**Seminarul 0 — Partea 2 (Laborator extins)**

**Scopul laboratorului și logica progresivă**

Acest laborator transformă conceptele din Partea 1 într‑un proiect funcțional, testabil și reproductibil.  
Veți porni de la un repository gol, veți inițializa un proiect Node (ESM), veți ridica un server Express minimal care servește  
fișiere statice și expune rute simple (`/ping`, `/api/time`), apoi veți scrie \*\*teste unitare\*\* și \*\*de contract\*\* în \*\*Vitest\*\* și \*\*Jest\*\*,  
în oglindă. Lucrarea este însoțită de un \*\*worksheet\*\* (cerință + checklist) și de un \*\*starter\*\* gata de rulat (în arhivele ZIP).

**Worksheet — cerință + checklist**

Cerință generală: inițializează proiectul „StudentHub Minisite” (Express, ESM), servește `public/index.html`, expune `GET /ping` și `GET /api/time` (ISO‑8601), adaugă testele Vitest/Jest și rulează‑le până sunt verzi.

Checklist minim (toate bifate pentru acceptare):

1) Repo Git inițializat; prim commit.

2) `npm init -y`; `type: "module"` în `package.json`.

3) Dependențe instalate: `express`, `dotenv`; dev: `nodemon`, `vitest`, `jest`, `babel-jest`, `@babel/preset-env`, `supertest`.

4) Structură: `public/index.html`, `src/app.js` (cu `createApp()`), `src/index.js` (pornire server).

5) Rute OK: `GET /ping -> pong`; `GET /api/time -> { now: ISO }`.

6) Servire statică OK: `GET /` întoarce `index.html` (conține `<h1>StudentHub Minisite</h1>`).

7) Scripturi: `dev`, `start`, `test:vitest`, `test:jest`, `test`.

8) Teste trec în ambele suite (Vitest & Jest).

9) `.env.example` + `.gitignore` la zi; port configurabil (`PORT`).

**E1. Inițializare repo & proiect**

Creează un director nou (nume sugestiv) și inițializează Git. Primul commit e o ancoră: îți oferă diferențe clare  
când vei adăuga fișiere ulterior. Apoi rulează `npm init -y` ca să creezi `package.json`. Decide devreme pentru ESM:  
adaugă `"type": "module"`. În această etapă, nu adăuga încă dependențe; prioritară este claritatea scopului.

mkdir tw-seminar-0 && cd tw-seminar-0

git init

echo "# TW Seminar 0" > README.md

git add -A && git commit -m "chore: initial commit (scaffold repo)"

npm init -y

Deschide `package.json` și setează:

{

"name": "tw-seminar-0-lab",

"version": "1.0.0",

"type": "module",

"scripts": {}

}

**E2. Instalează dependențele și definește scripturile**

Instalează \*\*Express\*\* (server HTTP minimalist) și \*\*dotenv\*\* (pentru variabile de mediu).  
În \*dev\*, adaugă \*\*nodemon\*\* (watch), \*\*vitest\*\* și \*\*jest\*\* (test runner‑e), \*\*babel-jest\*\* și `@babel/preset-env` (compat ESM în Jest),  
\*\*supertest\*\* (client de test pentru rute).

npm i express dotenv

npm i -D nodemon vitest jest babel-jest @babel/preset-env supertest

Completează scripturile în `package.json`:

{

"scripts": {

"dev": "nodemon src/index.js",

"start": "node src/index.js",

"test:vitest": "vitest run --reporter verbose",

"test:jest": "jest --runInBand",

"test": "npm run test:vitest && npm run test:jest"

}

}

De ce două suite? Pentru a te obișnui cu ambele ecosisteme și a înțelege că testarea validează \*contracte\*,  
nu un test runner anume. Pe termen lung, vei alege unul în proiectele proprii; aici le punem „umăr la umăr”.

**E3. Structură de directoare & fișiere inițiale**

Separă clar \*\*sursele\*\* de \*\*activele statice\*\*. În `public/` vei pune `index.html` (client),  
iar în `src/` vei avea logica server (Express). `createApp()` e o funcție fabrică; `src/index.js` doar pornește serverul.

tw-seminar-0/

public/

index.html

src/

app.js

index.js

package.json

README.md

**E4. `public/index.html`: un „fum” vizual**

Creează o pagină simplă; acest fișier e dovada că servirea statică e corectă:

<!doctype html>

<html lang="ro">

<head><meta charset="utf-8"><meta name="viewport" content="width=device-width, initial-scale=1.0"><title>StudentHub Minisite</title></head>

<body><h1>StudentHub Minisite</h1><p>Acest conținut este servit de Express.</p></body>

</html>

**E5. `src/app.js`: funcția fabrică a aplicației (Express)**

Folosim ESM și construim aplicația ca funcție. Avantaj: putem importa aplicația în test fără a porni un port,  
economisind timp și evitând coliziuni de porturi.

import express from 'express'

import path from 'path'

import { fileURLToPath } from 'url'

const \_\_filename = fileURLToPath(import.meta.url)

const \_\_dirname = path.dirname(\_\_filename)

export function createApp() {

const app = express()

app.use(express.json())

app.use(express.static(path.join(\_\_dirname, '../public')))

app.get('/ping', (req, res) => res.send('pong'))

app.get('/api/time', (req, res) => res.json({ now: new Date().toISOString() }))

app.use((req, res) => res.status(404).json({ error: 'Not Found' }))

app.use((err, req, res, next) => { console.error(err); res.status(500).json({ error: 'Internal Error' }) })

return app

}

**E6. `src/index.js`: intrarea care pornește serverul**

Separăm clar pornirea serverului (ascultarea pe port) de definiția aplicației. Adăugăm `dotenv` ca să poți suprascrie `PORT`.

import dotenv from 'dotenv'

dotenv.config()

import { createApp } from './app.js'

const PORT = process.env.PORT || 3000

createApp().listen(PORT, () => console.log(`Server on http://localhost:${PORT}`))

**E7. Configurarea testelor — Vitest & Jest „umăr la umăr”**

\*\*Vitest\*\* suportă direct ESM; setăm doar mediul de test ca `node`. \*\*Jest\*\*, în schimb, e mai sensibil la ESM.  
Îi adăugăm `babel-jest` și `@babel/preset-env` pentru a transforma importurile ESM la runtime.

// vitest.config.ts

import { defineConfig } from 'vitest/config'

export default defineConfig({ test:{ environment:'node' } })

// jest.config.cjs

module.exports = {

testEnvironment: 'node',

transform: { '^.+\.(js|mjs)$': 'babel-jest' },

extensionsToTreatAsEsm: ['.js', '.mjs']

};

// babel.config.cjs

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

**E8. Teste de contract pentru rute (Vitest)**

Folosim `supertest` pentru a simula request‑uri HTTP fără a porni efectiv un port. Aserțiile validează \*\*forma\*\* răspunsurilor.

import { describe, it, expect } from 'vitest'

import request from 'supertest'

import { createApp } from '../../src/app.js'

const app = createApp()

describe('Seminar 0 — Express minimal (Vitest)', () => {

it('GET /ping -> pong', async () => {

const res = await request(app).get('/ping')

expect(res.status).toBe(200)

expect(res.text).toBe('pong')

})

it('GET /api/time -> { now: ISO }', async () => {

const res = await request(app).get('/api/time')

expect(res.status).toBe(200)

expect(typeof res.body.now).toBe('string')

expect(Number.isNaN(Date.parse(res.body.now))).toBe(false)

})

it('GET / -> serves index.html', async () => {

const res = await request(app).get('/')

expect(res.status).toBe(200)

expect(res.text).toContain('<h1>StudentHub Minisite</h1>')

})

})

**E9. Teste echivalente în Jest**

Aceleași aserții, dar rulând pe Jest. Transformarea ESM e asigurată de `babel-jest`.

import { describe, it, expect } from '@jest/globals'

import request from 'supertest'

import { createApp } from '../../src/app.js'

const app = createApp()

describe('Seminar 0 — Express minimal (Jest)', () => {

it('GET /ping -> pong', async () => {

const res = await request(app).get('/ping')

expect(res.status).toBe(200)

expect(res.text).toBe('pong')

})

it('GET /api/time -> { now: ISO }', async () => {

const res = await request(app).get('/api/time')

expect(res.status).toBe(200)

expect(typeof res.body.now).toBe('string')

expect(Number.isNaN(Date.parse(res.body.now))).toBe(false)

})

it('GET / -> serves index.html', async () => {

const res = await request(app).get('/')

expect(res.status).toBe(200)

expect(res.text).toContain('<h1>StudentHub Minisite</h1>')

})

})

**E10. Rulare și debugging**

Instalează dependențele cu `npm i`. Rulează `npm test` — ar trebui să vezi suitele Vitest și Jest trecând.  
Pornește serverul cu `npm run dev` (cu `nodemon`) sau `npm start` (fără watch). Dacă portul 3000 este ocupat, setează `PORT=3001`  
într‑un fișier `.env` local (nu îl comite) și repornește.

**AI‑assist (VSL) — cum să câștigi timp fără a pierde controlul**

Exersează cereri scurte, precise: „\*\*Create an Express app factory\*\* `createApp()` with `express.json()`, `express.static('public')`, `/ping`, `/api/time`.”, „\*\*Write Vitest tests\*\* for `/ping` & `/api/time` with `supertest`.”, „\*\*Mirror Jest suite\*\* for the same assertions.”.   
După ce primești cod, \*\*rulează\*\* testele; dacă ceva e neclar, cere \*\*explicații\*\* („why this approach”, „edge cases”).

**Troubleshooting (cele mai frecvente erori)**

• `ERR\_MODULE\_NOT\_FOUND`: verifică extensiile la importuri (ESM cere `.js`).   
• `EADDRINUSE: 3000`: schimbă `PORT` (folosește `.env`).   
• `SyntaxError: Cannot use import statement outside a module` (Jest): confirmă `babel-jest` + `extensionsToTreatAsEsm`.   
• 404 la `GET /`: verifică `express.static('public')` și calea relativă `../public` din `app.js`.

**Criterii de evaluare (laborator)**

(1) Toate elementele din checklist sunt îndeplinite; (2) testele trec în ambele suite; (3) codul este ESM, curat și bine structurat; (4) README prezintă pașii de rulare; (5) opțional: adaugi `.env.example`, `.gitignore` adecvat; (6) explici clar în commituri ce ai schimbat și de ce.