**Seminarul 8 — Partea 2: Laborator (extins)**

**Obiectivul laboratorului**

Construim incremental un \*\*REST API\*\* în \*\*Node.js + Express\*\* pentru domeniul „Clubs & Associations Hub”, urmând principii stricte de proiectare: rute clare, validări robuste (input și, selectiv, output), erori standardizate \*\*RFC 7807\*\* (`application/problem+json`), CORS configurat prudent, \*\*ETag\*\* și condițional (demo), \*\*Idempotency-Key\*\* (demo) și o \*\*observabilitate\*\* minimă (correlation‑id + loguri structurate). Vom susține toate acestea cu \*\*teste\*\* rulate în paralel, în două cadre: \*\*Vitest\*\* și \*\*Jest\*\* (side‑by‑side).

**Context & temă transversală**

Tema aplicată este consistentă cu Partea 1 și cu Seminarul 7 (clientul Fetch): \*\*UI‑ul\*\* consumă `GET /api/clubs` (cu paginare/filtrare/sortare), `GET /api/clubs/:id` (ETag), `POST /api/clubs`, `PATCH/DELETE /api/clubs/:id`, respectiv `GET /api/registrations?club=…` și `POST /api/registrations` (cu `Idempotency-Key`). În laborator, persistăm \*\*in‑memory\*\* (pentru claritate) — accentul cade pe contractul HTTP și pe disciplinele de validare și erori.

**Worksheet — enunț (cerință)**

1) \*\*Bootstrap proiect\*\* (Node 20+, Express 4.x): middlewares de bază — `correlationId`, `express.json()`, `cors`, `requestLogger`, `notFound`, `errorHandler`.   
2) \*\*Resursa `clubs`\*\*:   
 • `GET /api/clubs` (paginare & sortare),   
 • `GET /api/clubs/:id` cu \*\*ETag\*\* și suport `If-None-Match → 304`,   
 • `POST /api/clubs` (201 + `Location`),   
 • `PATCH /api/clubs/:id` (JSON Merge Patch minim),   
 • `DELETE /api/clubs/:id` (204).   
3) \*\*Resursa `registrations`\*\*:   
 • `GET /api/registrations?club=…` (filtrare),   
 • `POST /api/registrations` (validare + \*\*Idempotency‑Key\*\* pentru evitarea duplicatelor).   
4) \*\*Validări\*\*: predicate pure în `schemas/\*`, middleware `validateBody`, mapare `422` (RFC 7807) cu câmp `errors`.   
5) \*\*Erori\*\*: `problem()` + `toProblem()`; codurile corecte 404/409/422/5xx; `application/problem+json`.   
6) \*\*Testare\*\*: suite \*\*Vitest\*\* și \*\*Jest\*\* în paralel: teste unitare (validatori, serializer problem) și teste HTTP (Supertest) pentru rute cheie.   
7) \*\*AI‑assist (VSL)\*\*: scrie prompturi scurte pentru generarea scheletelor, apoi rafinează manual: `validate`, `problem`, `routes`, `tests`.

**Checklist — criterii de acceptare**

• `GET /api/clubs` returnează `{ items, page, limit, total }` și respectă sortarea `?sort=name,-createdAt`.   
• `GET /api/clubs/:id` expune `ETag` și respectă `If-None-Match → 304`.   
• `POST /api/clubs` mapează corect `201 + Location`; input invalid → `422` cu `errors`.   
• `PATCH /api/clubs/:id` acceptă modificări parțiale (minim `name`/`category`).   
• `DELETE /api/clubs/:id` → `204`; resursa inexistentă → `404`.   
• `POST /api/registrations` validează și suportă `Idempotency-Key` (aceeași cheie → același rezultat).   
• Toate erorile ies ca `application/problem+json` (RFC 7807).   
• Testele Vitest & Jest trec pe aceleași scenarii (HTTP + unit).

**Structură recomandată de directoare**

s8p2-standalone/  
 src/  
 app.js # compunerea Express  
 server.js # pornire  
 routes/{clubs.routes.js, registrations.routes.js, health.routes.js}  
 controllers/{clubs.controller.js, registrations.controller.js}  
 services/{clubs.service.js, registrations.service.js}  
 schemas/{club.schema.js, registration.schema.js}  
 middlewares/{correlationId.js, logger.js, notFound.js, errors.js, validate.js}  
 errors/problem.js  
 repo/inMemoryStore.js  
 utils/{etag.js, pagination.js}  
 tests/  
 vitest/{http.test.ts, unit.test.ts}  
 jest/{http.test.cjs, unit.test.cjs}  
 package.json  
 vitest.config.ts  
 jest.config.cjs

**Pașii laboratorului — explicați pas cu pas**

\*\*Pasul A — Bootstrap & middlewares\*\*. În `src/app.js` configurăm pipeline‑ul Express: `correlationId` adaugă/propagă un `X‑Correlation‑Id`, `express.json()` parsează JSON cu limită rezonabilă, `cors()` admite origini locale pentru dezvoltare, `requestLogger` emite un JSON per request (ts, method, url, status, durMs, cid). La final, `notFound` și `errorHandler` unifică erorile sub formă RFC 7807.

\*\*Pasul B — Repo & seed in‑memory.\*\* În `repo/inMemoryStore.js` punem 3 cluburi inițiale (tech, debate, arts) și structurile `registrations` și `idempotency`. Aceasta ne permite să testăm fără DB.

\*\*Pasul C — `clubs` services & controllers.\*\* `listClubs` aplică filtrări, sortări și paginare; `getClub` returnă entitatea; `createClub` verifică duplicarea `id`‑ului; `patchClub` actualizează parțial; `deleteClub` elimină resursa (sau ridică `404`). Controller‑ele sunt subțiri: preiau și normalizează parametrii, apelează servicii, setează antete (`ETag`, `Cache-Control`) și status‑uri corecte.

\*\*Pasul D — `registrations` services & idempotency.\*\* `createRegistration` verifică existenta clubului și respectă \*\*Idempotency-Key\*\*: dacă aceeași cheie a mai fost procesată, returnăm exact același obiect (și același `Location`).

\*\*Pasul E — Validări & erori.\*\* `validateBody` face apel la validatori pure din `schemas/\*` și mapează erorile la `422` (`Unprocessable Content`) cu câmp `errors` granular. `problem.js` expune fabrici scurte pentru `badRequest`, `notFound`, `conflict`, `unprocessable` etc.; `errorHandler` serializează uniform.

\*\*Pasul F — ETag & condițional.\*\* `GET /api/clubs/:id` calculează un ETag „weak” sha1‑base64; dacă clientul trimite `If‑None‑Match` egal, răspundem `304`. Beneficiul: reducere trafic & timp.

\*\*Pasul G — Testare dublă (Vitest & Jest).\*\* Scriem aceleași scenarii o dată cu `vitest`, o dată cu `jest`. Pentru HTTP folosim `supertest` (fără a porni serverul la port). Pentru validatori & serializer problem testăm izolat.

**Cod cheie — app.js (pipeline-ul Express)**

const express = require('express');  
const cors = require('cors');  
  
// middlewares  
const { correlationId } = require('./middlewares/correlationId');  
const { requestLogger } = require('./middlewares/logger');  
const { errorHandler } = require('./middlewares/errors');  
const { notFound } = require('./middlewares/notFound');  
  
// routes  
const clubsRoutes = require('./routes/clubs.routes');  
const regsRoutes = require('./routes/registrations.routes');  
const healthRoutes = require('./routes/health.routes');  
  
const app = express();  
  
// base middlewares  
app.use(correlationId);  
app.use(express.json({ limit: '256kb' }));  
app.use(cors({ origin: [/^http:\/\/localhost:\d+$/], credentials: false }));  
app.use(requestLogger);  
  
// mount routes  
app.use('/ping', healthRoutes);  
app.use('/api/clubs', clubsRoutes);  
app.use('/api/registrations', regsRoutes);  
  
// 404 & error  
app.use(notFound);  
app.use(errorHandler);  
  
module.exports = app;

Observați ordinea: \*\*correlationId\*\* → \*\*json\*\* → \*\*cors\*\* → \*\*requestLogger\*\* → \*\*routes\*\* → \*\*notFound\*\* → \*\*errorHandler\*\*. Orice excepție ajunge în `errorHandler`, care va serializa `problem+json`.

**Cod cheie — problem.js (RFC 7807)**

// RFC 7807 helpers  
function problem({ type, title, status, detail, instance, errors, extras }){  
 const obj = { type, title, status, detail };  
 if(instance) obj.instance = instance;  
 if(errors) obj.errors = errors;  
 if(extras && typeof extras === 'object') Object.assign(obj, extras);  
 return obj;  
}  
  
function toProblem(err, { instance } = {}){  
 // Known http-like  
 if(err && typeof err.status === 'number'){  
 const p = problem({  
 type: err.type || 'about:blank',  
 title: err.title || err.message || 'HTTP error',  
 status: err.status,  
 detail: err.detail,  
 instance,  
 errors: err.errors  
 });  
 return p;  
 }  
 // Validation shape  
 if(err && err.name === 'ValidationError'){  
 return problem({  
 type: 'https://example.edu/problems/validation',  
 title: 'Validation failed',  
 status: 422,  
 detail: 'One or more fields are invalid.',  
 instance,  
 errors: err.errors || {}  
 });  
 }  
 // Generic  
 return problem({  
 type: 'about:blank',  
 title: 'Internal Server Error',  
 status: 500,  
 detail: process.env.NODE\_ENV === 'production' ? undefined : (err && err.stack) || String(err),  
 instance  
 });  
}  
  
// Shortcuts  
function badRequest(detail, extras){ const e = new Error('Bad Request'); e.status = 400; e.detail = detail; if(extras) e.errors = extras; return e; }  
function notFound(detail){ const e = new Error('Not Found'); e.status = 404; e.detail = detail; return e; }  
function conflict(detail){ const e = new Error('Conflict'); e.status = 409; e.detail = detail; return e; }  
function unprocessable(errors){ const e = new Error('Unprocessable Content'); e.status = 422; e.name='ValidationError'; e.errors = errors || {}; return e; }  
function tooMany(detail){ const e = new Error('Too Many Requests'); e.status = 429; e.detail = detail; return e; }  
  
module.exports = { problem, toProblem, badRequest, notFound, conflict, unprocessable, tooMany };

`toProblem` convertește erorile interne într‑un obiect \*\*problem+json\*\*: astfel, aplicația răspunde \*\*uniform\*\*, iar clienții pot interpreta consistent erorile (ex. hărți de câmpuri pentru `422`).

**Cod cheie — ETag (utils/etag.js)**

const crypto = require('node:crypto');  
function calcEtag(obj){  
 const json = JSON.stringify(obj);  
 const hash = crypto.createHash('sha1').update(json).digest('base64');  
 return `W/"${hash}"`;  
}  
module.exports = { calcEtag };

ETag‑ul este generat ca un hash „weak” pe serializarea JSON; în scop didactic e suficient. În producție, îl corelați cu un câmp `version` sau un hash pe reprezentarea stocată.

**Cod cheie — Idempotency-Key (registrations.service.js)**

const { db, genId } = require('../repo/inMemoryStore');  
const { notFound } = require('../errors/problem');  
  
function listRegistrations({ club }){  
 let items = db.registrations.slice();  
 if(club){ items = items.filter(r => r.club === club); }  
 return items;  
}  
  
function createRegistration(data, idemKey){  
 // optional idempotency  
 if(idemKey && db.idempotency.has(idemKey)){  
 return db.idempotency.get(idemKey);  
 }  
 // business rule: club must exist  
 const c = db.clubs.find(x => x.id === data.club);  
 if(!c) throw notFound('Club does not exist');  
  
 const now = new Date().toISOString();  
 const r = { id: genId(), ...data, createdAt: now };  
 db.registrations.push(r);  
 if(idemKey){ db.idempotency.set(idemKey, r); }  
 return r;  
}  
  
module.exports = { listRegistrations, createRegistration };

Mapa `idempotency` păstrează rezultatul pentru o cheie; dacă aceeași cheie revine, întoarcem aceeași înscriere. Într‑un sistem real, acest cache trebuie să fie \*\*persistent\*\* (ex. Redis) și să aibă o \*\*fereastră TTL\*\* limitată.

**Teste — HTTP (Vitest)**

import { describe, it, expect } from 'vitest';  
import request from 'supertest';  
import { createRequire } from 'module';  
const require = createRequire(import.meta.url);  
const app = require('../../src/app.js');  
  
describe('HTTP — clubs', () => {  
 it('GET /api/clubs returns paginated list', async () => {  
 const res = await request(app).get('/api/clubs').expect(200);  
 expect(res.body).toHaveProperty('items');  
 });  
 it('GET /api/clubs/:id returns 200 with ETag', async () => {  
 const res = await request(app).get('/api/clubs/tech').expect(200);  
 expect(res.headers).toHaveProperty('etag');  
 });  
 it('GET /api/clubs/:id with If-None-Match returns 304', async () => {  
 const first = await request(app).get('/api/clubs/tech').expect(200);  
 const etag = first.headers.etag;  
 await request(app).get('/api/clubs/tech').set('If-None-Match', etag).expect(304);  
 });  
 it('POST /api/clubs creates (201)', async () => {  
 const res = await request(app).post('/api/clubs').set('Content-Type','application/json')  
 .send({ name:'Media Lab', category:'arts', id:'media' }).expect(201);  
 expect(res.headers).toHaveProperty('location');  
 });  
 it('POST /api/clubs invalid → 422', async () => {  
 await request(app).post('/api/clubs').set('Content-Type','application/json')  
 .send({ name:'a', category:'x' }).expect(422);  
 });  
});  
  
describe('HTTP — registrations', () => {  
 it('POST /api/registrations creates (201) and is idempotent with key', async () => {  
 const key = 'abc-123';  
 const body = { fullName:'Ana Pop', email:'ana@uni.ro', club:'tech', consent:true };  
 const r1 = await request(app).post('/api/registrations').set('Content-Type','application/json').set('Idempotency-Key', key).send(body).expect(201);  
 const r2 = await request(app).post('/api/registrations').set('Content-Type','application/json').set('Idempotency-Key', key).send(body).expect(201);  
 expect(r1.body.id).toEqual(r2.body.id);  
 });  
 it('GET /api/registrations?club=tech filters', async () => {  
 const res = await request(app).get('/api/registrations?club=tech').expect(200);  
 expect(Array.isArray(res.body.items)).toBe(true);  
 });  
});

**Teste — HTTP (Jest)**

const request = require('supertest');  
const app = require('../../src/app.js');  
  
test('GET /api/clubs returns list', async () => {  
 const res = await request(app).get('/api/clubs').expect(200);  
 expect(res.body).toHaveProperty('items');  
});  
  
test('ETag flow', async () => {  
 const first = await request(app).get('/api/clubs/tech').expect(200);  
 expect(first.headers).toHaveProperty('etag');  
 const etag = first.headers.etag;  
 await request(app).get('/api/clubs/tech').set('If-None-Match', etag).expect(304);  
});  
  
test('POST /api/clubs invalid → 422', async () => {  
 await request(app).post('/api/clubs').set('Content-Type','application/json')  
 .send({ name:'a', category:'x' }).expect(422);  
});

**Teste — Unit (validatori & problem)**

import { describe, it, expect } from 'vitest';  
import { validateCreateClub } from '../../src/schemas/club.schema';  
import { validateRegistration } from '../../src/schemas/registration.schema';  
import { toProblem, unprocessable } from '../../src/errors/problem';  
  
describe('validators', () => {  
 it('validateCreateClub ok', () => {  
 const r = validateCreateClub({ name:'Robotics', category:'technology' });  
 expect(r.ok).toBe(true);  
 });  
 it('validateRegistration invalid email', () => {  
 const r = validateRegistration({ fullName:'Ana', email:'x', club:'tech', consent:true });  
 expect(r.ok).toBe(false);  
 expect(r.errors).toHaveProperty('email');  
 });  
});  
  
describe('problem serializer', () => {  
 it('422 serializes to problem+json format', () => {  
 const err = unprocessable({ email:'invalid' });  
 const p = toProblem(err, { instance:'/api/registrations' });  
 expect(p.status).toBe(422);  
 expect(p.errors).toHaveProperty('email');  
 });  
});

**Rulare & verificare**

\*\*Standalone\*\*:   
1) `cd s8p2-standalone`   
2) `npm i`   
3) `npm run dev` (API pe `http://localhost:5380`)   
4) `npm test` (Vitest și Jest).

\*\*Monorepo PNPM\*\*:   
1) `cd s8p2-monorepo`   
2) `pnpm i -w`   
3) `pnpm --filter ./packages/api run dev`   
4) `pnpm -w run test`.

Odată rulat serverul, puteți verifica rapid: `curl -i http://localhost:5380/ping` sau `curl -i http://localhost:5380/api/clubs/tech` (observați `ETag`).

**Rubrică (evaluare)**

• \*\*Funcționalitate (40%)\*\*: rutele cerute funcționează; status‑uri corecte; ETag/If‑None‑Match; Idempotency‑Key.   
• \*\*Calitatea codului (25%)\*\*: separare pe layere; nume coerente; comentarii operaționale; lipsa duplicărilor evidente.   
• \*\*Erori & validări (20%)\*\*: `problem+json` standardizat; `422` granular; `404/409/500` mapate corect.   
• \*\*Testare (15%)\*\*: suite Vitest & Jest trec pe aceleași scenarii; acoperire minimă pe validatori & rute cheie.

**Troubleshooting**

• \*\*`422` vs `400`\*\*: JSON sintactic invalid → `400`; JSON valid dar date incorecte → `422`.   
• \*\*ETag nu se schimbă\*\*: uitați să actualizați `updatedAt` la patch; ETag e hash pe serializarea întregului obiect.   
• \*\*Idempotency neefectiv\*\*: cheile nu se păstrează (verificați headerul `Idempotency-Key`); în realitate, folosiți un store persistent.   
• \*\*CORS\*\*: originea UI nu e în lista permisă; ajustați regexul/array‑ul de origini.   
• \*\*Teste HTTP se blochează\*\*: ați pornit serverul pe port în test; cu `supertest` testați `app` direct, fără `listen()`.

**AI‑assist (VSL) — prompturi utile**

• „Generează `clubs.routes.js` cu rutele standard și importurile potrivite; exportă un `Router`.”   
• „Scrie `problem()` compatibil RFC 7807; adaugă `toProblem(err, { instance })`.”   
• „Compune `validateBody(validator)` care returnează `422` cu harta `errors`.”   
• „Propune 5 teste Supertest pentru `GET/POST/PATCH/DELETE` pe `/api/clubs`, inclusiv un 422 și un 404.”   
• „Dă un `Idempotency-Key` demo în service‑ul de înscrieri și două teste care demonstrează că aceeași cheie -> același `id`.”

**Referințe (APA 7, cu DOI)**

Fielding, R., Nottingham, M., & Reschke, J. (2022). HTTP Semantics (RFC 9110). RFC Editor. https://doi.org/10.17487/RFC9110  
Fielding, R., Nottingham, M., & Reschke, J. (2022). HTTP Caching (RFC 9111). RFC Editor. https://doi.org/10.17487/RFC9111  
Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259). RFC Editor. https://doi.org/10.17487/RFC8259  
Nottingham, M., & Wilde, E. (2016). Problem Details for HTTP APIs (RFC 7807). RFC Editor. https://doi.org/10.17487/RFC7807  
Barth, A. (2011). The Web Origin Concept (RFC 6454). RFC Editor. https://doi.org/10.17487/RFC6454  
Klyne, G., & Newman, C. (2002). Date and Time on the Internet: Timestamps (RFC 3339). RFC Editor. https://doi.org/10.17487/RFC3339  
Hunt, P., Bryan, P., & Nottingham, M. (2015). JSON Merge Patch (RFC 7386). RFC Editor. https://doi.org/10.17487/RFC7386