**Seminarul 12 — Tabele & Grafice — Partea 2 (Laborator extins)**

\*\*Obiectivul Părții 2\*\* este să construim un \*\*dashboard\*\* didactic „\*\*ClubHub+ Analytics\*\*” folosind \*\*TanStack Table\*\* (tabel interactiv) și \*\*Chart.js\*\* (grafic bar) peste un \*\*pipeline curat de date\*\* (selectors puri), astfel încât tabelele și graficele să fie \*\*sincronizate\*\* printr‑o \*\*sursă unică a adevărului\*\*. Vei implementa controlul căutării (search) și al filtrării pe club, vei exersa \*\*sortarea accesibilă\*\* (`aria-sort`) și vei genera un \*\*config determinist\*\* pentru chart (fără animații) ca să fie \*\*testabil\*\* în \*\*Vitest\*\* și \*\*Jest\*\*. Toate explicațiile sunt pas cu pas; la final ai un \*\*worksheet\*\* cu \*\*checklist\*\* și un \*\*starter\*\* complet cu testele aferente.

---

## 0) Setup (Vite + React + TS, TanStack Table, Chart.js, Vitest & Jest)

\*\*Scop.\*\* Pregătim proiectul pentru lucru rapid și testare duală. `package.json` definește scripturile, `vitest.config.ts` și `jest.config.cjs` setează mediul \*\*JSDOM\*\* și încărcarea `@testing-library/jest-dom`. Structura este minimalistă și lizibilă: `src/` (App, componente, selecții), `tests/` (Vitest/Jest), `public/data` (fixture).

\*\*De ce două framework‑uri de test?\*\* Pentru comparația API‑urilor și pentru robustețea suitei: unele echipe preferă Jest, altele Vitest. Vei observa paralele între \*mocks\* și \*matchers\* și vei înțelege mai bine decuplarea dintre \*\*contract\*\* (ce asertăm) și \*\*implementare\*\*.

\*\*Ce verifici acum:\*\* instalare dependențe, rulare `npm test` (ambele suite), pornire `npm run dev` și prezența headerului „ClubHub+ Analytics”.

---

## 1) Date & Selectori: „sursă unică a adevărului”

\*\*Scop.\*\* Evităm duplicarea și \*coupling\* inutil. Păstrăm un singur \*source\* (`useMembersData` → `rows`), iar derivele (filtrare, agregări pentru grafice) sunt \*\*funcții pure\*\* (selectors) — \*ușor de testat\*, \*predictibile\*, \*reutilizabile\*.

\*\*Pași.\*\*  
1. `src/data/members.json`: fișier fixture cu 15 membri (cluburi Tech/Arts/Sports).   
2. `useMembersData.ts`: hook simplu care expune `rows` (în proiecte reale: RTK Query).   
3. `selectors/analytics.ts`: `filterRows(rows, query, club)` și `deriveChartConfig(rows)` → întorc exact structurile de care au nevoie \*\*tabelul\*\* și \*\*graficul\*\*.

\*\*De ce așa?\*\* Separarea clară dintre datele „raw” și reprezentările pentru UI scade complexitatea accidentală și face testele naturale (verifici \*\*ieșirea\*\* selectorilor, nu UI‑ul în sine).

---

## 2) Tabelul (TanStack Table): sortare accesibilă + status rânduri

\*\*Scop.\*\* Redăm un tabel curat, cu headere sortabile, `aria-sort` corect și un indicator al numărului de rânduri vizibile. Ne bazăm pe TanStack pentru core‑row‑model + sorted‑row‑model; HTML rămâne semantic (`<table>`, `<thead>`, `<tbody>`, `<caption>`).

\*\*Pași.\*\*  
1. Definești coloanele cu `createColumnHelper` (name, club, joinedAt).   
2. Activezi sortarea și conectezi `SortingState` la headere (buton).   
3. Expui `aria-sort` în `th` (none/ascending/descending).   
4. Randare `tbody` cu `flexRender` — separă prezentarea de date.   
5. Afișezi `Rows: N` în footerul secțiunii (ajută și în teste).

\*\*Testare (RTL).\*\* Dăm click pe „sort Name” de 2 ori și asertăm ordinea rândurilor. Acest test acoperă contractul principal fără să se agațe de detalii interne.

---

## 3) Controale (search + club filter) și sincronizare tabel/grafic

\*\*Scop.\*\* Controale „controlled” (React) ce alimentează selectorii. Atunci când schimbi `query` sau `club`, se recalculează `visible` și `chart`. Conectăm totul în `App.tsx` cu `useMemo` pentru a evita calcule inutile.

\*\*Pași.\*\*  
1. `useState` pentru `query` și `club`.   
2. `visible = filterRows(rows, query, club)`; `chart = deriveChartConfig(visible)`.   
3. Tabelul primește `visible`, graficul primește `chart`.   
4. Verifici „Rows: N” și aspectul gros al graficului (smoke).

\*\*Observație.\*\* `deriveChartConfig` primește \*\*doar\*\* rândurile vizibile → graficul reflectă filtrarea curentă, fără altă sursă de adevăr.

---

## 4) Graficul (Chart.js): wrapper determinist, mock‑able

\*\*Scop.\*\* Construiți `ChartPanel` care primește `{ labels, datasets }` și instanțiază \*\*Chart.js\*\* pe un `<canvas>`; \*\*animațiile dezactivate\*\* pentru determinism în teste. `useEffect` distruge instanța la unmount sau la schimbarea config‑ului (evităm scurgeri de resurse).

\*\*Testare.\*\* În teste, \*\*mock‑uiți\*\* constructorul din `chart.js/auto` și asertați existența `<canvas>`; pentru contract mai strict, puteți intercepta \*\*parametrii\*\* cu care e chemat constructorul (exercițiu opțional).

---

## 5) Extensii opționale (după laboratorul de bază)

- \*\*Pagination\*\* (client‑side): mențineți `pageIndex`/`pageSize` în `App` și adăugați controale simple.   
- \*\*CSV export\*\*: exportați fie „raw”, fie „vizibil”; documentați în UI ce variantă ați ales.   
- \*\*Theming\*\* (CSS variables): buton dark/light.   
- \*\*i18n\*\*: formatări RO/EN cu `Intl.\*`.   
- \*\*RTK Query\*\*: înlocuiți `useMembersData` cu un `createApi` minimal, pentru dataset mai mare.

---

## 6) Teste: unit (selectors), integrare (table), contract (chart)

\*\*Scop.\*\* Să avem încredere în modificări.   
- \*\*Unit\*\*: `filterRows` (predicate text/club) + `deriveChartConfig` (agregare).   
- \*\*Integrare DOM\*\*: sortarea în tabel (RTL).   
- \*\*Contract\*\*: graficul instanțiat corect (constructor mock).   
Toate testele rulează \*\*identic\*\* în \*\*Vitest\*\* și \*\*Jest\*\*.

---

## Worksheet (Cerință + Checklist)

\*\*Cerință.\*\* Implementați cap‑coadă dashboard‑ul: controlați căutarea/filtrarea, redați tabelul cu sortare accesibilă, sincronizați graficul cu rândurile vizibile, scrieți testele duale până la \*\*verde\*\*. Extindeți cu cel puțin o funcționalitate opțională (pagination/CSV/theming).

\*\*Checklist.\*\*  
- [ ] `npm test` rulează Vitest & Jest (toate verzi).   
- [ ] Sortarea „Name” (asc/desc) funcționează (`aria-sort` corect).   
- [ ] `filterRows` reduce rândurile la cele potrivite query/club.   
- [ ] `deriveChartConfig` produce `labels`/`datasets` corecte.   
- [ ] `ChartPanel` instanțiază un `<canvas>` și distruge instanța la unmount.   
- [ ] (Opțional) Pagination/CSV/Theming/i18n implementate.

---

## AI‑assist (VSL) — prompts recomandate

- „Create TanStack Table columns for {name, club, joinedAt} with sortable headers and aria-sort.”   
- „Write pure selectors `filterRows(rows, query, club)` and `deriveChartConfig(rows)` returning `{labels,datasets}` for Chart.js.”   
- „Mock `chart.js/auto` constructor in Vitest & Jest and assert the canvas exists & is created with given labels/datasets.”

---

## Troubleshooting (tipic)

- \*\*Jest TSX\*\*: necesită `babel-jest` + preseturi (incluse).   
- \*\*Vitest JSDOM\*\*: `environment:'jsdom'` + `setupFiles` (incluse).   
- \*\*Eroare la import Chart.js\*\*: folosiți `chart.js/auto` sau wrapperul nostru; în teste, mock.   
- \*\*Performanță\*\*: data‑sets mari → luați în calcul virtualizarea / pagination la sursă.

\*\*Fragment:\*\* `package.json`

{  
 "name": "s12p2-tables-charts-lab",  
 "version": "1.0.0",  
 "private": true,  
 "type": "module",  
 "scripts": {  
 "dev": "vite",  
 "build": "vite build",  
 "preview": "vite preview",  
 "test:vitest": "vitest run --reporter verbose",  
 "test:jest": "jest --runInBand",  
 "test": "npm run test:vitest && npm run test:jest"  
 },  
 "dependencies": {  
 "react": "^19.0.0",  
 "react-dom": "^19.0.0",  
 "@tanstack/react-table": "^8.20.0",  
 "chart.js": "^4.4.3"  
 },  
 "devDependencies": {  
 "@babel/preset-env": "^7.25.0",  
 "@babel/preset-react": "^7.24.0",  
 "@babel/preset-typescript": "^7.24.0",  
 "@testing-library/jest-dom": "^6.4.2",  
 "@testing-library/react": "^14.2.1",  
 "@testing-library/user-event": "^14.5.2",  
 "@types/jest": "^29.5.12",  
 "@types/node": "^20.11.17",  
 "@types/react": "^18.3.3",  
 "@types/react-dom": "^18.3.0",  
 "@vitejs/plugin-react": "^4.3.0",  
 "babel-jest": "^29.7.0",  
 "jest": "^29.7.0",  
 "jsdom": "^24.1.0",  
 "typescript": "^5.4.0",  
 "vite": "^5.4.0",  
 "vitest": "^1.6.0"  
 }  
}

\*\*Fragment:\*\* `src/selectors/analytics.ts`

import type { Member } from '../data/useMembersData'  
  
export function filterRows(rows: Member[], query: string, club: 'all'|'Tech'|'Arts'|'Sports'){  
 const q = query.trim().toLowerCase()  
 return rows.filter(r => {  
 const okQ = !q || r.name.toLowerCase().includes(q) || r.club.toLowerCase().includes(q)  
 const okC = club === 'all' || r.club === club  
 return okQ && okC  
 })  
}  
  
export function deriveChartConfig(rows: Member[]){  
 const byClub = new Map<string, number>()  
 for(const r of rows) byClub.set(r.club, (byClub.get(r.club)||0)+1)  
 const labels = Array.from(byClub.keys())  
 const values = Array.from(byClub.values())  
 return {  
 labels,  
 datasets: [{ label: 'Members', data: values }]  
 }  
}

\*\*Fragment:\*\* `src/components/DataTable.tsx`

import React from 'react'  
import { useReactTable, getCoreRowModel, getSortedRowModel, getFilteredRowModel, flexRender, createColumnHelper, SortingState } from '@tanstack/react-table'  
import type { Member } from '../data/useMembersData'  
  
const h = createColumnHelper<Member>()  
  
const columns = [  
 h.accessor('name', { header: 'Name' }),  
 h.accessor('club', { header: 'Club' }),  
 h.accessor('joinedAt', { header: 'Joined', cell: (info)=> new Date(info.getValue()).toLocaleDateString() })  
]  
  
export function DataTable({ rows }: { rows: Member[] }){  
 const [sorting, setSorting] = React.useState<SortingState>([])  
 const table = useReactTable({  
 data: rows,  
 columns,  
 state: { sorting },  
 onSortingChange: setSorting,  
 getCoreRowModel: getCoreRowModel(),  
 getSortedRowModel: getSortedRowModel(),  
 getFilteredRowModel: getFilteredRowModel()  
 })  
  
 return (  
 <div role="region" aria-label="members-table">  
 <table className="table">  
 <caption>Members — sortable columns, accessible headers.</caption>  
 <thead>  
 {table.getHeaderGroups().map(hg => (  
 <tr key={hg.id}>  
 {hg.headers.map(header => {  
 const canSort = header.column.getCanSort()  
 const sortDir = header.column.getIsSorted()  
 const ariaSort = sortDir === 'asc' ? 'ascending' : sortDir === 'desc' ? 'descending' : 'none'  
 return (  
 <th key={header.id} scope="col" aria-sort={ariaSort as any}>  
 {canSort ? (  
 <button onClick={header.column.getToggleSortingHandler()} aria-label={`sort ${String(header.column.columnDef.header)}`}>  
 {flexRender(header.column.columnDef.header, header.getContext())}  
 </button>  
 ) : flexRender(header.column.columnDef.header, header.getContext())}  
 </th>  
 )  
 })}  
 </tr>  
 ))}  
 </thead>  
 <tbody>  
 {table.getRowModel().rows.map(row => (  
 <tr key={row.id}>  
 {row.getVisibleCells().map(cell => (  
 <td key={cell.id}>{flexRender(cell.column.columnDef.cell, cell.getContext())}</td>  
 ))}  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 <p role="status">Rows: {table.getRowModel().rows.length}</p>  
 </div>  
 )  
}

\*\*Fragment:\*\* `src/components/ChartPanel.tsx`

import React, { useEffect, useRef } from 'react'  
import { Chart } from 'chart.js/auto'  
  
type Dataset = { label: string; data: number[] }  
type Config = { labels: string[]; datasets: Dataset[] }  
  
export function ChartPanel({ config }: { config: Config }){  
 const ref = useRef<HTMLCanvasElement>(null)  
 useEffect(()=>{  
 if(!ref.current) return  
 const chart = new Chart(ref.current, {  
 type: 'bar',  
 data: { labels: config.labels, datasets: config.datasets },  
 options: { animation: false, responsive: true, plugins: { legend: { display: true } } }  
 })  
 return () => chart.destroy()  
 }, [config.labels.join(','), JSON.stringify(config.datasets)])  
 return <canvas aria-label="chart" ref={ref} />  
}

\*\*Fragment:\*\* `tests/vitest/table.test.tsx`

import { render, screen } from '@testing-library/react'  
import userEvent from '@testing-library/user-event'  
import { DataTable } from '../../src/components/DataTable'  
import type { Member } from '../../src/data/useMembersData'  
  
const rows: Member[] = [  
 { id:'1', name:'B', club:'Tech', joinedAt:'2025-01-10' },  
 { id:'2', name:'A', club:'Arts', joinedAt:'2025-01-12' }  
]  
  
test('table sorts by Name', async () => {  
 render(<DataTable rows={rows}/>)  
 const user = userEvent.setup()  
 const btn = screen.getByRole('button', { name: /sort Name/i })  
 await user.click(btn) // asc  
 const all = screen.getAllByRole('row')  
 // row[1] first data row  
 expect(all[1]).toHaveTextContent('A')  
 await user.click(btn) // desc  
 const all2 = screen.getAllByRole('row')  
 expect(all2[1]).toHaveTextContent('B')  
})

\*\*Fragment:\*\* `tests/vitest/chart.test.tsx`

import { render } from '@testing-library/react'  
import { ChartPanel } from '../../src/components/ChartPanel'  
  
vi.mock('chart.js/auto', () => {  
 return {  
 Chart: vi.fn().mockImplementation(() => ({ destroy: vi.fn() }))  
 }  
})  
  
test('ChartPanel constructs chart with labels/datasets', () => {  
 const { container } = render(<ChartPanel config={{ labels:['Tech','Arts'], datasets:[{ label:'Members', data:[2,1] }] }} />)  
 const canvas = container.querySelector('canvas')  
 expect(canvas).toBeInTheDocument()  
})

\*\*Fragment:\*\* `tests/vitest/selectors.test.ts`

import { filterRows, deriveChartConfig } from '../../src/selectors/analytics'  
import type { Member } from '../../src/data/useMembersData'  
  
const rows: Member[] = [  
 { id:'1', name:'Ana Pop', club:'Tech', joinedAt:'2025-01-10' },  
 { id:'2', name:'Mihai Ionescu', club:'Arts', joinedAt:'2025-01-12' },  
 { id:'3', name:'Elena Radu', club:'Tech', joinedAt:'2025-02-05' },  
]  
  
test('filterRows by query and club', () => {  
 const v1 = filterRows(rows, 'ana', 'all')  
 expect(v1.length).toBe(1)  
 const v2 = filterRows(rows, '', 'Tech')  
 expect(v2.length).toBe(2)  
})  
  
test('deriveChartConfig aggregates by club', () => {  
 const cfg = deriveChartConfig(rows)  
 expect(cfg.labels).toEqual(expect.arrayContaining(['Tech','Arts']))  
 expect(cfg.datasets[0].data.reduce((a,b)=>a+b,0)).toBe(3)  
})