Classwork: Building a Dynamic Category and Subcategory Viewer with Fetch

Objective

Students will create an interactive Category Explorer that dynamically fetches categories and subcategories from a REST API. They will implement functionality to display, filter, and interact with categories and their nested subcategories using JavaScript's fetch API.

Instructions

1. HTML Page Setup

Create a simple HTML structure with the following components:

- **Heading**: "Category Explorer"
- A **dropdown** (<select> element) to display the categories.
- An **unordered list** (<ul>) to display subcategories for the selected category.
- Optional: A button or input field for adding new categories or subcategories.

---

2. Fetching Categories

1. **API Endpoint**:
   Use the /api/categories/ endpoint to fetch the list of categories.

Example API Response for Categories:

json

code

```
[
  { "id": 1, "name": "Electronics" },
  { "id": 2, "name": "Fashion" },
  { "id": 3, "name": "Books" }
]
```

2. **Populating Dropdown**:
   Populate the <select> element with the fetched categories.

---

3. Fetching Subcategories

1. **When a Category is Selected**:
   After a category is selected from the dropdown, make a fetch request to the /api/categories/{id}/subcategories/ endpoint.
2. **API Endpoint for Subcategories**:
   /api/categories/{id}/subcategories/

Example API Response for Subcategories:

json

code

[

{ "id": 1, "name": "Smartphones" },

{ "id": 2, "name": "Laptops" }

]

3. **Displaying Subcategories**:
   Populate the <ul> element with the subcategories fetched from the API. Each subcategory should be displayed as a list item (<li>).

---

4. Displaying Subcategories

- For each subcategory, display:
  - Name
  - Optionally, a "View Details" button that fetches additional data about the selected subcategory, such as descriptions or images.

---

5. Fetching Nested Subcategories

1. **When a Subcategory is Selected**:
   If a subcategory has nested subcategories, clicking it should trigger another fetch request to /api/categories/{category_id}/subcategories/{subcategory_id}/nested/.
2. **Example API Response for Nested Subcategories**:

json

code

[

{ "id": 1, "name": "Android Phones" },

{ "id": 2, "name": "iOS Phones" }

]

---

6. Error Handling

- Implement error handling for both the main category fetch and subcategory fetch.
- Display a user-friendly message if data cannot be loaded, such as:
  - "Failed to load categories. Please try again later."
  - "No subcategories available."

---

7. Adding New Categories/Subcategories

1. **Add New Category**:
   - Provide a form for adding a new category (name field).
   - POST request to /api/categories/ to create a new category.

2. **Add New Subcategory**:
    - After selecting a category, provide a form for adding subcategories (name field).
    - POST request to /api/categories/{id}/subcategories/.

---

8. Editing and Deleting Categories/Subcategories

1. **Editing**:
    - Provide an "Edit" button for categories/subcategories to populate and update existing data.
    - PUT request to /api/categories/{id}/ or /api/categories/{id}/subcategories/{sub_id}/.
2. **Deleting**:
    - Provide a "Delete" button to remove categories/subcategories.
    - DELETE request to /api/categories/{id}/ or /api/categories/{id}/subcategories/{sub_id}/.

Example Walkthrough

1. **Step 1**: Fetch and display categories from /api/categories/.
2. **Step 2**: Select a category (e.g., "Electronics") and fetch its subcategories from /api/categories/1/subcategories/.
3. **Step 3**: Display subcategories in a list and allow nested fetching for deeper levels using nested subcategories (/api/categories/1/subcategories/2/nested/).
4. **Bonus**: Add forms to create new categories or subcategories, and implement edit/delete functionality.

Deliverables

- A fully functional dynamic Category and Subcategory Viewer.
- Users should be able to view categories, load subcategories, interact with nested levels, and handle CRUD operations for both categories and subcategories.

Learning Outcomes

- Practice working with nested API endpoints and handling hierarchical data.
- Gain hands-on experience with JavaScript's fetch API for multiple asynchronous requests.
- Enhance error handling and user experience in web applications.