# Week 04 — JS Advanced 2 — Lý thuyết (DOM, Events, Fetch, Storage)

## 1) DOM cơ bản: chọn phần tử, tạo/xoá, class/attr/dataset, style

• Chọn: querySelector / querySelectorAll (CSS selector). Tránh getElementBy... trừ khi cần tối ưu.

• Tạo: document.createElement; chèn: append/appendChild/before/after/replaceWith; xoá: remove().

• classList: add/remove/toggle/contains. Thuộc tính: getAttribute/setAttribute, dataset cho data-\*; style cho CSS inline (ưu tiên dùng class).

// Chọn & tạo  
const list = document.querySelector('#list');  
const li = document.createElement('li');  
li.textContent = 'Item mới';  
li.classList.add('active');  
li.dataset.id = '123'; // <li data-id="123">  
list.append(li); // hoặc list.appendChild(li)  
  
// Chèn/xoá linh hoạt  
list.prepend(document.createElement('hr'));  
li.remove(); // xoá node

## 2) Sự kiện: bubbling/capture, target/currentTarget, delegation, preventDefault

• Bubbling: sự kiện nổi lên từ phần tử con → cha. Capture: đi từ ngoài vào trong (ít dùng).

• e.target là nơi phát sinh; e.currentTarget là listener gán ở đâu. preventDefault() chặn hành vi mặc định; stopPropagation() dừng lan truyền.

• Event delegation: gắn 1 listener ở phần cha, kiểm tra e.target để xử lý nhiều phần tử con động.

const ul = document.querySelector('#todo');  
ul.addEventListener('click', (e) => {  
 const delBtn = e.target.closest('[data-del]');  
 if (delBtn) {  
 const li = delBtn.closest('li');  
 li?.remove();  
 }  
});  
  
// preventDefault cho form submit  
const form = document.querySelector('#form');  
form.addEventListener('submit', (e) => {  
 e.preventDefault();  
 // tự xử lý thay vì reload trang  
});

## 3) Form & Validation: required, pattern, custom message

• Dùng thuộc tính HTML5 (required, minLength, pattern) + kiểm tra JS cho các logic phức tạp.

• reportValidity() hiển thị message mặc định; setCustomValidity() để đặt thông báo tuỳ biến.

const form2 = document.querySelector('#register');  
const emailEl = form2.querySelector('input[name=email]');  
const passEl = form2.querySelector('input[name=password]');  
  
form2.addEventListener('submit', (e) => {  
 e.preventDefault();  
 emailEl.setCustomValidity('');  
 passEl.setCustomValidity('');  
 const emailOk = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(emailEl.value.trim());  
 if (!emailOk) {  
 emailEl.setCustomValidity('Email không hợp lệ');  
 }  
 if (passEl.value.length < 6) {  
 passEl.setCustomValidity('Mật khẩu tối thiểu 6 ký tự');  
 }  
 if (form2.reportValidity()) {  
 // Gửi dữ liệu hoặc xử lý  
 }  
});

## 4) Fetch API: GET/POST JSON, xử lý lỗi mạng

• fetch trả về Promise<Response>. Cần kiểm tra response.ok; parse JSON bằng res.json().

• Với POST, cần headers 'Content-Type: application/json' và JSON.stringify(body). Dùng try/catch + hiển thị trạng thái (loading/error).

async function getPosts(page=1, limit=10){  
 const url = `https://jsonplaceholder.typicode.com/posts?\_page=${page}&\_limit=${limit}`;  
 const res = await fetch(url);  
 if (!res.ok) throw new Error('Network error');  
 return res.json();  
}  
  
async function createPost(data){  
 const res = await fetch('https://jsonplaceholder.typicode.com/posts', {  
 method: 'POST',  
 headers: {'Content-Type':'application/json'},  
 body: JSON.stringify(data)  
 });  
 if (!res.ok) throw new Error('Create error');  
 return res.json();  
}  
  
async function load(){  
 const state = document.querySelector('#state');  
 const list = document.querySelector('#list');  
 state.textContent = 'Đang tải...';  
 try {  
 const posts = await getPosts(1, 10);  
 list.innerHTML = posts.map(p=>`<li><b>${p.id}</b> ${p.title}</li>`).join('');  
 state.textContent = '';  
 } catch (e){  
 state.textContent = 'Lỗi: ' + e.message;  
 }  
}

## 5) Phân trang & tải thêm (Load more / Infinite)

• Lưu biến page, cờ loading/finished; khi nhấn "Tải thêm" hoặc cuộn gần cuối trang thì nạp thêm dữ liệu.

• Tránh đúp request: khoá bằng biến loading; khi không còn dữ liệu, đặt finished=true.

let page = 1, loading = false, finished = false;  
const state = document.querySelector('#state');  
const list = document.querySelector('#list');  
const moreBtn = document.querySelector('#more');  
  
async function loadMore(){  
 if (loading || finished) return;  
 loading = true;  
 state.textContent = 'Đang tải...';  
 try {  
 const data = await getPosts(page, 10);  
 if (!data.length){  
 finished = true;  
 moreBtn.disabled = true;  
 state.textContent = 'Hết dữ liệu';  
 return;  
 }  
 list.insertAdjacentHTML('beforeend', data.map(p=>`<li><b>${p.id}</b> ${p.title}</li>`).join(''));  
 page++;  
 state.textContent = '';  
 } catch(e){  
 state.textContent = 'Lỗi: ' + e.message;  
 } finally {  
 loading = false;  
 }  
}  
  
moreBtn.addEventListener('click', loadMore);  
loadMore(); // lần đầu

## 6) Huỷ request (AbortController), Timeout, Retry/backoff

• AbortController giúp huỷ fetch khi người dùng đổi filter hoặc rời trang. Dùng setTimeout mô phỏng timeout; retry theo backoff khi lỗi tạm thời.

async function fetchWithAbort(url, {signal}={}){  
 const res = await fetch(url, { signal });  
 if (!res.ok) throw new Error('Network error');  
 return res.json();  
}  
  
const controller = new AbortController();  
const signal = controller.signal;  
fetchWithAbort('/api/search?q=abc', { signal }).catch(err=>console.log('aborted?', err.name));  
controller.abort(); // huỷ nếu người dùng thay đổi từ khoá  
  
// Retry/backoff đơn giản  
async function fetchRetry(url, tries=3){  
 let err;  
 for(let i=0;i<tries;i++){  
 try { return await fetch(url).then(r=> r.ok ? r.json() : Promise.reject(new Error('Bad status'))) }  
 catch(e){ err = e; await new Promise(r=> setTimeout(r, 300\*(i+1))); }  
 }  
 throw err;  
}

## 7) Lưu trữ: LocalStorage & SessionStorage — wrapper an toàn

• localStorage lưu chuỗi theo key/value, tồn tại lâu dài; sessionStorage sống theo tab. Cần JSON.stringify/parse khi lưu object.

• Bọc hàm get/set/remove/clear để tránh lỗi JSON và dễ test.

const storage = {  
 get(key, fallback=null){  
 try { return JSON.parse(localStorage.getItem(key)) ?? fallback; }  
 catch { return fallback; }  
 },  
 set(key, value){ localStorage.setItem(key, JSON.stringify(value)); },  
 remove(key){ localStorage.removeItem(key); },  
 clear(){ localStorage.clear(); }  
};  
  
// Dark mode toggle ví dụ  
const THEME\_KEY = 'theme';  
function applyTheme(theme){  
 document.body.classList.toggle('dark', theme==='dark');  
}  
function currentTheme(){ return storage.get(THEME\_KEY, 'light'); }  
applyTheme(currentTheme());  
document.querySelector('#toggle').addEventListener('click', ()=>{  
 const next = currentTheme()==='dark' ? 'light' : 'dark';  
 storage.set(THEME\_KEY, next);  
 applyTheme(next);  
});

## 8) Tìm kiếm & debounce (chống gọi API liên tục)

• Debounce: chờ người dùng dừng gõ một khoảng thời gian rồi mới thực thi; tránh spam API.

function debounce(fn, delay=300){  
 let t;  
 return (...args)=>{  
 clearTimeout(t);  
 t = setTimeout(()=>fn(...args), delay);  
 };  
}  
  
const input = document.querySelector('#q');  
const list = document.querySelector('#list');  
let cacheAll = [];  
  
async function init(){  
 cacheAll = await getPosts(1, 100); // lấy 100 post trước để filter client  
 render(cacheAll);  
}  
function render(data){  
 list.innerHTML = data.map(p=>`<li><b>${p.id}</b> ${p.title}</li>`).join('');  
}  
const handleSearch = debounce(()=>{  
 const kw = input.value.trim().toLowerCase();  
 const filtered = cacheAll.filter(p => p.title.toLowerCase().includes(kw));  
 render(filtered);  
}, 300);  
  
input.addEventListener('input', handleSearch);  
init();

## 9) Ghi chú nhanh về CORS & Same-origin

• Same-origin policy chặn JS truy cập tài nguyên khác origin nếu không được cho phép.

• CORS: server thêm header Access-Control-Allow-Origin để cho phép trình duyệt gửi/nhận; với các demo public API (như JSONPlaceholder) đã bật sẵn.

• Khi mở file .html trực tiếp (file://), một số trình duyệt có thể chặn fetch; chạy qua một static server (npx serve) sẽ an toàn hơn.

## 10) Lỗi thường gặp & mẹo

• Query selector sai (#id/.class) → không tìm thấy phần tử; nên kiểm tra null và dùng optional chaining.

• Gỡ/đặt listener nhiều lần → rò rỉ; khi dùng delegation thì chỉ cần 1 listener ở cha.

• Quên kiểm tra response.ok → JSON.parse trên body lỗi; luôn try/catch và hiển thị trạng thái.

• Không khoá cờ loading → bấm nhanh gây đúp request; nhớ dùng biến loading/finished.

• Lưu object vào localStorage mà quên JSON.stringify → lưu thành '[object Object]'.