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

Seminarul 2 — Partea 2 (Laborator, extins): „De la micro‑utilitare la pipeline robust”

Context și obiectiv

În această parte practică vom construi, testa și documenta un nucleu de utilitare funcționale pentru prelucrarea listelor (array‑uri) și vom compune aceste utilitare într‑un pipeline declarativ pentru un raport simplu (StudentHub Insights). Vom lucra „test‑first”, cu \*\*Vitest\*\* și \*\*Jest\*\* în oglindă, și vom separa clar logica pură de I/O. Laboratorul e gândit progresiv: începe cu exerciții focalizate (implementări de utilitare), continuă cu compoziții (chaining / pipe), și se încheie cu integrarea într‑un mini‑CLI care produce rapoarte în două formate (table/json).

Învățare asistată de AI

Vom folosi GitHub Copilot (sub VSL: Verify/Tests → Specify/Contracts → Learn/Lint) sau, alternativ, ChatGPT/Mistral/Claude, pentru: (1) brainstorming de edge‑cases și generare de teste, (2) schițe de implementări păstrând contractele, (3) propuneri de refactor. Regula de aur: \*\*nimic nu intră în main fără teste verzi\*\*.

Rezultatul final

– o mini‑bibliotecă cu utilitare pure: `groupBy`, `countBy`, `keyBy`, `partition`, `uniqBy`, `differenceBy`, `intersectionBy`, `chunk`, `zip`, `unzip`, `flatten`, `compact`, `pipe`, `compose`, `tap`;

– un set de predicate (byFaculty, hasInterest, isValidRecord);

– sortări deterministe (byCountThenAlpha);

– un adaptor de normalizare (normalizeRecord, parseCSV/JSON);

– un runner CLI (`run(argv, io)`) cu opțiuni minimale (`--limit`, `--format`, `--faculty`, `--has-interest`);

– suite de teste „în oglindă” (Vitest/Jest).

Structura de lucru

Proiectul însoțitor (ZIP) este pregătit să ruleze imediat. În această fișă, vei găsi: (A) Worksheet (cerință + checklist), (B) Pașii de lucru (progresivi), (C) Starter code, (D) Unit tests, (E) Note și sfaturi AI‑assist.

A) Worksheet — cerință + checklist

Cerință (scurt)

1. Implementează utilitarele menționate, ca funcții pure („no I/O inside”), în `src/lib/collect.js`.

2. Scrie predicate și sortere în fișierele dedicate; normalizează datele de intrare în `normalize.js`.

3. Construiește un pipeline parametric pentru top‑K interese și distribuția pe facultăți.

4. Asigură suite în oglindă: aceleași contracte în Vitest și Jest.

5. Documentează deciziile (README) și asigură invarianta „determinism la egalități”.

Checklist (minim de trecere)

– [ ] Toate utilitarele sunt pure și testate (liste goale, un element, multiplu; `null/undefined`).

– [ ] `uniqBy` tratează duplicate conform politicii (primul câștigător sau ultimul).

– [ ] `differenceBy`/`intersectionBy` funcționează pe chei proiectate și pe obiecte.

– [ ] Comparatorul `byCountThenAlpha` e determinist: `count` desc, apoi alfabetic (locale ro).

– [ ] Pipeline‑ul `map/flatMap/filter/reduce` pentru top‑K e lizibil și parametric (`limit`, `faculty`, `has-interest`).

– [ ] CLI‑ul funcționează: `--format table|json`; codurile de ieșire sunt corecte (0 OK, 1 Usage/Error).

– [ ] Suitele Vitest & Jest trec integral; `npm test` este verde.

B) Pașii de lucru (progresivi)

Pasul 0 — Setup local

– Descarcă arhiva ZIP `s2p2-lab.zip` și execută:

`npm i` → `npm test` → `npm run dev`. Verifică outputul.

– Deschide `src/lib/collect.js` și `tests/\*`. Observă contractele și comenzile NPM (`test:vitest`, `test:jest`).

Pasul 1 — Implementarea utilitarelor de bază

– Începe cu `groupBy`, `countBy`, `uniqBy`, `partition`. Scop: să treacă testele de bază (`collect.test.\*`).

– Fixează contractele: ce chei acceptă `groupBy`? cum tratează `uniqBy` `null` la cheie?

Pasul 2 — Operații pe mulțimi și segmentare

– `differenceBy`, `intersectionBy`, `chunk`, `zip`, `unzip`, `flatten`, `compact`. Gândește „zero–unu–mulți” și cazuri limită.

– Rulează din nou testele; adaugă în README avertismente despre limitări (ex.: `zip` truncatează la lista cea mai scurtă).

Pasul 3 — Predicate, sortere, normalizări

– `byFaculty`, `hasInterest`, `isValidRecord` (criterii minime), `byCountThenAlpha` (determinist), `normalizeRecord` (canon e‑mail, facultate, interese).

– Extinde testele: adaugă cel puțin un caz „negativ” pentru `isValidRecord` (interese lipsă).

Pasul 4 — Pipeline și CLI

– În `cli-runner.js`, compune pașii: normalize → filter → opționale (faculty/interest) → top‑K (count + sort + slice) → distribuție pe facultăți → formatare `table|json`.

– Scrie „smoke tests” pentru CLI (fericit + usage + fișier lipsă).

Pasul 5 — Calitate și lizibilitate

– Rulează `npm run lint`; curăță naming‑ul și parametrii.

– În README, documentează politicile (chei duplicate la `keyBy`, ordonare stabilă).

Pasul 6 — AI‑assist (iterativ)

– Cere 6–10 teste noi pentru `uniqBy` (edge‑cases), apoi validează implementarea existentă.

– Cere propuneri de refactor pentru `byCountThenAlpha`, cu menținerea contractului; compară.

C) Starter code (fragmente cheie)

(Întregul cod este livrat în arhiva ZIP.) Mai jos sunt extrase reprezentative pentru înțelegere.

1) Exemplu utilitare (`collect.js`)

(vezi arhiva; implementările sunt pure; fără I/O; `pipe`, `compose`, `tap` pentru compoziție și „debug”.)

2) Predicate și sortere

`byFaculty(name)` întoarce un predicat; `hasInterest(label)` verifică lista de interese; `byCountThenAlpha` asigură determinism (desc pe `count`, apoi alfabetic).

3) Normalizare (CSV/JSON)

`normalizeRecord` asigură canonul minimal (email lowercase, faculty ∈ {CSIE,FABBV,REI,MAN}, interests listă normalizată).

4) CLI runner

`run(argv, io)` este injectabil (I/O „memorie” în teste), ceea ce permite testarea fără acces la disc.

D) Unit tests (Vitest & Jest, în oglindă)

– `tests/vitest/\*.test.js` și `tests/jest/\*.jest.test.js` conțin aceleași contracte.

– Convenție: teste de utilitare (collect), normalizări, predicate/sortere și „smoke tests” pentru CLI.

– Rularea `npm test` execută mai întâi Vitest, apoi Jest, pentru a sublinia „dublarea” de ecosisteme.

E) Sfaturi AI‑assist (prompts concrete)

– „Generează 10 edge‑cases pentru `uniqBy(emailCanon)` incluzând spații, casefolding, diacritice, `null`, liste goale.”

– „Rescrie `byCountThenAlpha` păstrând contractul, adăugând `Intl.Collator` pentru ordonare naturală; măsoară impactul.”

– „Propune 6 teste pentru `differenceBy` pe obiecte fără `id` (cheie derivată).”

– „Creează 2 exemple negative pentru `isValidRecord` (lipsă `interests`, `faculty` necunoscut).”

Rulare

– Standalone:

`npm i` → `npm test` → `npm run dev`

– Exemple:

`node bin/cli.js data/sample.csv --limit 3 --format json`

`node bin/cli.js data/sample.json --faculty CSIE --has-interest AI`

Evaluare

– Laboratorul este „verde” când: (1) toate testele trec în ambele suite; (2) raportul CLI funcționează pe `sample.csv` și `sample.json`; (3) README descrie politicile și edge‑cases; (4) lintul este curat.

Concluzie

Acest laborator materializează în cod filozofia \*Seminarului 2\*: funcții pure, compoziție declarativă și contracte explicite. Vei folosi utilitare generice pentru a construi pipeline‑uri robuste, vei testa simetric în Vitest/Jest și vei documenta politicile care fac ca rezultatele să fie reproductibile și explicabile.

Exemplu: pipeline parametric pentru top‑K interese (fragment din `cli-runner.js`):

const interests = pipe(  
 (xs) => xs.map(r => r.interests || []),  
 flatten,  
 compact,  
 (xs) => xs.map(x => x.toLowerCase().trim()).filter(Boolean),  
 (xs) => Object.entries(countBy(xs, x => x)),  
 (entries) => entries.map(([key, count]) => ({ key, count })),  
 (arr) => arr.sort(byCountThenAlpha),  
 (arr) => arr.slice(0, opts.limit ?? 5),  
 tap(() => {})  
)(filtered)

Exemplu: test „smoke” pentru CLI (Vitest):

import { describe, it, expect } from 'vitest'  
import { run } from '../../src/cli-runner.js'  
function memIO (files = {}) {  
 const out = { out:'', err:'' }  
 return { readFile: async (p) => files[p], writeOut: (s) => { out.out += s + '\n' }, writeErr: (s) => { out.err += s + '\n' }, out }  
}  
describe('cli-runner', () => {  
 it('csv -> table report', async () => {  
 const io = memIO({ 'data.csv': 'name,email,faculty,year,interests\nAna,ana@x.org,CSIE,1,AI|Web' })  
 const code = await run(['data.csv','--limit','5','--format','table'], io)  
 expect(code).toBe(0)  
 expect(io.out.out).toContain('Top interests')  
 })  
})