Lab Assignment: Mini Blog — CRUD with Modern JavaScript

Objective

Build a small single-page "Mini Blog" web application using **modern JavaScript (ES6+)** that performs **Create, Read, Update, and Delete (CRUD)** operations on a public REST API.

You will use:

- **`fetch`** with **`async/await`** for API calls
- Modern JS features: template literals, destructuring, optional chaining, modules, arrow functions, `AbortController`
- A free **fake REST API**: JSONPlaceholder

By completing this lab, you'll gain **hands-on experience** in integrating modern JavaScript with a REST API and building interactive HTML UIs.

Scenario

You are hired to create a simple **"Mini Blog" dashboard** for managing blog posts. The app should:

- Display a paginated list of posts (title + snippet of body)
- Allow creating new posts
- Allow editing existing posts
- Allow deleting posts
- Update the UI dynamically without refreshing the page

Requirements

1. Setup

- Create a folder for your project.
- Create `index.html` for HTML + Bootstrap 5 styling.
- Create a `main.js` file (use `type="module"` in HTML).
- Link Bootstrap CSS via CDN.
- Link your JavaScript file at the bottom of `body`.

---

2. API to use

You will call the **JSONPlaceholder** API:

Operation	Method	Endpoint	Notes
Get all posts	GET	`/posts?_page=1&_limit=10`	Supports pagination via query params.
Get single post	GET	`/posts/{id}`	
Create post	POST	`/posts`	Send JSON body `{ title, body, userId }`
Update post	PUT	`/posts/{id}`	
Delete post	DELETE	`/posts/{id}`	

Base URL:

```
arduino
CopyEdit
https://jsonplaceholder.typicode.com
```

---

3. UI Layout

Your HTML page must have:

- **Left panel (form)**
  - "Create Post" form with:
    - Title (text input)
    - Body (textarea)
    - Save button
    - Cancel button
- **Right panel (list)**
  - List of posts (Bootstrap cards)
  - Each card shows:
    - Post title
    - Post body snippet
    - Edit button
    - Delete button
  - Pagination controls: "Previous", "Next", and current page number
  - Refresh button

---

4. JavaScript Features to Implement

A. GET posts list (Read)

- On page load, fetch first 10 posts from API.
- Render them as Bootstrap cards in the right panel.

- Add pagination buttons to navigate pages.
- Show a "Loading..." message while fetching.

B. GET single post (Read for editing)

- When clicking **Edit** on a card:
  - Fetch that post by ID
  - Populate the form with its title & body
  - Change form heading to "Edit Post #ID"
  - Change Save button text to "Update"

C. Create post (POST)

- When clicking **Save** in Create mode:
  - Send POST request with `{ title, body, userId: 1 }`
  - Show success/failure message
  - Prepend the created post to the list without reloading the page

D. Update post (PUT)

- When clicking **Update** in Edit mode:
  - Send PUT request to `/posts/{id}`
  - Show success/failure message
  - Refresh current page's posts list

E. Delete post (DELETE)

- When clicking **Delete**:
  - Ask for confirmation
  - Send DELETE request
  - Remove the card from the UI immediately (optimistic update)
  - Show success/failure message

F. Bonus: AbortController

- If the user clicks "Refresh" or changes pages while a fetch is in progress, cancel the previous request.

---

5. Modern JavaScript Practices

You **must**:

- Use **`async/await`** with `fetch`
- Use **template literals** for HTML creation
- Use **destructuring** when extracting data
- Use **arrow functions** for callbacks
- Use **event delegation** for handling Edit/Delete

- Use **optional chaining** (`obj?.prop`) and **nullish coalescing** (`??`) when accessing API data

---

6. Steps to Follow

Step 1: Build HTML structure (Bootstrap grid: left form + right list).
Step 2: Create `apiFetch` helper function that:

- Accepts endpoint and options
- Automatically sets JSON headers when sending a body
- Handles JSON parsing and error messages

Step 3: Implement `getPosts(page)` function to:

- Call `/posts?_page={page}&_limit=10`
- Render results

Step 4: Implement form submission handler:

- If no `id`, call POST
- If `id` exists, call PUT
- Reset form after success

Step 5: Implement Edit and Delete buttons using event delegation.

Step 6: Implement pagination and refresh buttons.

Step 7 (Optional): Add a loading spinner and success/error Bootstrap alerts.

---

7. Deliverables

At the end of the lab, submit:

- `index.html`
- `main.js`
- A short README describing how your app works

---

8. Extra Challenges

- Add a search box to filter posts by title.
- Use `PATCH` instead of `PUT` for partial updates.
- Implement local caching so navigating back to a page doesn't re-fetch.
- Style created posts differently (e.g., with a "New" badge).