Table of Contents

# Seminarul 0 — Prolog & Setup complet (toolchain, inițializare proiect, server minimal)

**Autor:** Cadrul didactic (coordonator)  
**Data:** 2025-09-22  
**Nivel:** Începători / Novici  
**Format:** Ghid pas cu pas, cu exemple testabile imediat în **Node.js ≥ 18** și în **Firefox → F12 (Consolă)**

## Cuprins

* [0. Rezumat executiv (tl;dr)](#X5125adf5f2d6fda3cc75e82196beae944543a0e)
* [1. Obiective de învățare & rezultate așteptate](#Xccab2485a607e0080cc6cf3c090f84f22afa2cc)
* [2. Mediu & reflex de testare rapidă](#X16c28bb046356de9e037c6c53b6478e638a7021)
  + [2.1. Experimente de 10–30 secunde (Node & Firefox)](#X80e90a36ae895cb236dc0624da029eca0969616)
  + [2.2. Comenzi compuse în shell: explicarea &&, ; și |](#X01a724a43297ad93cfee5699c59d67a6636f19a)
* [3. Instalare & configurare pe Ubuntu Desktop 25.04 LTS (pas cu pas)](#Xae2bb1b105fe00351f5e15782cdbfbf0465b535)
  + [3.1. Actualizare sistem & utilitare de bază](#X210f4bfec8216f372af26182c0cf0ab2b01fc91)
  + [3.2. Git + GitHub (SSH, configurare globală)](#X90b8942b0b878b3179ac725ed10c30a0951ff7f)
  + [3.3. Node.js ≥ 18 (cu nvm, alternativ apt)](#Xdf63a7e84b3d85cf214752159927dce771ac5d4)
  + [3.4. Editor & unelte: VS Code / Codium, ESLint, Prettier](#X09a0785123479586e99dc4c9794e1b76fd5f1b1)
  + [3.5. Testare: Vitest & Jest (în oglindă)](#X336854222b44ab7865aac22f13415e1cb7f41c7)
  + [3.6. Browser: Firefox pentru dezvoltare (DevTools + add‑ons utile)](#Xcc661070ea91a71058e33746c84e8329c8f8af3)
  + [3.7. Pandoc + LibreOffice (pentru export DOCX/PDF din Markdown)](#X4aad9a89595f12fd9485ad088a240e581f7a1fc)
  + [3.8. SQL local: SQLite3 CLI (pentru seminariile 9+)](#X37b5e16e8e2225f8e918aacb6b4ef1ae7c7a8a0)
  + [3.9. Alternative opționale: PNPM/Yarn, Workspaces, BrowserSync](#X0d69ef275230280671dc2739be6498fef2c30cd)
* [4. Teorie: HTTP vs HTML, Node.js vs browser, proiect minimal](#X4fdf7366b3a61763211a6bc07be8c5a14c26436)
* [5. Aplicații/Cod/Comenzi (rulabile imediat)](#X2b6c1b4793aa25a68d1eccd9675d2dd561655de)
  + [Ex.1 — „Hello web” static + server Express minimal](#Xf86b76b673d4dfae4bc109ae698c09f1b2326f1)
  + [Ex.2 — Rută de sănătate GET /ping → pong](#ex2--rută-de-sănătate-get-ping--pong)
  + [Ex.3 — API de timp GET /api/time → {{ now: ISO-8601 }}](#Xcc77cb2c273404bd7f89dc56605a36ff3d7f41b)
  + [Ex.4 — Script CLI: citirea argumentelor (process.argv)](#Xff040fc90121e8d3bd8c45c04bb689e2f46f86a)
  + [Ex.5 — Testare rapidă (Vitest + Jest) pe același contract](#Xf79f2e9623f1ee3ff3568ce41841a176a4052bc)
* [6. Mini‑laborator progresiv (L1→L2→L3)](#Xe0b2af62a835b767605e23e32bfb1d1c277afaf)
* [7. Anti‑pattern‑uri, capcane & troubleshooting](#X7a4f5b50f762c9b1c63645cc0987273c9f4388c)
* [8. Prompts utile (ChatGPT & GitHub Copilot)](#X7108255c97785b070080050d247ee097d63c82d)
* [9. Rubrică de evaluare & Definition of Done](#X05d3337a08173f3493a81a3280991a3f60127ea)
* [10. Întrebări de auto‑verificare + răspunsuri scurte](#X70a20412db0119218a91e3cfabf00d433d156a6)
* [11. Bibliografie (APA 7) & Standarde/Specificații](#X9301dae1fd70c005e11bd79c14378cc7076e505)

## 0. Rezumat executiv (tl;dr)

1. Diferență de rol: **HTTP** = protocol de transfer; **HTML** = limbaj de marcare.
2. **Node.js** rulează JavaScript pe server/CLI; **browserul** rulează JS în pagină, în sandbox.
3. Inițializezi un proiect în 3 pași: mkdir … && cd … && npm init -y → npm i express → cod minimal index.js.
4. Servești fișiere statice din public/, plus rute simple (/ping, /api/time).
5. Configurezi calitatea: **ESLint + Prettier**; teste „în oglindă”: **Vitest & Jest**.
6. Testezi rapid orice fragment în **Firefox (F12 → Console)** sau în **Node REPL**.
7. Git & GitHub: inițializezi repo, legi la remote, împingi codul.
8. Robusteză: folosește timeouts/AbortController doar când e logic (de exemplu, la fetch).
9. Comparații JS ↔ Python ↔ C++: modele diferite (dinamic vs static), dar ideile rămân aceleași.
10. **Definition of Done**: server pornește, rutele răspund corect, lint+teste verzi, README clar.

## 1. Obiective de învățare & rezultate așteptate

* Să distingi clar **protocol** (HTTP) de **limbaj** (HTML) și de **runtime** (Node).
* Să instalezi și să configurezi **toolchain‑ul complet** pe Ubuntu 25.04 LTS.
* Să creezi rapid un **proiect Node** cu un **server Express** care servește static și răspunde JSON.
* Să înțelegi **comenzile compuse** în shell și să poți explica fiecare componentă.
* Să rulezi **micro‑experimente** în Node/Firefox și să validezi output‑urile.
* Să configurezi **ESLint, Prettier** și **Vitest/Jest** pentru un workflow sănătos.
* Să folosești **Git/GitHub** pentru versionare și colaborare.
* Să poți formula **prompts** utile pentru AI‑assist, în stil VSL (Verify → Specify → Learn).

## 2. Mediu & reflex de testare rapidă

### 2.1. Experimente de 10–30 secunde (Node & Firefox)

**Node REPL:**

node  
> 2 + 3  
5  
> ((a,b)=>a+b)(2,3)  
5  
> process.version

**Firefox (F12 → Consolă):**

[1,2,3].map(x => x\*2) // [2, 4, 6]  
globalThis.location.href // URL-ul paginii curente

### 2.2. Comenzi compuse în shell: explicarea &&, ; și |

* A && B → execută **B doar dacă A reușește** (cod 0).
* A ; B → execută **B indiferent** de rezultatul lui A.
* A | B → trimite **stdout‑ul lui A ca stdin** în B (pipeline).

Ex.:

mkdir tw-seminar-0 && cd tw-seminar-0 && npm init -y

1. mkdir creează directorul; 2) cd intră în el (doar dacă mkdir a reușit); 3) npm init -y generează package.json cu răspunsuri implicite.

## 3. Instalare & configurare pe Ubuntu Desktop 25.04 LTS (pas cu pas)

Scop: o singură dată, un **setup complet** pentru toate cele 13 seminarii.

### 3.1. Actualizare sistem & utilitare de bază

sudo apt update && sudo apt -y upgrade  
sudo apt -y install build-essential curl wget ca-certificates gnupg git unzip zip

### 3.2. Git + GitHub (SSH, configurare globală)

git --version  
git config --global user.name "Prenume Nume"  
git config --global user.email "you@example.org"  
ssh-keygen -t ed25519 -C "you@example.org"  
cat ~/.ssh/id\_ed25519.pub

Adaugă cheia publică în **GitHub → Settings → SSH and GPG keys**. Testează:

ssh -T git@github.com

### 3.3. Node.js ≥ 18 (cu nvm, alternativ apt)

**Varianta recomandată (nvm):**

curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash  
# Reîncarcă shell-ul, apoi:  
nvm install --lts  
node -v && npm -v

**Alternativ apt (poate instala versiuni mai vechi):**

sudo apt -y install nodejs npm  
node -v && npm -v

### 3.4. Editor & unelte: VS Code / Codium, ESLint, Prettier

* Instalează **VS Code** (Snap) sau **VSCodium**:

sudo snap install code --classic # sau: sudo snap install codium --classic

* În proiect:

npm i -D eslint prettier  
npx eslint --init # alege ESM, Browser+Node, style standard sau airbnb

**Prettier**: creează .prettierrc.json:

{ "semi": true, "singleQuote": true, "trailingComma": "es5" }

### 3.5. Testare: Vitest & Jest (în oglindă)

npm i -D vitest jest babel-jest @babel/preset-env @types/jest

package.json:

{  
 "type": "module",  
 "scripts": {  
 "test:vitest": "vitest run --reporter verbose",  
 "test:jest": "jest --runInBand",  
 "test": "npm run test:vitest && npm run test:jest"  
 }  
}

jest.config.cjs:

module.exports = {  
 testEnvironment: 'node',  
 transform: { '^.+\.m?js$': 'babel-jest' },  
 extensionsToTreatAsEsm: ['.js']  
};

babel.config.cjs:

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

### 3.6. Browser: Firefox pentru dezvoltare (DevTools + add‑ons utile)

* **DevTools**: F12 → Inspector, Console, Network, Storage.
* **Add‑ons utile**: *JSON Viewer*, *React DevTools* (pentru seminariile 10–12).
* Preferințe: activați „**Enable browser chrome and add‑on debugging toolboxes**” când aveți nevoie de debugging avansat.

### 3.7. Pandoc + LibreOffice (pentru export DOCX/PDF din Markdown)

sudo apt -y install pandoc libreoffice

Export rapid:

pandoc Seminar0\_Prolog\_Setup\_extins.md -o Seminar0\_Prolog\_Setup\_extins.docx

### 3.8. SQL local: SQLite3 CLI (pentru seminariile 9+)

sudo apt -y install sqlite3  
sqlite3 --version

### 3.9. Alternative opționale: PNPM/Yarn, Workspaces, BrowserSync

sudo npm i -g pnpm yarn  
sudo npm i -g browser-sync

## 4. Teorie: HTTP vs HTML, Node.js vs browser, proiect minimal

* **HTTP**: protocol de transport pentru cereri/răspunsuri (metode, status codes, antete).
* **HTML**: limbaj de marcare pentru structurarea documentelor.
* **Node.js**: runtime JS pe server/CLI; are module, acces la FS, rețea.
* **Browser**: rulează JS în pagină, cu API‑uri DOM, restricții de securitate (CSP, CORS).
* **Proiect minimal**: structură clară public/ (statice) + index.js (server).

## 5. Aplicații/Cod/Comenzi (rulabile imediat)

### Ex.1 — „Hello web” static + server Express minimal

**(a) Specificație.** Servește /public/index.html la http://localhost:3000/.

**(b) Cod minimal.**

mkdir tw-seminar-0 && cd tw-seminar-0 && npm init -y  
npm i express && npm i -D nodemon  
mkdir public  
printf '<!doctype html><html><body><h1>hello web</h1></body></html>' > public/index.html

index.js:

import express from 'express';  
const app = express();  
app.use(express.static('public'));  
app.listen(3000, () => console.log('http://localhost:3000'));

**(c) Rulare.**

node index.js # sau: npx nodemon index.js (autoreload)

**(d) De ce funcționează.** express.static('public') mapează fișierele din director la rădăcină.

**(e) JS ↔ Python ↔ C++ (mini‑echivalente).**

* *Python (Flask)*:

from flask import Flask, send\_from\_directory  
app = Flask(\_\_name\_\_)  
@app.route('/')  
def home(): return send\_from\_directory('public', 'index.html')  
app.run(port=3000)

* *C++ (cpp-httplib)*:

#include "httplib.h"  
int main(){  
 httplib::Server svr;  
 svr.set\_mount\_point("/", "./public");  
 svr.listen("0.0.0.0", 3000);  
}

**(f) Test rapid.** Deschide http://localhost:3000/ → vezi „hello web”.  
**(g) Erori tipice.** Port ocupat (EADDRINUSE) → schimbă portul; folder greșit → verifică public/.

### Ex.2 — Rută de sănătate GET /ping → pong

**(a) Specificație.** La GET /ping serverul răspunde text pong.

**(b) Cod.**

app.get('/ping', (req,res)=> res.type('text/plain').send('pong'));

**(c) Verificare.**

curl -i http://localhost:3000/ping

**(d) Mecanism.** Express face *routing* pe metodă+cale.  
**(e) Echivalente.**

* *Python Flask*: @app.get('/ping') → return 'pong'
* *C++ httplib*: svr.Get("/ping", [](const auto&, auto& res){ res.set\_content("pong","text/plain"); }); **(f) Test (Vitest).** expect('pong').toBe('pong') (schelet).  
  **(g) Capcane.** res.send() dublu → eroare.

### Ex.3 — API de timp GET /api/time → { now: ISO-8601 }

**(a) Specificație.** JSON cu timpul curent în ISO‑8601.  
**(b) Cod.**

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

**(c) Verificare.**

curl -s http://localhost:3000/api/time

**(d) De ce.** res.json setează automat Content-Type.  
**(e) Echivalente.** Python/C++ după caz.  
**(f) Test rapid.** Date.parse(...) e valid.  
**(g) Edge‑cases.** Fus orar.

### Ex.4 — Script CLI: process.argv

**(a) Specificație.** Suma numerelor din linia de comandă.  
**(b) Cod (sum-cli.js).**

const xs = process.argv.slice(2).map(Number);  
const ok = xs.every(Number.isFinite);  
if (!ok) { console.error('Folosește numai numere'); process.exit(1); }  
console.log(xs.reduce((a,b)=>a+b,0));

**(c) Rulare.** node sum-cli.js 1 2 3 → 6.  
**(d) Mecanism.** argv conține argumentele utilizatorului.  
**(e) Echivalente.** Python/C++ minimal.  
**(f) Test.** Intrări non‑numerice → exit 1.  
**(g) Capcane.** Number('') → 0.

### Ex.5 — Testare în oglindă (Vitest + Jest)

**(a) Specificație.** add(2,3)=5.  
**(b) Cod src/add.js.**

export const add = (a,b) => a + b;

**(c) Test Vitest & Jest** (schelet ca mai sus).  
**(d) Rulare.** npm run test.  
**(e) De ce.** Portabilitate a suitei.  
**(f) Capcane.** ESM + Jest → babel-jest.

## 6. Mini‑laborator progresiv (L1→L2→L3)

* **L1:** proiect minimal (express.static, /ping, /api/time), README, lint OK.
* **L2:** adaugă CLI, teste Vitest+Jest, repo GitHub.
* **L3:** withTimeout + AbortController pentru fluxuri externe.

## 7. Anti‑pattern‑uri, capcane & troubleshooting

* „Totul în index.js” → structurează în module.
* Fără scripturi în package.json → adaugă start, dev, test.
* ESM + Jest fără configurare → adaugă babel-jest.
* Port 3000 ocupat → schimbă portul.
* Ignori codurile de ieșire în CLI → folosește process.exit(1) la erori.

## 8. Prompts utile (ChatGPT & GitHub Copilot)

* **Testare:** „Generează 10 *edge‑cases* pentru /api/time și scrie teste Vitest care validează formatul ISO‑8601.”
* **Refactor:** „Împarte serverul în app.js (exportă app) și index.js (bootstrap).”
* **Comparativ:** „Oferă echivalent minimal în Python/C++ pentru /ping.”
* **Robustețe:** „Adaugă withTimeout pentru un fetch și scrie teste care verifică respingerea după 50ms.”

## 9. Rubrică de evaluare & Definition of Done

* **Corectitudine (40%)**, **Calitate cod (25%)**, **Testare (20%)**, **Documentare (10%)**, **Git workflow (5%)**.  
  **DoD:** server funcțional, rute verificate, lint+teste verzi, README clar, repo public.

## 10. Întrebări de auto‑verificare + răspunsuri scurte

1. HTTP vs HTML? → Protocol vs limbaj de marcare.
2. express.static? → Servește fișiere direct din director.
3. process.argv.slice(2)? → Argumentele utilizatorului.
4. ESM+Jest? → babel-jest + extensionsToTreatAsEsm.
5. Când AbortController? → Anulezi operații asincrone nefolositoare.

## 11. Bibliografie (APA 7) & Standarde/Specificații

### Literatură științifică (cu DOI)

* Wirfs‑Brock, A., & Eich, B. (2020). *JavaScript: The first 20 years*. **Proceedings of the ACM on Programming Languages, 4**(HOPL), 1–189. <https://doi.org/10.1145/3386327>
* Maffeis, S., Mitchell, J. C., & Taly, A. (2008). *An operational semantics for JavaScript*. In **Programming Languages and Systems** (pp. 307–325). Springer. <https://doi.org/10.1007/978-3-540-89330-1_22>
* Loring, M. C., Laurenzano, M. A., Newsham, Z., Hovsmith, N., Pande, S., Barik, T., & others. (2017). *Semantics of asynchronous JavaScript*. **ACM SIGPLAN Notices, 52**(11), 51–62. <https://doi.org/10.1145/3170472.3133846>
* Michie, D. (1968). “Memo” functions and machine learning. **Nature, 218**(5136), 19–22. <https://doi.org/10.1038/218019a0>
* Ungar, D., & Smith, R. B. (1991). SELF: The power of simplicity. **Higher‑Order and Symbolic Computation, 4**(3), 171–216. <https://doi.org/10.1007/BF01806105>
* Ungar, D., & Smith, R. B. (1987). Self: The power of simplicity. In **OOPSLA ‘87**. ACM. <https://doi.org/10.1145/38765.38828>

### Standarde/Specificații (fără DOI)

* ECMAScript® Language Specification (ECMA‑262), TC39.
* HTTP/1.1 (RFC 7230+), IETF.
* HTML Living Standard, WHATWG.

Acest prolog pregătește terenul pentru toate cele 13 seminarii și poate fi exportat în DOCX prin pandoc.