**Seminarul 4 — Partea 2 (Laborator, extins): ESM, evenimente & DOM**

\*\*Obiectiv general.\*\* În această parte practică construim, pas cu pas, „StudentHub UI” folosind \*\*ES modules\*\* (ESM) nativ în browser, gestionăm \*\*evenimente\*\* DOM în mod robust (delegare, opțiuni, cleanup cu `AbortController`) și discutăm \*\*dependency management\*\* (import maps vs bundler) din perspectiva unui laborator didactic. Fiecare exercițiu are un \*worksheet\* (cerință + checklist), un \*starter code\* și o suită de \*\*unit tests\*\* în oglindă (\*\*Vitest\*\* și \*\*Jest\*\*) pentru a valida conceptele. Pe parcurs, indicăm \*AI‑assist (VSL)\* pentru accelerarea muncii: \*\*Verify\*\* (edge‑cases), \*\*Specify\*\* (contracte/semnături), \*\*Learn\*\* (refactor/măsurare).

---

## 0. Setup inițial (5–10 minute)

\*\*Cerință.\*\* Clonați arhiva `s4-p2-lab-standalone.zip` sau porniți de la scheletul minim descris aici. În directorul proiectului, rulați instalarea și testele:

npm i  
npm run test  
npm run serve # http://localhost:5173/public/index.html

\*\*Checklist.\*\* □ Node 20+ instalat; □ `npm run test` trece (Vitest + Jest); □ pagina se deschide fără erori în console; □ markup‑ul inițial afișează lista din `public/data/events.json`.

\*\*AI‑assist (Verify):\*\* cere modelului 15 posibile erori de configurare (versiuni, ESM în Jest, `jsdom`, căi relative HTML→module).

---

## 1. L1 — Fundament: ESM + DOM helpers + delegare evenimente

### 1.1. Bootstrap ESM (script type="module")

\*\*Explicație.\*\* `public/index.html` încarcă `src/main.js` ca ESM. Avantaj: izolare lexicală, `import`/`export` declarativ, analiză statică. În laborator, ne asigurăm că toate importurile sunt relative (`/src/...` sau `./...`) pentru a evita ambiguități.

<script type="module" src="/src/main.js"></script>

\*\*Cerință.\*\* Verificați că `main.js` exportă `bootstrap(document)` și că, la încărcarea paginii, este apelat automat (auto‑bootstrap).

\*\*Checklist.\*\* □ există `<script type="module">`; □ fără `inline handlers` în HTML; □ `bootstrap` randă lista la pornire.

### 1.2. DOM helpers cu invariante

\*\*Explicație.\*\* `qs` aruncă dacă selectorul lipsește — mai bine să „eșuăm devreme” decât să propagăm `null`.

export const qs = (sel, root=document) => { const el=root.querySelector(sel); if(!el) throw new Error(`Missing element: ${sel}`); return el; };  
export const qsa = (sel, root=document) => Array.from(root.querySelectorAll(sel));

\*\*Cerință.\*\* Refactorizați codul astfel încât toate selectările critice folosesc `qs`. Scrieți teste care demonstrează aruncarea.

\*\*Checklist.\*\* □ `qs` folosit pentru `#list`, `#q`, `#clear`; □ testele confirmă aruncarea pentru elemente lipsă.

### 1.3. Delegare evenimente (event delegation)

\*\*Explicație.\*\* Atașăm un singur handler pe listă și filtrăm `event.target` cu `.closest('li.card')`. Acest tipar face față elementelor inserate dinamic și reduce numărul de handler‑e.

export function on(container, type, selector, handler, options){  
 const listener = (ev) => {  
 const t = ev.target.closest(selector);  
 if (!t || !container.contains(t)) return;  
 handler(ev, t);  
 };  
 container.addEventListener(type, listener, options);  
 return () => container.removeEventListener(type, listener, options);  
}

\*\*Cerință.\*\* Marcați cardul apăsat cu clasa `active`. Scrieți teste care simulează click pe butonul din card și verifică togglingul.

\*\*Checklist.\*\* □ un singur handler pe container; □ toggling corect; □ test de delegare verde.

### 1.4. Cleanup cu `AbortController`

\*\*Explicație.\*\* Legăm ascultările la un `AbortSignal`; la `teardown` apelăm `abort()` și curățăm toate subscrierile.

export function listenWithSignal(el, type, cb, signal, options){ el.addEventListener(type, cb, { ...options, signal }); }

\*\*Cerință.\*\* Injectați un `AbortController` în `bootstrap` și utilizați `listenWithSignal` pentru butoane/inputs.

\*\*Checklist.\*\* □ `bootstrap` returnează o funcție `dispose()` care apelează `abort()`; □ test dedicat confirmă că, după `abort()`, click‑urile nu mai cresc contorul.

---

## 2. L2 — Intermediar: strat de servicii, lazy import, măsurători simple

### 2.1. Strat `services/data.js` cu fallback local

\*\*Explicație.\*\* Stratul de servicii decuplează UI de sursa datelor. În laborator, folosim un JSON local (`/public/data/events.json`), dar designul permite înlocuirea cu un endpoint ulterior.

export async function loadEvents(){  
 try{  
 const r = await fetch('/public/data/events.json'); if(!r.ok) throw new Error(`HTTP ${r.status}`);  
 const arr = await r.json();  
 return arr.map(x => ({ id:String(x.id), title:String(x.title||''), when:new Date(x.when).toISOString(), club:String(x.club||'') }));  
 }catch(e){  
 return [];  
 }  
}

\*\*Cerință.\*\* Stabiliți contractul: array de obiecte `{id,title,when,club}`. În teste, \*mock\*‑uiți `fetch` și verificați normalizarea datelor.

\*\*Checklist.\*\* □ `loadEvents` nu aruncă în caz de eșec, ci întoarce `[]`; □ ISO‑8601 pentru `when`.

### 2.2. Import dinamic cu wrapper spionabil

\*\*Explicație.\*\* Pentru a testa că lazy‑loadingul se întâmplă \*\*doar\*\* la interacțiune, introducem `loader.dynamicImport(path)` ca fațadă peste `import()`; în teste, îl spionăm.

export const dynamicImport = (path) => import(path);

\*\*Cerință.\*\* În `main.js`, la primul `input` în câmpul de căutare, apelați `dynamicImport('/src/filters.js')` și filtrați lista.

\*\*Checklist.\*\* □ `dynamicImport` nu este chemat la bootstrap; □ este chemat la primul `input`; □ testul „lazy” trece.

### 2.3. `modulepreload` & feedback de performanță

\*\*Explicație.\*\* Adăugăm `<link rel="modulepreload" href="/src/main.js">` pentru a accelera încărcarea. În proiecte reale, am preîncărca și dependențe secundare critice. În laborator, descrieți în README cum se măsoară un \*first interaction time\* minimal (ex.: `performance.now()` înainte/după).

\*\*Cerință.\*\* Documentați în README observabilele (de ex., diferența percepută pe conexiuni lente).

\*\*Checklist.\*\* □ tag `modulepreload` prezent; □ secțiune în README despre măsurători.

---

## 3. L3 — Avansat: event bus, MutationObserver, boundary modules, CSP sanity

### 3.1. Bus de evenimente pe DOM cu `CustomEvent`

\*\*Explicație.\*\* Pentru comunicare între module (de ex., `ui/list` și `services/analytics`), folosim evenimente „namespaced” (`hub:\*`).

export function emit(target, name, detail){ target.dispatchEvent(new CustomEvent(name, { detail, bubbles:true })); }  
export function onEvent(target, name, handler, options){ target.addEventListener(name, handler, options); return () => target.removeEventListener(name, handler, options); }

\*\*Cerință.\*\* Emiteți `hub:filter` când utilizatorul tastează. Alt modul (mock în teste) numără evenimentele primite.

\*\*Checklist.\*\* □ `detail` conține șirul căutat; □ testul confirmă recepția.

### 3.2. `MutationObserver` & batch vizual cu `requestAnimationFrame` (concept)

\*\*Explicație.\*\* Pentru liste care se actualizează frecvent, un `MutationObserver` poate declanșa efecte secundare (ex.: actualizare unui contor) dar \*\*bătute\*\* în `rAF` pentru a evita „thrash‑ul”. În laborator, menționăm practicile în README; implementarea efectivă e opțională la acest punct.

\*\*Cerință.\*\* Documentați cum ați conecta un `MutationObserver` la modificările din `#list`.

\*\*Checklist.\*\* □ în README o schemă de evenimente clară.

### 3.3. Boundary modules & index modules

\*\*Explicație.\*\* Introduceți un `services/index.js` care re‑exportă din `data.js`. Consumatorii nu mai depind de layout‑ul intern al `services/`.

\*\*Cerință.\*\* Creați un modul `services/index.js` care re‑exportă `loadEvents`. Actualizați importurile în `main.js`.

\*\*Checklist.\*\* □ importurile din `main.js` vin din `services/index.js`.

### 3.4. Igienă CSP: fără inline handlers

\*\*Explicație.\*\* În HTML nu folosim `onclick`/`oninput`. Testul `csp.test.js` scanează `public/index.html` pentru a verifica absența `on\*=`.

\*\*Checklist.\*\* □ testul CSP verde; □ README explică legătura cu CSP.

---

## Worksheet — Cerință & Checklist consolidate

\*\*Cerință finală.\*\* Implementați „StudentHub UI” astfel încât să: (1) folosească ESM; (2) aibă \*delegation\* pe listă; (3) curețe subscrierile la `dispose()` via `AbortController`; (4) încarce filtrarea \*\*lazy\*\*; (5) respecte igiena CSP (fără `on\*=` în HTML).

\*\*Checklist complet.\*\*  
□ `script type="module"`; □ exporturi nominale clare; □ `qs/qsa` cu invariante; □ `on()` pentru delegare; □ `listenWithSignal()` pentru cleanup; □ `loadEvents()` normalizează datele; □ `dynamicImport()` folosit doar la input; □ `modulepreload`; □ `README` cu măsurători; □ testele Vitest + Jest verzi; □ testul CSP verde.

---

## Unit tests în oglindă (Vitest și Jest)

\*\*Explicație.\*\* Același comportament este verificat cu ambele tool‑uri: \*environment\* `jsdom`, \*fixtures\* minimaliste, \*mocks\* pentru `fetch`, spion pe `loader.dynamicImport` pentru a verifica \*lazy\*.

\*\*Exemple incluse în arhivă:\*\* `dom.test.js`, `renderList.test.js`, `delegation.test.js`, `abort.test.js`, `bus.test.js`, `lazy.test.js`, `csp.test.js` (+ oglinzi Jest).

---

## Rulare & depanare

# Standalone  
npm i  
npm run test  
npm run serve # deschide http://localhost:5173/public/index.html  
  
# Monorepo (dacă folosiți varianta PNPM)  
pnpm i  
pnpm -r run test  
pnpm -F @s4/lab-s4 run serve

\*\*Probleme comune.\*\* ESM în Jest (rezolvat cu `babel-jest` + `extensionsToTreatAsEsm`), căi relative (`/src/...` din `public/index.html`), lipsa `jsdom`, versiuni Node prea vechi.

---

## AI‑assist (VSL) — sugestii aplicate

\*\*Verify.\*\* Cereți 20 de capcane potențiale pentru DOM/events (ex.: `target` vs `currentTarget`, `passive` pe scroll, memorie când nu se apelează `abort()`).  
\*\*Specify.\*\* Contracte de modul (semnături, tipuri de erori) — fixați înainte de a scrie implementarea.  
\*\*Learn.\*\* După ce testele sunt verzi, cereți un refactor: extrageți `bus` într‑un modul separat și demonstrați, prin teste, că nu mai există „dublu attach”.

---

## Bibliografie (APA 7, cu DOI)

Wirfs‑Brock, A., & Eich, B. (2020). JavaScript: The first 20 years. Proceedings of the ACM on Programming Languages, 4(HOPL), 1–189. https://doi.org/10.1145/3386327

Loring, M. C., Laurenzano, M. A., Newsham, Z., Hovsmith, N., Pande, S., Barik, T., & others. (2017). Semantics of asynchronous JavaScript. ACM SIGPLAN Notices, 52(11), 51–62. https://doi.org/10.1145/3170472.3133846

Maffeis, S., Mitchell, J. C., & Taly, A. (2008). An operational semantics for JavaScript. In Programming Languages and Systems (pp. 307–325). Springer. https://doi.org/10.1007/978-3-540-89330-1\_22

Ungar, D., & Smith, R. B. (1991). SELF: The power of simplicity. Higher‑Order and Symbolic Computation, 4(3), 171–216. https://doi.org/10.1007/BF01806105