**Seminarul 1 — Partea 2 (Laborator extins)**

Această \*\*Partea 2 — Laborator\*\* transformă conceptele din Partea 1 (sintaxă modernă, funcții, iterație, CLI) într‑un exercițiu cap‑la‑cap pe aceeași temă: generarea rapidă a unui \*\*raport „StudentHub”\*\* din date CSV/JSON. Structura e \*\*progresivă\*\*: începem cu micro‑exerciții pe funcții pure, continuăm cu integrarea într‑un \*\*pipeline\*\* map/filter/reduce, apoi ambalăm logica într‑un \*\*CLI\*\* testabil care produce rezultate în \*\*format tabelar\*\* sau \*\*JSON\*\*. Toate etapele vin cu \*\*worksheet (cerință + checklist)\*\*, \*\*starter code\*\*, și \*\*teste\*\* (Vitest & Jest în oglindă).

────────────────────────────────────────────────────────────────────────

# 0. Imagine de ansamblu  
\*\*Obiectiv:\*\* să exersezi scrierea de funcții pure, compunerea lor în pipelines declarative și separarea dintre \*\*logică\*\* și \*\*I/O\*\*, pentru a obține un \*\*CLI\*\* robust și testabil.

\*\*Livrabile:\*\*   
- proiect Node.js minimal (`s1p2-lab`) cu `src/lib` (funcții pure), `src/cli-runner.js` (orchestrare), `bin/cli.js` (intrare CLI), `data/` (fișiere exemplu), `tests/` (Vitest & Jest), configurări ESLint (flat), Vitest, Jest, Babel‑Jest.  
- rulări demonstrative (tabel/JSON), plus \*edge cases\* (fișier lipsă, format greșit, filtrări).

\*\*Principii:\*\*   
- \*\*Puritate\*\* pentru logică (fără I/O în funcții pure).  
- \*\*Contracte verificabile\*\* prin teste (același set în două ecosisteme).  
- \*\*CLI\*\* ca adaptor la margine (parsing argumente, I/O), testabil prin injecție de I/O.

────────────────────────────────────────────────────────────────────────

# 1. Worksheet general (cerință + checklist)

## Cerință  
Construiește un utilitar CLI numit `studenthub-report` care primește un fișier de intrare (`.csv` sau `.json`) și produce un raport cu:  
- \*\*Total\*\* înregistrări.  
- \*\*Distribuție pe facultăți\*\* (count per `faculty`).  
- \*\*Top K interese\*\* (agregare pe `interests`, K ajustabil cu `--limit`).  
Opțiuni CLI:  
- `--format table|json` (implicit `table`).  
- `--limit N` (implicit `5`).  
- `--faculty NAME` (filtrare pe facultate exactă).  
- `--new-only` + `--ref-date YYYY-MM-DD` (filtrare după vechime).

## Checklist  
- [ ] Toată logica de procesare e în \*\*funcții pure\*\*.  
- [ ] \*\*I/O\*\* (citire fișier, scriere stdout/stderr) e exclusiv în runner/CLI.  
- [ ] Există \*\*teste\*\* care confirmă: parsarea datelor, agregările, `topK`, fluxul CLI „fericit” și cazuri de eroare.  
- [ ] `npm test` rulează \*\*Vitest\*\* și \*\*Jest\*\* (ambele verzi).  
- [ ] `npm run dev` produce un raport tabelar valid pe `data/sample.csv`.  
- [ ] Lintul trece fără erori critice; stilul e consecvent.  
- [ ] README explică clar pașii de rulare.

────────────────────────────────────────────────────────────────────────

# 2. Setup proiect — structură & fișiere

\*\*Arbore recomandat\*\* (exact așa cum îl primești în arhivă):  
- `src/lib/parse.js` — `parseCSV`, `parseJSON` (fără I/O).  
- `src/lib/transform.js` — `groupBy`, `countBy`, `topK`.  
- `src/lib/filters.js` — `filterNewMembers`, `byFaculty`.  
- `src/lib/format.js` — `formatTable`, `toJSON`.  
- `src/cli-runner.js` — \*\*orchestrator\*\* (parsing argumente, invocare funcții pure); acceptă „I/O injectabil” pentru teste.  
- `bin/cli.js` — intrare CLI; cheamă `run(args)`.  
- `data/sample.csv`, `data/sample.json` — fișiere exemplu.  
- `tests/` (Vitest & Jest) — aceleași contracte în oglindă.  
- Config: ESLint (flat), Vitest, Jest, Babel‑Jest, `.gitignore`.  
- `package.json` — `scripts` (dev, test, lint).

\*\*De ce această structură?\*\* Pentru că trebuie să poți testa logică \*\*fără\*\* să pornești procesul Node ca executabil. Prin `run(argv, io)` poți injecta un „I/O de memorie” în teste (capturând `stdout/stderr`).

────────────────────────────────────────────────────────────────────────

# 3. Etapa A — Micro‑exerciții (funcții pure)

\*\*A1 — Parsare\*\* (`parseCSV`, `parseJSON`):  
- transformă textul în array de obiecte.  
- normalizează câmpurile (ex.: `interests` → listă; `is\_new` → boolean).

\*\*A2 — Transformare\*\* (`groupBy`, `countBy`, `topK`):  
- `groupBy` — dicționar `cheie → listă`.  
- `countBy` — dicționar `cheie → număr`.  
- `topK` — cele mai frecvente K elemente (sau cele cu valoarea cea mai mare după `valFn`).

\*\*A3 — Filtrare\*\* (`filterNewMembers`, `byFaculty`):  
- compune predicatele pentru `Array.prototype.filter`.  
- ziua de referință `--ref-date` e injectată pentru testabilitate.

\*\*A4 — Formatare\*\* (`formatTable`, `toJSON`):  
- mută tot ce ține de „cum arată” raportul în funcții dedicate (ușor de testat).

\*\*\*Rezultat A\*\*\*: toate funcțiile pure au teste verzi (Vitest & Jest).

────────────────────────────────────────────────────────────────────────

# 4. Etapa B — Integrare (runner CLI)

\*\*Obiectiv\*\*: să compui funcțiile pure într‑un flux complet, cu parsing de argumente minimal (fără dependențe externe).

\*\*Chei de proiectare\*\*:  
- `run(argv, io)` → returnează \*\*cod de ieșire\*\* `0/1`.  
- În `io` ai: `readFile`, `writeOut`, `writeErr` — injectabile la testare.  
- \*\*Parsare argumente\*\*: prefix `--` cu `--k=v` sau `--k v` (minimalist).

\*\*\*Rezultat B\*\*\*: `npm run dev` produce raport pe `data/sample.csv` în `table` și `json` (dacă modifici `--format`).

────────────────────────────────────────────────────────────────────────

# 5. Etapa C — Calitate: teste duble, lint, stil

- \*\*Teste unitare\*\*: pentru parse/transform/filters/format.  
- \*\*Teste runner\*\*: cel puțin 3 — lipsă argumente, CSV „fericit”, JSON „fericit”.  
- \*\*Lint\*\*: ESLint flat config — reguli minime și ecmaVersion 2022.  
- \*\*Naming\*\*: denumiri expresive; funcții scurte.

────────────────────────────────────────────────────────────────────────

# 6. Explicații pas cu pas (build‑along)

1) \*\*Rulează testele\*\*: vezi ce pică (e firesc la început).   
2) \*\*Completează parse/transform\*\*: urmărește mesajele testelor — ele descriu \*\*contractul\*\*.   
3) \*\*Conectează runner‑ul\*\*: leagă funcțiile pure și formatele de ieșire.   
4) \*\*Rafinează\*\* filtrările/opțiunile.   
5) \*\*Rulează CLI\*\* pe `data/sample.csv` → validează cu ochiul liber.   
6) \*\*Adaugă teste\*\* pentru „margini” (listă goală, facultate lipsă, interes duplicat).

────────────────────────────────────────────────────────────────────────

# 7. AI‑assist — cum îl folosim inteligent

- \*\*Prompt test‑first\*\*: „scrie 5 teste pentru `countBy` (inclusiv `[]`, chei `null/undefined`, stringuri goale)”.   
- \*\*Refactor\*\*: „propune un `parseArgs` mai clar; păstrează contractul existenței”.   
- \*\*Edge‑cases\*\*: „listează 10 cazuri limită pentru `topK` (K=0, K>len, valori egale, comparator custom)”.   
- \*\*Guardrails\*\*: acceptă propunerile AI doar după ce \*\*trec testele\*\*.

────────────────────────────────────────────────────────────────────────

# 8. Extensii (opționale)  
- Suport `--output <file>` (scriere în fișier).  
- Suport \*CSV quoting\* mai avansat (câmpuri cu virgulă; necesită parser robust).  
- Integrare cu `stdin` (citire din pipe).  
- Export Markdown/HTML al raportului.

────────────────────────────────────────────────────────────────────────

# 9. Rubrica de evaluare  
- \*\*Corectitudine\*\* (teste verzi în Vitest & Jest): 40%  
- \*\*Calitate cod\*\* (nume, structură, puritate, separare I/O): 30%  
- \*\*CLI UX\*\* (mesaje, usage, coduri de ieșire): 15%  
- \*\*Testare extinsă\*\* (edge‑cases suplimentare): 10%  
- \*\*Documentare (README)\*\*: 5%

────────────────────────────────────────────────────────────────────────

# 10. Anexă — Comenzi utile

bash  
# Instalare deps  
npm i

# Rulare teste  
npm test  
npm run test:vitest  
npm run test:jest

# Rulare CLI  
npm run dev  
node bin/cli.js data/sample.csv --limit 3 --format json  
node bin/cli.js data/sample.json --faculty CSIE --new-only --ref-date 2025-09-14  
```