**Seminarul 3 — Partea 2 (Laborator practic, extins)**

# Seminarul 3 — Partea 2: Laborator practic (progresiv, extins)

\*\*Tema unificatoare:\*\* \*StudentHub Core\* — un set coerent de module (\*\*ES modules\*\*) concepute să modeleze entități (`Student`, `Club`, `Event`), să aplice \*\*invariante\*\* robuste și \*\*ierarhii de erori\*\* utile domeniului, să realizeze \*\*memoization\*\* corectă (chei stabile, `Map`/`WeakMap`, opțional `TTL`), și să orchestreze fluxuri \*\*asincrone\*\* cu \*\*async/await\*\* (inclusiv `timeout`, `retry` cu \*exponential backoff\*, `AbortController` și limitare a concurenței prin semafor). Laboratorul este \*\*progresiv\*\* (L1→L2→L3), include \*\*worksheet\*\* (cerințe + checklist), \*\*starter code\*\* și \*\*unit tests\*\* \*\*side‑by‑side\*\* în \*\*Vitest\*\* și \*\*Jest\*\*, plus \*AI‑assist (VSL)\* pentru a modela cum integrăm GitHub Copilot/ChatGPT/Mistral/Claude în proces.

> \*\*Hook realist.\*\* Imaginează‑ți că primești, vineri seara, o cerință: „Luni la 8:00, prezintă un raport despre \*top‑interests\* ale studenților și evenimentele asociate, indiferent că un API extern e căzut”. Ai nevoie de: (1) obiecte cu \*\*invariante\*\* (nu poți raporta pe date stricate), (2) \*\*memoization\*\* (raportul e scump — îl refolosești dacă inputul e același), (3) \*\*async\*\* robust (limite de timp, reîncercări, anulări), (4) \*\*teste\*\* care să verifice că nu livrezi surprize, duminică seara, la 2 AM.

---

## 0) Principii de lucru & evaluare

1. \*\*Contracte peste implementări.\*\* Scriem întâi „gardurile” (invariante, erori tipizate), apoi funcțiile.

2. \*\*Puritate locală.\*\* Funcțiile care calculează rapoarte nu trebuie să cunoască \*I/O\*. Datele intră, rezultatele ies.

3. \*\*Imutabilitate superficială.\*\* Obiectele de domeniu (`Student`, `Club`, `Event`) se \*„ștampilează”\* (`Object.freeze`) după validare.

4. \*\*Async previzibil.\*\* Evităm „așteptarea fără capăt”: `withTimeout`, `withRetry`, `AbortController`, semafor pentru „rate‑limit”.

5. \*\*Testare în oglindă.\*\* Aceleași contracte sunt testate în \*\*Vitest\*\* și \*\*Jest\*\*, ca verificare de portabilitate a suitei.

\*\*Definiția de finalizare (Definition of Done):\*\*

- [ ] Toate testele \*\*Vitest\*\*/\*\*Jest\*\* sunt verzi.

- [ ] Invariantele entităților sunt aplicate; erorile sunt tipizate (mesaje + `cause`).

- [ ] Memoizationul produce \*\*cache‑hit\*\* (aceeași referință) când inputul e identic logic; are opțional `TTL`.

- [ ] Fluxurile asincrone gestionează `timeout`, `retry/backoff`, `abort`, \*semafor\*.

- [ ] Cod modular, comentat, cu README și instrucțiuni de rulare.

---

## 1) Worksheet — Cerințe & Checklist

\*\*Cerință centrală:\*\* construiește „StudentHub Core” astfel încât \*\*următorul scenariu\*\* să ruleze determinist:

1. `DirectoryService.loadAll()` încarcă \*\*concurent\*\* (mock) `students`, `clubs`, `events` și compune un „snapshot” coerent.

2. `ReportService.topInterests(students, k)` calculează \*\*determinist\*\* top‑K interese, \*\*memoizat\*\*.

3. \*\*Erori\*\*: dacă o „sursă” este lentă (sau cade), avem `withTimeout` și `withRetry` cu \*exponential backoff\*; există opțiune de `AbortController`.

4. \*\*Concurență\*\*: pentru „surse” \*rate‑limited\*, limităm accesul cu un „mini‑semafor”.

5. \*\*Testare\*\*: suite în oglindă (Vitest/Jest) pentru entități, erori, memoization, async & concurrency.

\*\*Checklist (bază → avansat):\*\*

- [ ] Entități cu invariante (L1).

- [ ] Ierarhie erori: `AppError`, `ValidationError`, `NotFoundError`, `RemoteError`, `TimeoutError` (L1).

- [ ] `ReportService.topInterests` pură + \*\*memoization\*\* (L2).

- [ ] `DirectoryService.loadAll` concurent; \*fallback\* parțial (L2).

- [ ] `withTimeout`, `withRetry/backoff`, `makeSemaphore` + `AbortController` (L3).

- [ ] Teste negative pentru fiecare mecanism (L3).

> \*\*AI‑assist (VSL).\*\* Începe cu \*\*Verify\*\*: listează 12 \*edge‑cases\* pentru `Student`/`Club`/`Event`. Treci la \*\*Specify\*\*: impune semnături + erori tipizate. În \*\*Learn\*\*, cere refactor pentru `WeakMap`/`TTL` și compară trade‑off‑urile.

---

## 2) Inițializare proiect (scaffold)

\*\*Variantă npm (standalone):\*\*

mkdir s3-lab && cd s3-lab  
npm init -y  
npm i -D vitest jest babel-jest @babel/preset-env

\*\*`package.json` (scripturi):\*\*

{  
 "name": "s3-lab",  
 "type": "module",  
 "version": "1.0.0",  
 "scripts": {  
 "test:vitest": "vitest run --reporter verbose",  
 "test:jest": "jest --runInBand",  
 "test": "npm run test:vitest && npm run test:jest"  
 },  
 "devDependencies": {  
 "vitest": "^1.6.0",  
 "jest": "^29.7.0",  
 "babel-jest": "^29.7.0",  
 "@babel/preset-env": "^7.25.0"  
 }  
}

\*\*Config Jest & Babel:\*\*

// jest.config.cjs  
module.exports = {  
 testEnvironment: 'node',  
 transform: { '^.+\\.m?js$': 'babel-jest' },  
 extensionsToTreatAsEsm: ['.js']  
};

// babel.config.cjs  
module.exports = {  
 presets: [['@babel/preset-env', { targets: { node: 'current' } }]]  
};

\*\*Config Vitest:\*\*

// vitest.config.ts  
import { defineConfig } from 'vitest/config'  
export default defineConfig({  
 test: {  
 environment: 'node',  
 include: ['tests/vitest/\*\*/\*.test.js']  
 }  
})

> \*\*Monorepo (PNPM workspaces)\*\* este livrat în arhiva separată. Pentru laboratorul local, varianta \*standalone\* este suficientă.

---

## 3) Starter code (explicat)

### 3.1) Erori (taxonomie aplicată)

// src/errors.js  
export class AppError extends Error {  
 constructor(message, { cause } = {}) {  
 super(message);  
 this.name = this.constructor.name;  
 if (cause) this.cause = cause;  
 Error.captureStackTrace?.(this, this.constructor);  
 }  
}  
export class ValidationError extends AppError {}  
export class NotFoundError extends AppError {}  
export class RemoteError extends AppError {}  
export class TimeoutError extends AppError {}

\*\*De ce:\*\* păstrăm `name` predictibil, adăugăm `cause` (diagnosticabil), avem o bază (`AppError`) pentru \*catch‑all\* controlat.

### 3.2) Entități (immutabile superficial, invariante)

// src/entities.js  
import { ValidationError } from './errors.js';  
  
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); // shallow immutability  
 }  
 toString() { return `[Student ${this.id}] ${this.name} <${this.email}>`; }  
 get [Symbol.toStringTag]() { return 'Student'; }  
}  
  
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'; }  
}  
  
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'; }  
}

\*\*De ce:\*\* \*freeze\* previne „aliasingul” care rupe memoizationul; `toStringTag` oferă „branding”.

### 3.3) Memoization (Map, WeakMap, TTL)

// src/services/memo.js  
export function memoize(fn) {  
 const cache = new Map();  
 return function (...args) {  
 const key = JSON.stringify(args);  
 if (cache.has(key)) return cache.get(key);  
 const out = fn.apply(this, args);  
 cache.set(key, out);  
 return out;  
 };  
}  
  
export function memoizeWeak(fn) {  
 const cache = new WeakMap();  
 return function (obj, ...rest) {  
 if (obj && typeof obj === 'object') {  
 if (cache.has(obj)) return cache.get(obj);  
 const out = fn.call(this, obj, ...rest);  
 cache.set(obj, out);  
 return out;  
 }  
 return fn.call(this, obj, ...rest);  
 };  
}  
  
export function memoizeWithTTL(fn, ttlMs = 60000) {  
 const cache = new Map();  
 return function (...args) {  
 const now = Date.now();  
 const key = JSON.stringify(args);  
 const hit = cache.get(key);  
 if (hit && (now - hit.t) < ttlMs) return hit.v;  
 const v = fn.apply(this, args);  
 cache.set(key, { v, t: now });  
 return v;  
 };  
}

\*\*De ce:\*\* `Map` pentru chei serializate; `WeakMap` pentru chei‑obiect (evită scurgeri); `TTL` când datele „expiră”.

### 3.4) Raport (funcție pură, deterministă, memoizată)

// src/services/report.js  
import { memoize } from './memo.js';  
  
function topInterestsImpl(students, k = 3) {  
 const freq = new Map();  
 for (const s of students || []) {  
 for (const it of s.interests || []) {  
 const key = String(it).toLowerCase().trim();  
 if (!key) continue;  
 freq.set(key, (freq.get(key) || 0) + 1);  
 }  
 }  
 const arr = [...freq.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);  
}  
  
export const ReportService = {  
 topInterests: memoize(topInterestsImpl) // determinist + cache  
};

\*\*De ce:\*\* pură → testabilă; deterministă → \*memoization\* corectă; ordonare stabilă (tie‑break alfabetic).

### 3.5) Async utilities (timeout, retry/backoff, semafor)

// src/services/concurrency.js  
import { TimeoutError } from '../errors.js';  
  
export function withTimeout(promise, ms) {  
 return new Promise((resolve, reject) => {  
 const t = setTimeout(() => reject(new TimeoutError(`Timeout after ${ms}ms`)), ms);  
 promise.then(v => { clearTimeout(t); resolve(v); }, e => { clearTimeout(t); reject(e); });  
 });  
}  
  
export async function withRetry(fn, { retries = 2, baseMs = 50, classify } = {}) {  
 let lastErr;  
 for (let i = 0; i <= retries; i++) {  
 try {  
 return await fn();  
 } catch (e) {  
 if (classify && classify(e) === 'non-retryable') throw e;  
 lastErr = e;  
 await new Promise(r => setTimeout(r, baseMs \* (2 \*\* i)));  
 }  
 }  
 throw lastErr;  
}  
  
export function makeSemaphore(max = 3) {  
 let inFlight = 0;  
 const queue = [];  
 const runNext = () => {  
 if (inFlight >= max || queue.length === 0) return;  
 inFlight++;  
 const { task, resolve, reject } = queue.shift();  
 task().then(v => { inFlight--; resolve(v); runNext(); }, e => { inFlight--; reject(e); runNext(); });  
 };  
 return {  
 schedule(task) {  
 return new Promise((resolve, reject) => {  
 queue.push({ task, resolve, reject });  
 runNext();  
 });  
 }  
 };  
}

\*\*De ce:\*\* „taie așteptarea”, reîncearcă cu pauze crescătoare, limitează cererile concurente.

### 3.6) Directory service (mock concurent + abort)

// src/services/directory.js  
import { Student, Club, Event } from '../entities.js';  
import { TimeoutError } from '../errors.js';  
  
const delay = (ms) => new Promise(r => setTimeout(r, ms));  
  
async function fetchStudentsMock() {  
 await delay(20);  
 return [  
 new Student({ id: 's1', name: 'Ana Pop', email: 'ana@x.org', interests: ['AI','Web'] }),  
 new Student({ id: 's2', name: 'Mihai I.', email: 'mihai@x.org', interests: ['Finance','Web'] }),  
 new Student({ id: 's3', name: 'Ioana T.', email: 'ioana@x.org', interests: ['Web','UX'] }),  
 ];  
}  
async function fetchClubsMock() {  
 await delay(10);  
 return [  
 new Club({ id: 'c1', title: 'AI Club', tags: ['AI','ML'] }),  
 new Club({ id: 'c2', title: 'Web Club', tags: ['Web','UX'] }),  
 ];  
}  
async function fetchEventsMock() {  
 await delay(5);  
 return [  
 new Event({ id: 'e1', label: 'Hackathon', when: Date.now(), clubId: 'c2' }),  
 ];  
}  
  
export const DirectoryService = {  
 async loadAll({ signal } = {}) {  
 const abortable = (p) => new Promise((resolve, reject) => {  
 const onAbort = () => reject(new TimeoutError('Aborted'));  
 signal?.addEventListener?.('abort', onAbort, { once: true });  
 p.then(resolve, reject);  
 });  
 const results = await Promise.allSettled([  
 abortable(fetchStudentsMock()),  
 abortable(fetchClubsMock()),  
 abortable(fetchEventsMock())  
 ]);  
 const [students, clubs, events] = results.map(r => r.status === 'fulfilled' ? r.value : []);  
 return { students, clubs, events };  
 }  
};

\*\*De ce:\*\* `allSettled` → raport parțial, util didactic; `AbortController` opțional.

---

## 4) Exerciții L1 (fundamental) — pas cu pas

\*\*Obiectiv:\*\* entități corecte + ierarhie minimă de erori + teste de invariante.

1. \*\*Implementează\*\* `Student`, `Club`, `Event` ca mai sus. Explică în comentarii de ce `Object.freeze` este \*shallow\* și suficient aici.

2. \*\*Adaugă\*\* `toString` și `Symbol.toStringTag` — confirmă în test că `Object.prototype.toString.call(new Student(...))` conține `"[object Student]"`.

3. \*\*Verifică\*\* că la câmpuri lipsă se aruncă `ValidationError`.

4. \*\*Testează\*\* în \*\*Vitest\*\* și \*\*Jest\*\* (oglindă) aceleași contracte.

\*\*Test (Vitest):\*\*

// tests/vitest/entities.test.js  
import { describe, it, expect } from 'vitest';  
import { Student } from '../../src/entities.js';  
import { ValidationError } from '../../src/errors.js';  
  
describe('Student entity', () => {  
 it('frozen + normalized email', () => {  
 const s = new Student({ id: 1, name: 'Ana', email: 'ANA@X.ORG', interests: ['Web'] });  
 expect(Object.isFrozen(s)).toBe(true);  
 expect(s.email).toBe('ana@x.org');  
 });  
 it('invalid → ValidationError', () => {  
 expect(() => new Student({ name: 'x' })).toThrow(ValidationError);  
 });  
});

\*\*Test (Jest):\*\*

// tests/jest/entities.jest.test.js  
import { describe, it, expect } from '@jest/globals';  
import { Student } from '../../src/entities.js';  
import { ValidationError } from '../../src/errors.js';  
  
describe('Student entity (Jest)', () => {  
 it('frozen + normalized', () => {  
 const s = new Student({ id: 1, name: 'Ana', email: 'ANA@X.ORG', interests: ['Web'] });  
 expect(Object.isFrozen(s)).toBe(true);  
 expect(s.email).toBe('ana@x.org');  
 });  
 it('invalid → ValidationError', () => {  
 expect(() => new Student({ name: 'x' })).toThrow(ValidationError);  
 });  
});

\*\*Explicații:\*\* Invariantele devin \*\*teste\*\*. Învățăm să separăm modelul (entitățile) de I/O, setând scena pentru \*memoization\*.

---

## 5) Exerciții L2 (intermediar) — memoization & încărcare concurentă

\*\*Obiectiv:\*\* `ReportService.topInterests` (pur + memoizat) și `DirectoryService.loadAll()` (concurent + tolerant).

1. \*\*Pornește\*\* de la implementările din \*starter code\*.

2. \*\*Scrie\*\* un test de \*\*cache‑hit\*\* (același input → aceeași referință la rezultat).

3. \*\*Adaugă\*\* o variantă cu `memoizeWeak` (cheie obiect) și \*\*discută\*\* când e potrivită (obiect‑rădăcină identic).

4. \*\*Extinde\*\* testele: ordonare stabilă, \*tie‑break\* alfabetic; insensibilitate la caz/whitespace în interese.

\*\*Test (Vitest) – memo & ordonare:\*\*

// tests/vitest/memo.test.js  
import { describe, it, expect } from 'vitest';  
import { ReportService } from '../../src/services/report.js';  
  
describe('ReportService.topInterests (memo + determinism)', () => {  
 it('cache-hit and deterministic order', () => {  
 const students = [  
 { interests: ['Web','AI'] }, { interests: [' Web ' ] }, { interests: ['UX'] }  
 ];  
 const r1 = ReportService.topInterests(students, 2);  
 const r2 = ReportService.topInterests(students, 2);  
 expect(r2).toBe(r1); // cache hit: aceeași referință  
 expect(r1[0].key).toBe('web'); // tie-break corect dacă egalități  
 });  
});

\*\*Explicații:\*\* Deteminism + imutabilitate superficială → \*memoization\* predictibilă.

---

## 6) Exerciții L3 (avansat) — timeout, retry/backoff, abort, semafor, TTL

\*\*Obiectiv:\*\* comportament „de producție” în miniatură.

1. \*\*`withRetry`\*\* cu parametru `classify(e)` → dacă eroarea e \*non‑retryable\*, nu o reîncercăm.

2. \*\*`withTimeout`\*\* cu mesaje prietenoase + includerea \*elapsed ms\* (opțional).

3. \*\*`makeSemaphore(n)`\*\* → verifică în test că `maxSeen` nu depășește `n`.

4. \*\*`AbortController`\*\* → confirmă că `DirectoryService.loadAll({ signal })` respinge la `abort()`.

5. \*\*`memoizeWithTTL`\*\* → testează că după TTL rezultatul e recalculat (nu \*\*aceeași referință\*\*).

\*\*Test (Vitest) – async & concurrency:\*\*

// tests/vitest/async.test.js  
import { describe, it, expect } from 'vitest';  
import { withTimeout, withRetry, makeSemaphore } from '../../src/services/concurrency.js';  
  
describe('concurrency helpers (L3)', () => {  
 it('withRetry eventually succeeds (exponential backoff)', async () => {  
 let n = 0;  
 const fn = async () => { if (n++ < 1) throw new Error('fail'); return 42; };  
 const val = await withRetry(fn, { retries: 2, baseMs: 1 });  
 expect(val).toBe(42);  
 });  
  
 it('withTimeout rejects when slow', async () => {  
 const slow = new Promise(r => setTimeout(() => r('late'), 30));  
 await expect(withTimeout(slow, 5)).rejects.toBeInstanceOf(Error);  
 });  
  
 it('semaphore limits inflight', async () => {  
 const sem = makeSemaphore(1);  
 let concurrent = 0, maxSeen = 0;  
 const task = () => new Promise(r => {  
 concurrent++; maxSeen = Math.max(maxSeen, concurrent);  
 setTimeout(() => { concurrent--; r(1); }, 5);  
 });  
 await Promise.all([sem.schedule(task), sem.schedule(task), sem.schedule(task)]);  
 expect(maxSeen).toBe(1);  
 });  
});

\*\*Test (Jest) – retry (oglindă):\*\*

// tests/jest/async.jest.test.js  
import { describe, it, expect } from '@jest/globals';  
import { withTimeout, withRetry, makeSemaphore } from '../../src/services/concurrency.js';  
  
describe('concurrency helpers (Jest)', () => {  
 it('retry works', async () => {  
 let n = 0;  
 const fn = async () => { if (n++ < 1) throw new Error('fail'); return 7; };  
 const val = await withRetry(fn, { retries: 2, baseMs: 1 });  
 expect(val).toBe(7);  
 });  
});

\*\*Test (Vitest) – memoizeWithTTL:\*\*

// tests/vitest/memo-ttl.test.js  
import { describe, it, expect } from 'vitest';  
import { memoizeWithTTL } from '../../src/services/memo.js';  
  
describe('memoizeWithTTL', () => {  
 it('recomputes after TTL', async () => {  
 let calls = 0;  
 const fn = (x) => { calls++; return { v: x \* 2 }; };  
 const m = memoizeWithTTL(fn, 5);  
 const a = m(2);  
 const b = m(2);  
 expect(b).toBe(a); // cache hit  
 await new Promise(r => setTimeout(r, 8));  
 const c = m(2);  
 expect(c).not.toBe(a); // recalculat după TTL  
 expect(c.v).toBe(4);  
 expect(calls).toBe(2);  
 });  
});

\*\*Explicații:\*\* TTL e util când inputul e logic același, dar vrem „prospețime” (ex. raport periodic).

---

## 7) AI‑assist (VSL): cum integrăm sigur completările

- \*\*Verify:\*\* „Generează 12 \*edge‑cases\* pentru entități (diacritice, whitespace, email invalid, liste `null`/`[]`), apoi suite de teste în Vitest/Jest.”

- \*\*Specify:\*\* „Păstrează semnăturile existente; nu schimba numele claselor de erori; `topInterests(students, k)` trebuie să fie \*pură\* și deterministă.”

- \*\*Learn:\*\* „Propune o variantă `memoizeWeak` și explică când e \*unsafe\*; adaugă `TTL` cu test.”

- \*\*Practici:\*\* Review manual al completărilor; nu accepta refactor ce rupe contractele; păstrează aceleași teste „în oglindă”.

---

## 8) Troubleshooting (capcane uzuale)

- \*\*Mutabilitate după memoizare.\*\* Evită să editezi `interests` pe obiecte deja folosite ca intrare; altfel, rezultatele sunt „stale”. Creează entități noi.

- \*\*Chei de cache fragile.\*\* `JSON.stringify` e sensibil la ordinea proprietăților; pentru obiecte mari → chei bazate pe hashing stabil sau `WeakMap`.

- \*\*Retry neatent.\*\* Reîncearcă doar erori \*\*retryable\*\*; altfel înrăutățești situația.

- \*\*`try/catch` care înghite eroarea.\*\* Propagă cu `cause` și \*logging\* util.

- \*\*Divergență între suite.\*\* Menține contractele identice în Vitest și Jest; nu „adaptezi” testele separat.

---

## 9) README (pentru proiectul de laborator)

\*\*Instalare & rulare:\*\*

npm i  
npm run test # rulează Vitest + Jest  
# sau doar unul:  
npm run test:vitest  
npm run test:jest

\*\*Structură:\*\*

s3-lab/  
 package.json  
 src/  
 entities.js  
 errors.js  
 services/  
 directory.js  
 memo.js  
 report.js  
 concurrency.js  
 tests/  
 vitest/\*.test.js  
 jest/\*.jest.test.js  
 README.md

\*\*Extensii recomandate:\*\*

- Adaugă `memoizeWeak` în `ReportService` când inputul este un \*\*obiect‑rădăcină\*\* stabil.

- Introdu `TTL` acolo unde „prospezimea” e critică.

- Transformă `DirectoryService` într‑un \*adapter\* peste `fetch` real, cu `withRetry/withTimeout`.

---

## 10) Rubrică de evaluare (orientativ)

- \*\*Corectitudine & contracte (35%)\*\* — invariante, erori tipizate, determinism.

- \*\*Async & robustețe (25%)\*\* — timeout, retry, abort, concurență.

- \*\*Memoization & performanță (20%)\*\* — chei stabile, evitarea aliasingului, complexitate.

- \*\*Testare (15%)\*\* — acoperire și \*edge‑cases\*, Vitest/Jest în oglindă.

- \*\*Calitatea codului (5%)\*\* — modularizare, claritate, comentarii și README.

---

## 11) 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., 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

- 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

- Ungar, D., & Smith, R. B. (1987). \*Self: The power of simplicity\*. In \*\*OOPSLA ‘87\*\*. ACM. https://doi.org/10.1145/38765.38828