**Seminarul 3 — Partea 3 (Proiecte/teme, extins)**

Această parte conține 45 de proiecte (L1–L3) cu specificații detaliate și soluții rezumate.

# L1 — Fundamental

## L1-p01-student-vo: Student VO — invariante, freeze, branding

Specificație:

Creează clasa `Student` cu câmpuri obligatorii `id`, `name`, `email`, listă opțională `interests`.  
- Normalizează `email` la lowercase + trim.  
- Aplică `Object.freeze(this)` (immutabilitate superficială).  
- Setează `Symbol.toStringTag = 'Student'` și `toString()` informativ.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export class Student {  
 constructor({ id, name, email, interests = [] } = {}) {  
 if (!id || !name || !email) throw new ValidationError('Student fields required');  
 this.id = String(id);  
 this.name = String(name).trim();  
 this.email = String(email).toLowerCase().trim();  
 this.interests = Array.isArray(interests) ? interests.slice() : [];  
 Object.freeze(this);  
 }  
 toString() { return `[Student ${this.id}] ${this.name} <${this.email}>`; }  
 get [Symbol.toStringTag]() { return 'Student'; }  
}

## L1-p02-club-vo: Club VO — invariante și tags

Specificație:

Creează clasa `Club` cu câmpuri `id`, `title` și `tags` (array). Aplică `freeze`, branding și `toString()`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export class Club {  
 constructor({ id, title, tags = [] } = {}) {  
 if (!id || !title) throw new ValidationError('Club fields required');  
 this.id = String(id);  
 this.title = String(title).trim();  
 this.tags = Array.isArray(tags) ? tags.slice() : [];  
 Object.freeze(this);  
 }  
 toString() { return `[Club ${this.id}] ${this.title}`; }  
 get [Symbol.toStringTag]() { return 'Club'; }  
}

## L1-p03-event-vo: Event VO — dată și toString()

Specificație:

Implementează `Event` cu `id`, `label`, `when` (Date), `clubId?`. `freeze`, branding și `toString()` ISO.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export class Event {  
 constructor({ id, label, when, clubId } = {}) {  
 if (!id || !label || !when) throw new ValidationError('Event fields required');  
 this.id = String(id);  
 this.label = String(label).trim();  
 this.when = new Date(when);  
 this.clubId = clubId ? String(clubId) : null;  
 Object.freeze(this);  
 }  
 toString() { return `[Event ${this.id}] ${this.label} @ ${this.when.toISOString()}`; }  
 get [Symbol.toStringTag]() { return 'Event'; }  
}

## L1-p04-factory-student: Factory + Object.create + prototip

Specificație:

Scrie `makeStudent` care aplică invariante și setează prototipul `studentProto` via `Object.create`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
const studentProto = {  
 toString() { return `[Student ${this.id}] ${this.name} <${this.email}>`; },  
 get [Symbol.toStringTag]() { return 'Student'; }  
};  
export function makeStudent({ id, name, email, interests = [] } = {}) {  
 if (!id || !name || !email) throw new ValidationError('Student fields required');  
 const self = { id:String(id), name:String(name).trim(), email:String(email).toLowerCase().trim(), interests:Array.isArray(interests)?interests.slice():[] };  
 Object.freeze(self);  
 return Object.setPrototypeOf(self, studentProto);  
}

## L1-p05-descriptor-secret: Descriptori — proprietate neenumerabilă

Specificație:

Scrie `defineSecret(obj, value)` care atașează `secret` \*\*neenumerabil\*\* (`enumerable:false`).

Soluție (esentialul):

export function defineSecret(obj, value) {  
 Object.defineProperty(obj, 'secret', { value, enumerable:false, writable:false, configurable:false });  
 return obj;  
}

## L1-p06-branding-tag: Branding — Symbol.toStringTag utilitar

Specificație:

`brand(obj, tag)` setează `Symbol.toStringTag` pentru obiecte date.

Soluție (esentialul):

export function brand(obj, tag){Object.defineProperty(obj, Symbol.toStringTag, { value:String(tag), configurable:true }); return obj;}

## L1-p07-catalog-iterable: Catalog iterabil — Symbol.iterator

Specificație:

Construiți `Catalog` care iterează studenții în ordinea inserării.

Soluție (esentialul):

export class Catalog{constructor(items=[]){this.items=[...items];} \*[Symbol.iterator](){for(const it of this.items) yield it;}}

## L1-p08-equals-by: equalsBy — egalitate pe câmpuri

Specificație:

`equalsBy(keys, a, b)` verifică egalitatea pe un set de chei.

Soluție (esentialul):

export function equalsBy(keys, a, b){return keys.every(k=> (a?.[k]) === (b?.[k]));}

## L1-p09-key-by-email: keyByEmail — mapare rapidă

Specificație:

`keyByEmail(list)` → `Map(email → obiect)` cu email normalizat.

Soluție (esentialul):

export function keyByEmail(list){const m=new Map(); for(const x of list||[]){const k=String(x.email||'').toLowerCase().trim(); if(k) m.set(k,x);} return m;}

## L1-p10-find-or-throw: findStudentById — NotFoundError

Specificație:

`findStudentById(list, id)` → returnează studentul sau aruncă `NotFoundError`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export function findStudentById(list,id){const it=(list||[]).find(s=>String(s.id)===String(id)); if(!it) throw new NotFoundError('Student not found'); return it;}

## L1-p11-validation-report: Raport de validare (Ok/Err)

Specificație:

`validateStudent(obj)` → `{ ok:true, value }` sau `{ ok:false, error:'...' }` fără a arunca.

Soluție (esentialul):

export function validateStudent(o){if(!o||!o.id||!o.name||!o.email) return {ok:false, error:'missing'}; return {ok:true, value:{id:String(o.id),name:String(o.name).trim(),email:String(o.email).toLowerCase().trim(), interests:Array.isArray(o.interests)?o.interests.slice():[]}};}

## L1-p12-mixins: Mixins — Taggable

Specificație:

`Taggable(obj)` → adaugă `addTag/removeTag/listTags` pe un obiect (asumă câmpul `tags`).

Soluție (esentialul):

export function Taggable(obj){obj.addTag=(t)=>{obj.tags??=[]; if(!obj.tags.includes(t)) obj.tags.push(t);}; obj.removeTag=(t)=>{obj.tags=(obj.tags||[]).filter(x=>x!==t)}; obj.listTags=()=>[...(obj.tags||[])]; return obj;}

## L1-p13-brand-id: Branding ID cu Symbol

Specificație:

`brandId(obj, ns)` atașează `Symbol('id:'+ns)` pentru a „marca” un obiect fără coliziuni.

Soluție (esentialul):

export function brandId(obj, ns){const sym = Symbol('id:'+String(ns)); Object.defineProperty(obj, sym,{value:true}); return {obj, sym};}

## L1-p14-clone-shallow: cloneShallow — copiere superficială

Specificație:

`cloneShallow(o)` → copiază proprietăți proprii fără a duplica referințe interne.

Soluție (esentialul):

export function cloneShallow(o){return Object.assign({}, o);}

## L1-p15-memoize-basic: memoize — versiune de bază

Specificație:

`memoize(fn)` — cache pe cheie `JSON.stringify(args)`.

Soluție (esentialul):

export function memoize(fn){const cache=new Map(); return (...args)=>{const k=JSON.stringify(args); if(cache.has(k)) return cache.get(k); const v=fn(...args); cache.set(k,v); return v;};}

# L2 — Intermediar

## L2-p01-catalog-repo: Catalog + repository în memorie

Specificație:

`CatalogRepo` cu `add(entity)` idempotent și `getById(id)`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export class CatalogRepo {  
 constructor() { this.store=new Map(); }  
 add(e) { this.store.set(String(e.id), e); return e; }  
 getById(id) { const x=this.store.get(String(id)); if(!x) throw new NotFoundError('Not found'); return x; }  
 list() { return [...this.store.values()]; }  
}

## L2-p02-memoize-weak: memoizeWeak pe obiect‑cheie

Specificație:

`memoizeWeak(fn)` → cache `WeakMap(obj→rezultat)`.

Soluție (esentialul):

export function memoizeWeak(fn){const cache=new WeakMap(); return (obj,...rest)=>{ if(obj&&typeof obj==='object'){ if(cache.has(obj)) return cache.get(obj); const v=fn(obj,...rest); cache.set(obj,v); return v; } return fn(obj,...rest);} }

## L2-p03-errors-cause: Erori cu `cause` și mesaje prietenoase

Specificație:

Definește `AppError` + derivate și propagă `cause` în `wrap(fn)`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export async function wrap(fn){ try{ return await fn(); }catch(e){ throw new RemoteError('Remote failed', { cause:e }); } }

## L2-p04-directory-allsettled: DirectoryService cu Promise.allSettled

Specificație:

Încarcă 3 surse mock în paralel; la eșec, întoarce liste goale în câmpul aferent.

Soluție (esentialul):

const delay=ms=>new Promise(r=>setTimeout(r,ms));  
async function s(){await delay(5); return [{id:'s1'}];}  
async function c(){await delay(5); return [{id:'c1'}];}  
async function e(){await delay(5); return [{id:'e1'}];}  
export async function loadAll(){ const res=await Promise.allSettled([s(),c(),e()]); return {students:res[0].status==='fulfilled'?res[0].value:[], clubs:res[1].status==='fulfilled'?res[1].value:[], events:res[2].status==='fulfilled'?res[2].value:[]}; }

## L2-p05-report-topk: ReportService.topInterests determinist

Specificație:

Calculează top‑K interese (insensibil la caz + whitespace) cu tie‑break alfabetic.

Soluție (esentialul):

export function topInterests(students,k=3){const m=new Map(); for(const s of students||[]){for(const it of (s.interests||[])){const key=String(it).toLowerCase().trim(); if(!key) continue; m.set(key,(m.get(key)||0)+1);}} const arr=[...m.entries()].map(([key,count])=>({key,count})); arr.sort((a,b)=> b.count - a.count || a.key.localeCompare(b.key,'ro')); return arr.slice(0,k);}

## L2-p06-adapter-dto: Adapter DTO din JSON cu whitelisting

Specificație:

`adaptStudent(json)` extrage doar câmpurile permise, ignorând `\_\_proto\_\_/constructor`.

Soluție (esentialul):

export function adaptStudent(j){const o={id:j?.id, name:j?.name, email:j?.email, interests:Array.isArray(j?.interests)?j.interests:[]}; return o;}

## L2-p07-async-iterator: Async iterator peste evenimente (batchuri)

Specificație:

`makeAsyncEvents(batches)` → obiect cu `Symbol.asyncIterator` care produce evenimentele pe rând.

Soluție (esentialul):

export function makeAsyncEvents(batches){return { async \*[Symbol.asyncIterator](){ for(const b of batches){ for(const e of b){ yield e; } } } }}

## L2-p08-decorator-logger: Decorator funcțional de logging/timing

Specificație:

`timed(fn)` → întoarce o funcție care returnează `{ value, ms }`.

Soluție (esentialul):

export function timed(fn){return async (...args)=>{const t=Date.now(); const v=await fn(...args); return {value:v, ms: Date.now()-t};};}

## L2-p09-idempotent-add: Idempotent add în repo

Specificație:

`addOnce(repo, entity)` — nu adăuga dacă `id` există deja.

Soluție (esentialul):

export function addOnce(repo, e){ if(repo.store?.has(String(e.id))) return repo.getById(e.id); return repo.add(e);}

## L2-p10-schema-check: Validator simplu de „schemă”

Specificație:

`validateShape(obj, shape)` — shape: `{ field: 'string'|'number'|'array' }`.

Soluție (esentialul):

export function validateShape(obj, shape){ for(const [k,t] of Object.entries(shape)){ const v=obj?.[k]; const ok = (t==='array') ? Array.isArray(v) : typeof v===t; if(!ok) return {ok:false, error:k}; } return {ok:true}; }

## L2-p11-compose-pipeline: Compose de funcții pure pentru agregare

Specificație:

`compose(...fns)` → `f(g(h(x)))`.

Soluție (esentialul):

export const compose=(...fns)=>(x)=>fns.reduceRight((v,f)=>f(v),x);

## L2-p12-hasinstance: Personalizează instanceof (Symbol.hasInstance)

Specificație:

Clasa `Email` să accepte stringuri `a@b` ca instanțe logice: `'a@b' instanceof Email` → true.

Soluție (esentialul):

export class Email{ static [Symbol.hasInstance](x){ return typeof x==='string' && x.includes('@'); } }

## L2-p13-collator-sort: Sortare cu Intl.Collator('ro')

Specificație:

`sortAlphaRo(arr)` sortează alfabetizat, cu diacritice corect.

Soluție (esentialul):

export function sortAlphaRo(arr){ const c=new Intl.Collator('ro'); return [...arr].sort((a,b)=>c.compare(String(a),String(b))); }

## L2-p14-deep-freeze: deepFreeze utilitar (simplu)

Specificație:

`deepFreeze(o)` care îngheață recursiv obiectele simple (evită cicluri).

Soluție (esentialul):

export function deepFreeze(o, seen=new Set()){ if(!o||typeof o!=='object'||seen.has(o)) return o; seen.add(o); Object.freeze(o); for(const v of Object.values(o)) deepFreeze(v, seen); return o; }

## L2-p15-memo-ttl: memoizeWithTTL

Specificație:

`memoizeWithTTL(fn, ttlMs)` ca în Partea 2.

Soluție (esentialul):

export function memoizeWithTTL(fn, ttlMs=50){ const cache=new Map(); return (...args)=>{ const now=Date.now(); const k=JSON.stringify(args); const h=cache.get(k); if(h && (now-h.t)<ttlMs) return h.v; const v=fn(...args); cache.set(k,{v,t:now}); return v; }; }

# L3 — Avansat

## L3-p01-retry-backoff: Retry cu backoff și clasificare erori

Specificație:

`withRetry(fn, {retries, baseMs, classify})` care oprește pentru `non-retryable`.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export async function withRetry(fn, { retries=2, baseMs=10, classify }={}) {  
 let last;  
 for(let i=0;i<=retries;i++){ try{ return await fn(); }catch(e){ if(classify && classify(e)==='non-retryable') throw e; last=e; await new Promise(r=>setTimeout(r, baseMs\*(2\*\*i))); } }  
 throw last;  
}

## L3-p02-timeout: Timeout wrapper pentru promisiuni

Specificație:

`withTimeout(promise, ms)` aruncă `TimeoutError` la depășire.

Soluție (esentialul):

export class AppError extends Error {  
 constructor(message, { cause } = {}) { super(message); this.name = this.constructor.name; if (cause) this.cause = cause; }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class TimeoutError extends AppError {}  
  
export function withTimeout(promise, ms){ return new Promise((resolve,reject)=>{ const t=setTimeout(()=>reject(new TimeoutError('Timeout')), ms); promise.then(v=>{clearTimeout(t); resolve(v);},e=>{clearTimeout(t); reject(e);}); }); }

## L3-p03-semaphore: Semafor pentru limitarea concurenței

Specificație:

`makeSemaphore(n)` cu `schedule(task)`.

Soluție (esentialul):

export function makeSemaphore(max=2){ let inFlight=0; const q=[]; const run=()=>{ if(inFlight>=max||!q.length) return; inFlight++; const {task,res,rej}=q.shift(); task().then(v=>{inFlight--; res(v); run();},e=>{inFlight--; rej(e); run();}); }; return { schedule(task){ return new Promise((res,rej)=>{ q.push({task,res,rej}); run(); }); } }; }

## L3-p04-abort: AbortController în flux async

Specificație:

Funcție `abortable(promise, signal)` care respinge la `abort()`.

Soluție (esentialul):

export function abortable(promise, signal){ return new Promise((resolve,reject)=>{ const on=()=>reject(new Error('Aborted')); signal?.addEventListener?.('abort', on, {once:true}); promise.then(resolve,reject); }); }

## L3-p05-circuit-breaker: Circuit breaker minimal

Specificație:

`makeBreaker({failThreshold, coolMs})` care „deschide” circuitul după erori repetate și „închide” după o pauză.

Soluție (esentialul):

export function makeBreaker({failThreshold=3, coolMs=50}={}){ let fails=0, openUntil=0; return async (fn)=>{ const now=Date.now(); if(now<openUntil) throw new Error('Open'); try{ const v=await fn(); fails=0; return v; }catch(e){ fails++; if(fails>=failThreshold){ openUntil=now+coolMs; } throw e; } }; }

## L3-p06-async-pipeline: Pipeline asincron cu routing de erori

Specificație:

`runPipeline(steps)` unde fiecare step este `async (ctx) => ctx`. Erorile annotate cu `ctx.errors`.

Soluție (esentialul):

export async function runPipeline(steps, ctx={}){ ctx.errors=[]; for(const s of steps){ try{ await s(ctx); }catch(e){ ctx.errors.push(e.message||String(e)); } } return ctx; }

## L3-p07-e2e-site: Playwright e2e — pagină statică StudentHub

Specificație:

Mic site static care afișează „Top interests”. Test E2E cu Playwright care verifică conținutul H1 și tabelul.

Soluție (esentialul):

// server.js  
import express from 'express';  
const app = express();  
app.use(express.static('public'));  
const PORT = process.env.PORT || 4173;  
app.listen(PORT, ()=> console.log('Server on http://localhost:'+PORT));

## L3-p08-pwa-workbox: PWA (Workbox) — precache „public/”

Specificație:

Generează un SW cu `workbox-build` pentru `public/\*` și un test smoke că fișierul `sw.js` e generat.

Soluție (esentialul):

// build-sw.mjs  
import {{ generateSW }} from 'workbox-build';  
const { count, size, warnings } = await generateSW({{  
 globDirectory: 'public',  
 globPatterns: ['\*\*/\*.\*'],  
 swDest: 'public/sw.js',  
 skipWaiting: true,  
 clientsClaim: true  
}});  
console.log('SW', {count, size, warnings});

## L3-p09-stable-hash: Hash stabil pentru chei de memo

Specificație:

`stableHash(x)` — serializare deterministă (chei sortate).

Soluție (esentialul):

export function stableHash(x){ if(x===null||typeof x!=='object') return JSON.stringify(x); const keys=Object.keys(x).sort(); const parts=keys.map(k=>k+':'+stableHash(x[k])); return '{'+parts.join(',')+'}'; }

## L3-p10-plugins: Pluginuri pentru ReportService

Specificație:

`makeReportService(plugins)` unde pluginurile sunt funcții `(data)=>partial` combinate într‑un obiect rezultat.

Soluție (esentialul):

export function makeReportService(plugins=[]){ return { run(data){ const out={}; for(const p of plugins){ Object.assign(out, p(data)); } return out; } }; }

## L3-p11-fault-injection: Fault injection pentru retry

Specificație:

`sometimesFail(p)` returnează o funcție care eșuează cu probabilitate p; testează `withRetry`.

Soluție (esentialul):

export function sometimesFail(p){ let n=0; return async (v=1)=>{ if(Math.random()<p && n++<2) throw new Error('flaky'); return v; }; }

## L3-p12-randomized-tests: Teste randomizate controlate

Specificație:

Generator de date de test pentru `topInterests` cu semințe deterministe.

Soluție (esentialul):

export function rng(seed=1){ let s=seed>>>0; return ()=> (s = (s\*1664525 + 1013904223)>>>0) / 2\*\*32; }  
export function genStudents(n, seed=1){ const r=rng(seed); const vals=['Web','AI','UX']; const out=[]; for(let i=0;i<n;i++){ out.push({interests:[ vals[Math.floor(r()\*vals.length)] ]}); } return out; }

## L3-p13-async-queue: Coada asincronă cu ratelimiting

Specificație:

`makeQueue({concurrency, intervalMs})` execută taskurile cu un delay minim între starturi.

Soluție (esentialul):

export function makeQueue({concurrency=1, intervalMs=10}={}){ let inflight=0, lastStart=0; const q=[]; const run=async()=>{ if(!q.length || inflight>=concurrency) return; const now=Date.now(); const wait=Math.max(0, intervalMs - (now-lastStart)); await new Promise(r=>setTimeout(r, wait)); lastStart=Date.now(); inflight++; const {task,res,rej}=q.shift(); task().then(v=>{inflight--; res(v); run();},e=>{inflight--; rej(e); run();}); }; return { add(task){ return new Promise((res,rej)=>{ q.push({task,res,rej}); run(); }); } }; }

## L3-p14-e2e-table: Playwright: verifică tabel sortat

Specificație:

Test E2E care verifică ordinea alfabetică a intereselor într‑o pagină de demo.

Soluție (esentialul):

// server.js  
import express from 'express';  
const app = express();  
app.use(express.static('public'));  
const PORT = process.env.PORT || 4174;  
app.listen(PORT, ()=> console.log('Server on http://localhost:'+PORT));

## L3-p15-e2e-form: Playwright: mini‑formular validare

Specificație:

Pagină cu formular de email; test E2E care verifică mesajul „invalid email”.

Soluție (esentialul):

// server.js  
import express from 'express';  
const app = express();  
app.use(express.static('public'));  
const PORT = process.env.PORT || 4175;  
app.listen(PORT, ()=> console.log('Server on http://localhost:'+PORT));

# Bibliografie (APA 7, cu DOI)

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

Michie, D. (1968). “Memo” functions and machine learning. Nature, 218(5136), 19–22. https://doi.org/10.1038/218019a0

Loring, M. C., et al. (2017). Semantics of asynchronous JavaScript. ACM SIGPLAN Notices, 52(11), 51–62. https://doi.org/10.1145/3170472.3133846

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

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