**Seminarul 7 — Partea 2: Laborator (extins)  
Fetch API + formulare + JSON + persistență minimă**

**Obiectivul laboratorului și scenariul pe care îl implementăm**

În acest laborator construim, pas cu pas, o experiență completă „form + listă de date” pentru \*\*Clubs & Associations Hub\*\*, folosind strict browser APIs moderne: \*\*Fetch API\*\* pentru comunicarea HTTP, \*\*HTML forms\*\* cu \*\*Constraint Validation API\*\* + validare custom, \*\*JSON\*\* pentru schimb de date și o \*\*persistență minimă\*\* pe client (cache + draft autosave) pentru reziliență. Vom urmări o abordare \*\*mobile‑first\*\*, vom proiecta fluxul ca pe o \*\*mașină de stări\*\* (`idle → loading → success|error`) și vom scrie \*\*teste unitare\*\* în paralel (Vitest & Jest) pentru părțile critice: `fetchClient` (timeout/retry), validare (predicates pure), persistență (TTL/draft) și actualizări UI.

Scenariul pedagogic este următorul: pagina \*\*Clubs\*\* consumă `GET /api/clubs`, afișează lista cluburilor și oferă butoane pentru „reload”/„clear cache”. Pagina \*\*Enrol\*\* conține formularul de înscriere (nume, email, club, consimțământ), cu validare nativă+custom, `POST /api/registrations` în format \*\*JSON\*\*, `aria-live` pentru feedback și un \*\*autosave\*\* al draftului în `localStorage`. Pe partea de server folosim un \*\*mock Express\*\* pentru rute minime, iar toate cerințele sunt verificabile prin testele incluse.

**Structura proiectului (starter) și scripturi de lucru**

s7p2-lab/  
 public/{index.html, enrol.html}  
 src/  
 api/fetchClient.js  
 services/clubs.js  
 forms/{validation.js, enrolController.js}  
 pages/{clubsPage.js, enrolPage.js}  
 utils/storage.js  
 main.js  
 tests/  
 unit/{fetchClient.vitest.test.ts, fetchClient.jest.test.cjs,  
 validation.vitest.test.ts, validation.jest.test.cjs,  
 persistence.vitest.test.ts, persistence.jest.test.cjs,  
 render.vitest.test.ts, render.jest.test.cjs}  
 fixtures/{clubs.json, registration.ok.json, registration.problem.json}  
 styles.css  
 server.mjs  
 package.json  
 vitest.config.ts  
 jest.config.cjs

Scripturile standard sunt: `npm run dev` (pornește mock server pe \*\*http://localhost:5270\*\*), `npm run test:vitest`, `npm run test:jest` și `npm test` (le rulează pe ambele). Păstrăm intențiile strict separate (API client, validare, stocare locală, pagini) pentru ca testele să fie curate și pentru a promova un design modular.

**Worksheet — enunțul de lucru și checklist de predare**

\*\*Enunț.\*\* Realizează paginile \*\*Clubs\*\* și \*\*Enrol\*\* (HTML, CSS minimal, JS). Pentru \*\*Clubs\*\*, datele provin din `GET /api/clubs`; afișezi lista, asiguri „reload” și „clear cache”. Pentru \*\*Enrol\*\*, colectezi datele în format JSON, validezi (nativ & custom), trimiți `POST /api/registrations`, raportezi stările în `aria-live` și menții un „draft autosave”.

\*\*Checklist.\*\*  
• ✅ `fetch` cu antete corecte (`Accept`, `Content-Type`) pentru \*\*JSON\*\*.  
• ✅ `AbortController` + \*\*timeout\*\*; `retry` pentru erori tranzitorii (`5xx`, network).  
• ✅ \*\*Validare nativă\*\* (atribute HTML) + \*\*validare custom\*\* (predicates pure).  
• ✅ \*\*JSON parsing\*\* defensiv și mapare a erorilor în mesaje UI.  
• ✅ \*\*localStorage\*\*: cache cu \*\*TTL + SWR\*\*; „draft autosave” pentru formular.  
• ✅ \*\*a11y\*\*: `aria-live`, focus pe primul câmp invalid; buton `disabled` în `submitting`.  
• ✅ \*\*Teste\*\*: Vitest & Jest trec pentru părțile indicate.

**Pasul 1 — configurarea mediului (mock server, scripturi, CSS minim)**

Rulează `npm i`, apoi `npm run dev` și deschide în browser \*\*/public/index.html\*\* și \*\*/public/enrol.html\*\*. Mock serverul Express răspunde la `GET /api/clubs` și `POST /api/registrations` aplicând validări server‑side de bun‑simț (nume ≥3, email valid, club, consent). CSS‑ul este minimal și servește doar lizibilității și pattern‑urilor de layout simple.

**Pasul 2 — Fetch: GET /api/clubs cu cache (TTL + SWR)**

În `src/services/clubs.js` folosim `jsonGet` pentru a obține lista cluburilor. Resultatul e memorat în `localStorage` cu \*\*TTL 5 min\*\*; dacă există cache, îl afișăm imediat și declanșăm un \*\*refresh\*\* în fundal (SWR).

import { jsonGet } from '../api/fetchClient.js';  
import { lsGet, lsSet } from '../utils/storage.js';  
  
const TTL = 5 \* 60 \* 1000; // 5 minutes  
  
export async function getClubsCached(){  
 const cached = lsGet('clubs');  
 if(cached){  
 // SWR în fundal  
 setTimeout(() => refreshClubs(), 0);  
 return cached;  
 }  
 return refreshClubs();  
}  
  
export async function refreshClubs(){  
 const data = await jsonGet('/api/clubs');  
 lsSet('clubs', data, TTL);  
 return data;  
}

**Pasul 3 — Pagina Clubs: randare, reload, clear cache**

În `src/pages/clubsPage.js` randăm lista (`<ul class="cards">`), raportăm starea în `#status` (`aria-live`) și conectăm butoanele „Reload”/„Clear cache”.

import { getClubsCached } from '../services/clubs.js';  
  
const list = document.querySelector('#clubs');  
const statusEl = document.querySelector('#status');  
const reloadBtn = document.querySelector('#reload');  
const clearBtn = document.querySelector('#clear-cache');  
  
function renderClubs(clubs){  
 list.innerHTML = '';  
 for(const c of clubs){  
 const li = document.createElement('li');  
 li.textContent = `${c.name} — ${c.category}`;  
 list.appendChild(li);  
 }  
}  
  
async function load(){  
 statusEl.textContent = 'Loading…';  
 try{  
 const clubs = await getClubsCached();  
 renderClubs(clubs);  
 statusEl.textContent = `Loaded ${clubs.length} clubs.`;  
 }catch(err){  
 console.error(err);  
 statusEl.textContent = 'Network/server error. Try again.';  
 }  
}  
  
reloadBtn?.addEventListener('click', load);  
clearBtn?.addEventListener('click', () => { localStorage.clear(); statusEl.textContent = 'Cache cleared.'; });  
load();

**Pasul 4 — Formularul Enrol: semnatică, validare nativă + custom, focus management**

Validarea nativă (atribute) este completată cu validarea \*\*custom\*\* în `src/forms/validation.js`, implementată ca funcții pure pentru testabilitate. Rezultatul este `{ ok, data, errors }`; primul câmp invalid primește \*\*focus\*\*.

export function validateEnrol(data){  
 const errors = {};  
 const fullName = (data.fullName||'').trim();  
 const email = (data.email||'').trim();  
 const club = (data.club||'').trim();  
 const consent = !!(data.consent === true || data.consent === 'on' || data.consent === 'true');  
 if(fullName.length < 3) errors.fullName = 'Numele trebuie să aibă cel puțin 3 caractere.';  
 if(!/^[^@\s]+@[^@\s]+\.[^@\s]+$/.test(email)) errors.email = 'Email invalid.';  
 if(!club) errors.club = 'Alege un club.';  
 if(!consent) errors.consent = 'Este necesar consimțământul.';  
 return { ok: Object.keys(errors).length===0, data: { fullName, email, club, consent }, errors };  
}  
  
export function formToJSON(form){  
 const fd = new FormData(form);  
 const raw = Object.fromEntries(fd.entries());  
 // normalizare checkbox  
 if(!('consent' in raw)) raw.consent = false;  
 return validateEnrol(raw);  
}

**Pasul 5 — Submit JSON: mașină de stări, `aria-live`, reset & autosave**

Controllerul din `src/forms/enrolController.js` gestionează `submit`, actualizează UI în funcție de stări (`updateUI`) și folosește `jsonPost` din `fetchClient.js`. La succes, \*\*șterge draftul\*\* și face `reset` pe formular.

import { formToJSON } from './validation.js';  
import { jsonPost } from '../api/fetchClient.js';  
import { saveDraft, restoreDraft, clearDraft } from '../utils/storage.js';  
  
export function updateUI(form, state){  
 const submitBtn = form.querySelector('#submit');  
 const statusEl = form.querySelector('#form-status');  
 if(state.status === 'submitting'){  
 submitBtn.disabled = true; statusEl.textContent = 'Se trimite…';  
 }else if(state.status === 'success'){  
 submitBtn.disabled = false; statusEl.textContent = 'Înscriere trimisă cu succes.';  
 }else if(state.status === 'error'){  
 submitBtn.disabled = false; statusEl.textContent = (state.problem?.title) || 'Eroare de rețea/server.';  
 }else{  
 submitBtn.disabled = false; statusEl.textContent = '';  
 }  
}  
  
export function attachEnrolForm(form){  
 restoreDraft(form);  
 form.addEventListener('input', () => saveDraft(form));  
 form.addEventListener('submit', async (e) => {  
 e.preventDefault();  
 const v = formToJSON(form);  
 // focus pe primul câmp invalid  
 if(!v.ok){  
 const first = Object.keys(v.errors)[0];  
 const el = form.elements.namedItem(first);  
 if(el && el.focus) el.focus();  
 updateUI(form, { status: 'error', problem: { title: Object.values(v.errors).join(' ') } });  
 return;  
 }  
 updateUI(form, { status: 'submitting' });  
 try{  
 const res = await jsonPost('/api/registrations', v.data, { timeoutMs: 8000, retries: 2 });  
 updateUI(form, { status: 'success' });  
 clearDraft();  
 form.reset();  
 }catch(err){  
 updateUI(form, { status: 'error', problem: { title: err?.payload?.title || err?.message || 'Eroare' } });  
 }  
 });  
}

**Pasul 6 — Fetch client: timeout + retry (exponențial cu jitter), JSON helpers**

`src/api/fetchClient.js` conține un \*\*wrapper\*\* cu `AbortController` pentru timeout, \*\*retry\*\* pe `5xx`/network și helperi `jsonGet/jsonPost` care ridică `HttpError` cu payload (dacă există).

const BASE\_URL = '';  
const DEFAULT\_TIMEOUT = 8000;  
  
export class HttpError extends Error{  
 constructor(message, status, payload){ super(message); this.name = 'HttpError'; this.status = status; this.payload = payload; }  
}  
  
export async function fetchWithRetry(input, init = {}, { retries = 3, timeoutMs = DEFAULT\_TIMEOUT } = {}){  
 let attempt = 0;  
 while(true){  
 const ac = new AbortController();  
 const to = setTimeout(() => ac.abort(), timeoutMs);  
 try{  
 const res = await fetch(input, { ...init, signal: ac.signal });  
 clearTimeout(to);  
 if(!res.ok){  
 if(res.status >= 500 && res.status < 600 && attempt < retries){ attempt++; await backoff(attempt); continue; }  
 return res;  
 }  
 return res;  
 }catch(err){  
 clearTimeout(to);  
 if(attempt < retries){ attempt++; await backoff(attempt); continue; }  
 throw err;  
 }  
 }  
}  
  
function jitter(ms){ return ms \* (0.8 + Math.random()\*0.4); }  
function sleep(ms){ return new Promise(r => setTimeout(r, ms)); }  
async function backoff(attempt){ const base = Math.min(1000 \* (2 \*\* (attempt-1)), 8000); await sleep(jitter(base)); }  
  
async function parseJSON(res){  
 const ct = res.headers.get('content-type') || '';  
 const isJSON = ct.includes('application/json');  
 if(!isJSON) throw new Error('Unexpected content-type');  
 try{ return await res.json(); }catch{ throw new Error('Invalid JSON'); }  
}  
  
export async function jsonGet(path, options = {}){  
 const url = BASE\_URL + path;  
 const res = await fetchWithRetry(url, { method: 'GET', headers: { 'Accept': 'application/json', ...(options.headers||{}) } }, options);  
 if(!res.ok){  
 let payload = null; try{ payload = await res.clone().json(); }catch{}  
 throw new HttpError(`HTTP ${res.status}`, res.status, payload);  
 }  
 return parseJSON(res);  
}  
  
export async function jsonPost(path, data, options = {}){  
 const url = BASE\_URL + path;  
 const res = await fetchWithRetry(url, {  
 method: 'POST',  
 headers: { 'Accept': 'application/json', 'Content-Type': 'application/json', ...(options.headers||{}) },  
 body: JSON.stringify(data)  
 }, options);  
 if(!res.ok){  
 let payload = null; try{ payload = await res.clone().json(); }catch{}  
 throw new HttpError(`HTTP ${res.status}`, res.status, payload);  
 }  
 return parseJSON(res);  
}

**Pasul 7 — Persistență minimă: localStorage TTL + draft autosave**

`src/utils/storage.js` oferă un namespace, perechi `lsGet/lsSet` (cu `expiresAt`) și utilitare pentru \*\*draft\*\* (save/restore/clear).

const NS = 's7:';  
  
export function lsGet(key){  
 try{  
 const raw = localStorage.getItem(NS+key); if(!raw) return null;  
 const { value, expiresAt } = JSON.parse(raw);  
 if(expiresAt && Date.now() > expiresAt){ localStorage.removeItem(NS+key); return null; }  
 return value;  
 }catch{ return null; }  
}  
  
export function lsSet(key, value, ttlMs){  
 try{  
 const payload = { value, expiresAt: ttlMs ? Date.now()+ttlMs : null };  
 localStorage.setItem(NS+key, JSON.stringify(payload));  
 }catch{}  
}  
  
export function lsRemove(key){ try{ localStorage.removeItem(NS+key); }catch{} }  
  
const DRAFT\_KEY = 'draft:enrol';  
  
export function saveDraft(form){  
 const fd = new FormData(form);  
 const data = Object.fromEntries(fd.entries());  
 try{ localStorage.setItem(NS + DRAFT\_KEY, JSON.stringify(data)); }catch{}  
}  
export function restoreDraft(form){  
 try{  
 const raw = localStorage.getItem(NS + DRAFT\_KEY); if(!raw) return;  
 const data = JSON.parse(raw)||{};  
 for(const [k,v] of Object.entries(data)){  
 const el = form.elements.namedItem(k); if(!el) continue;  
 if(el instanceof RadioNodeList || el.type === 'checkbox'){ el.checked = !!v; }  
 else el.value = v;  
 }  
 }catch{}  
}  
export function clearDraft(){ try{ localStorage.removeItem(NS + DRAFT\_KEY); }catch{} }

**Pasul 8 — Teste unitare (Vitest & Jest, side‑by‑side)**

Testele vizează funcționalitatea critică și sunt oglindite pe ambele framework‑uri: `fetchClient` (retry/timeout/erori), validare (predicates), persistență (TTL) și comportament UI (`updateUI`). În Vitest folosim `vi.stubGlobal`, în Jest `global.fetch = jest.fn()`.

import { describe, it, expect, vi } from 'vitest';  
import { jsonGet, jsonPost, fetchWithRetry, HttpError } from '../../src/api/fetchClient.js';  
  
function mkResponse(body, init){ return new Response(JSON.stringify(body), { headers: { 'content-type': 'application/json' }, ...(init||{}) }); }  
  
describe('fetchClient — retry/timeout/headers', () => {  
 it('retries on 500 then succeeds', async () => {  
 const spy = vi.fn()  
 .mockResolvedValueOnce(mkResponse({ error: 'boom' }, { status: 500 }))  
 .mockResolvedValueOnce(mkResponse({ ok: true }, { status: 200 }));  
 vi.stubGlobal('fetch', spy);  
 const res = await fetchWithRetry('/x', {}, { retries: 1, timeoutMs: 2000 });  
 expect(spy).toHaveBeenCalledTimes(2);  
 expect(res.ok).toBe(true);  
 });  
  
 it('does not retry on 400', async () => {  
 const spy = vi.fn().mockResolvedValue(mkResponse({ error: 'bad' }, { status: 400 }));  
 vi.stubGlobal('fetch', spy);  
 const res = await fetchWithRetry('/x', {}, { retries: 3, timeoutMs: 2000 });  
 expect(spy).toHaveBeenCalledTimes(1);  
 expect(res.status).toBe(400);  
 });  
  
 it('jsonPost throws HttpError on 422 with payload', async () => {  
 vi.stubGlobal('fetch', vi.fn().mockResolvedValue(mkResponse({ title: 'Invalid' }, { status: 422 })));  
 await expect(jsonPost('/y', { a: 1 })).rejects.toBeInstanceOf(HttpError);  
 });  
});

const { jsonGet, jsonPost, fetchWithRetry, HttpError } = require('../../src/api/fetchClient.js');  
  
function mkResponse(body, init){ return new Response(JSON.stringify(body), { headers: { 'content-type': 'application/json' }, ...(init||{}) }); }  
  
test('retries on 500 then succeeds', async () => {  
 global.fetch = jest.fn()  
 .mockResolvedValueOnce(mkResponse({ error: 'boom' }, { status: 500 }))  
 .mockResolvedValueOnce(mkResponse({ ok: true }, { status: 200 }));  
 const res = await fetchWithRetry('/x', {}, { retries: 1, timeoutMs: 2000 });  
 expect(global.fetch).toHaveBeenCalledTimes(2);  
 expect(res.ok).toBe(true);  
});  
  
test('does not retry on 400', async () => {  
 global.fetch = jest.fn().mockResolvedValue(mkResponse({ error: 'bad' }, { status: 400 }));  
 const res = await fetchWithRetry('/x', {}, { retries: 3, timeoutMs: 2000 });  
 expect(global.fetch).toHaveBeenCalledTimes(1);  
 expect(res.status).toBe(400);  
});  
  
test('jsonPost throws HttpError on 422 with payload', async () => {  
 global.fetch = jest.fn().mockResolvedValue(mkResponse({ title: 'Invalid' }, { status: 422 }));  
 await expect(jsonPost('/y', { a: 1 })).rejects.toBeInstanceOf(HttpError);  
});

import { describe, it, expect } from 'vitest';  
import { validateEnrol } from '../../src/forms/validation.js';  
  
describe('validation', () => {  
 it('accepts valid data', () => {  
 const r = validateEnrol({ fullName:'Ana Pop', email:'ana@uni.ro', club:'tech', consent:true });  
 expect(r.ok).toBe(true);  
 });  
 it('rejects invalid email and short name', () => {  
 const r = validateEnrol({ fullName:'A', email:'x', club:'', consent:false });  
 expect(r.ok).toBe(false);  
 expect(Object.keys(r.errors).length).toBeGreaterThanOrEqual(3);  
 });  
});

const { validateEnrol } = require('../../src/forms/validation.js');  
test('accepts valid data', () => {  
 const r = validateEnrol({ fullName:'Ana Pop', email:'ana@uni.ro', club:'tech', consent:true });  
 expect(r.ok).toBe(true);  
});  
test('rejects invalid email and short name', () => {  
 const r = validateEnrol({ fullName:'A', email:'x', club:'', consent:false });  
 expect(r.ok).toBe(false);  
 expect(Object.keys(r.errors).length).toBeGreaterThanOrEqual(3);  
});

import { describe, it, expect } from 'vitest';  
import { lsSet, lsGet } from '../../src/utils/storage.js';  
  
describe('localStorage TTL', () => {  
 it('returns value before expiry, null after expiry', async () => {  
 lsSet('probe', { ok:true }, 10);  
 const a = lsGet('probe'); expect(a?.ok).toBe(true);  
 await new Promise(r => setTimeout(r, 20));  
 const b = lsGet('probe'); expect(b).toBe(null);  
 });  
});

const { lsSet, lsGet } = require('../../src/utils/storage.js');  
test('localStorage TTL', async () => {  
 lsSet('probe', { ok:true }, 10);  
 expect(lsGet('probe')?.ok).toBe(true);  
 await new Promise(r => setTimeout(r, 20));  
 expect(lsGet('probe')).toBe(null);  
});

import { describe, it, expect } from 'vitest';  
import { updateUI } from '../../src/forms/enrolController.js';  
  
document.body.innerHTML = `<form id="enrol">  
 <button id="submit" type="submit">OK</button>  
 <p id="form-status" aria-live="polite"></p>  
</form>`;  
  
const form = document.querySelector('#enrol');  
  
describe('updateUI', () => {  
 it('sets disabled while submitting and writes status text', () => {  
 updateUI(form, { status:'submitting' });  
 expect(form.querySelector('#submit').disabled).toBe(true);  
 expect(form.querySelector('#form-status').textContent).toMatch(/Se trimite/);  
 });  
});

const { updateUI } = require('../../src/forms/enrolController.js');  
  
document.body.innerHTML = `<form id="enrol">  
 <button id="submit" type="submit">OK</button>  
 <p id="form-status" aria-live="polite"></p>  
</form>`;  
  
test('sets disabled while submitting and writes status text', () => {  
 const form = document.querySelector('#enrol');  
 updateUI(form, { status:'submitting' });  
 expect(form.querySelector('#submit').disabled).toBe(true);  
 expect(form.querySelector('#form-status').textContent).toMatch(/Se trimite/);  
});

**Rubrică de evaluare (criterii și ponderi)**

• \*\*Funcționalitate & UX (40%)\*\* — listă clubs cu reload/clear cache; formular înscriere cu feedback robust; timpi rezonabili.   
• \*\*Calitatea codului (30%)\*\* — separare de responsabilități, funcții pure la validare, wrapper coerent pentru fetch, naming consecvent.  
• \*\*Accesibilitate (15%)\*\* — aria‑live, focus management, stări vizibile; fără bariere evidente.  
• \*\*Testare (15%)\*\* — testele unitare trec în Vitest & Jest; explicații succinte în README.

**Troubleshooting (cazuri frecvente)**

• \*\*CORS\*\*: dacă vei extinde către un backend real, configurează antetele `Access-Control-Allow-\*` (mock‑ul folosește `cors()`).  
• \*\*Timeout\*\*: prea scurt → false negatives; prea lung → UX slab. 6–8s este o limită pragmatică.  
• \*\*Retry\*\*: aplică‑l doar pe 5xx/rețea; 4xx nu se repară prin reîncercare.  
• \*\*localStorage TTL\*\*: verifică `expiresAt`; în test folosim TTL foarte mic.  
• \*\*Validare\*\*: nu uita fallback‑ul nativ (atribute); nu te baza exclusiv pe JS.

**AI‑assist (Copilot/LLM), VSL — cum te accelerezi fără a ceda controlul**

• „Rescrie `fetchWithRetry` ca funcție pură de policy (parametrizabilă).”  
• „Propune 3 mesaje `aria-live` prietenoase pentru `submitting/success/error`.”  
• „Generează teste pentru validarea numelui/emailului (cazuri limită).”  
• „Sugerează două alternative pentru `SWR` (inclusiv cu `Cache API`/`Service Worker`, dacă vrei să explorezi).”

**Referințe (APA 7, DOI real)**

Fielding, R., Nottingham, M., & Reschke, J. (2022). HTTP Semantics (RFC 9110). RFC Editor. https://doi.org/10.17487/RFC9110

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259). RFC Editor. https://doi.org/10.17487/RFC8259

Nottingham, M., & Wilde, E. (2016). Problem Details for HTTP APIs (RFC 7807). RFC Editor. https://doi.org/10.17487/RFC7807

WHATWG. (Living Standard). Fetch Standard. https://fetch.spec.whatwg.org/ (fără DOI)

WHATWG. (Living Standard). HTML Standard — Forms & Constraint Validation. https://html.spec.whatwg.org/ (fără DOI)