborative development.

---

5. Functional Requirements

Core Features

1. **Add Expense**:
   - Input fields for expense description, amount, and category.
   - Validate inputs before adding the expense.
2. **Render Expenses**:
   - Dynamically display the list of expenses with:
     - Description.
     - Category.
     - Amount.
   - Include a delete button for each expense.
3. **Calculate Total**:
   - Sum up the `amount` field of all expenses and display it dynamically.
4. **Delete Expense**:
   - Remove an expense from the list using its `id`.
5. **Data Persistence**:
   - Save the `expenses` array to `localStorage`.
   - Load and render expenses from `localStorage` on page load.

---

6. Testing Scenarios

Functional Tests

1. **Add Expense**:
   - Add an expense and verify it appears in the list.
   - Test with invalid inputs (e.g., empty description or negative amount).
2. **Delete Expense**:
   - Remove an expense and ensure it disappears from the list and `localStorage`.

3. **Calculate Total**:
    - Verify the total updates correctly after adding or deleting expenses.
4. **Load Expenses**:
    - Reload the page and confirm that the saved expenses are restored from `localStorage`.

Edge Cases

1. Adding an expense with an empty description or invalid amount.
2. Attempting to delete an expense that doesn't exist.
3. Handling empty or corrupted `localStorage` data.

---

7. Enhancements (Optional)

1. **Search Functionality**:
    - Add a search bar to filter expenses by description or category.
2. **Sorting**:
    - Allow sorting expenses by amount, description, or category.
3. **Date Field**:
    - Add a date field to track when the expense was made.
4. **Analytics**:
    - Display breakdowns of expenses by category.
5. **Pagination**:
    - Handle large numbers of expenses by paginating the list.

Development Workflow

1. **Setup Basic HTML Structure**:
    - Input fields for `description`, `amount`, and `category`.
    - Buttons for adding expenses and clearing all expenses.
    - Containers for rendering expenses and displaying totals.

2. **Implement Core JavaScript Logic**:
    - Initialize an `expenses` array.
    - Implement functions for adding, deleting, and rendering expenses.
3. **LocalStorage Integration**:
    - Save expenses to `localStorage` after every update.
    - Load and restore expenses on page load.
4. **Test and Debug**:
    - Verify the app handles all actions correctly and gracefully handles errors.