Node A01 — Portfolio Launchpad

Node & Express + Hybrid Pages + Projects API + Client Render
(Starter UI + Starter Data)

Goal

Build an Express server that:

- serves HTML pages from `src/pages/` (no templating yet),
- serves static assets from `/public`, and
- exposes a JSON API for project data that the Projects page renders dynamically using `fetch()` + vanilla JS.

Important: Your WebDevTnT A05 portfolio UI is due next weekend. For A01, you'll use the provided starter UI + starter dataset. Next week you'll swap in your A05 UI. (Yes, this is intentional. No, you don't get to suffer twice.)

What I Provide (Starter Pack)

- `data/projects.sample.json` (6 projects, correct schema)
- `src/pages/*.html` starter shells (simple but valid)
- `public/css/styles.css` starter stylesheet
- `public/js/projects.js` starter JS file (you will complete it)

You may modify the starter HTML/CSS, but you are not graded on "portfolio design" yet.

Learning Outcomes

- Express routing + middleware
- Static hosting via `express.static`
- Page routing via `res.sendFile()` with safe pathing
- Simple REST-ish JSON endpoints
- Reading JSON data from disk
- `fetch()` + DOM rendering with vanilla JS
- Basic code organization using Express Routers

Required Tech

- Node.js + Express
- Vanilla frontend JS (no frameworks)
- Plain CSS (single file)

Required Repo Structure

Day 01 structure: single `server.js`, plus a minimal `routes` folder for page and API routers. No `app.js`.

```
None
/server.js

/src
  /server
    /routes
      pages.routes.js
      api.routes.js
  /pages
    index.html
    about.html
    projects.html
    contact.html

/public
  /css
    styles.css
  /js
    projects.js
  /images
    /projects
      /<slug>
        cover.png
        screen.png

/data
  projects.json

ai-interaction-log.txt
README.md
package.json
```

Rule:

- `src/pages` = HTML pages served by Express routes
- `public` = static assets (CSS/JS/images)

- `src/server/routes` = Express Router modules (page routes + API routes)

NPM Scripts (Required)

Your repo must support:

- `npm run dev` — runs the server in watch mode
- `npm start` — starts the server normally (no watch)

Required scripts:

```
None
{
  "scripts": {
    "dev": "node --watch server.js",
    "start": "node server.js"
  }
}
```

AI Usage Requirement

You must include an **ai-interaction-log.txt** in the root of your repo describing how you used AI (if at all) while completing this assignment.

Required contents:

- Which tool(s) you used (ChatGPT, Copilot, etc.)
- What you used it for (debugging, generating starter code, explaining concepts, etc.)
- At least 3 bullet points summarizing specific prompts/questions you asked
- What you changed after receiving the AI output (i.e., your own decisions)

Note: Using AI is allowed and encouraged. Not logging your usage is not.

Data Contract (Required)

Create `/data/projects.json` by copying the provided sample file. Minimum: **6 projects**.

Top-level shape:

```
None
{
  "projects": [ ... ]
```

```
}
```

Each project object must match:

```
None
{
  "id": "p-2001",
  "slug": "vanilla-js-game",
  "title": "Neon Dodger",
  "tagline": "One short sentence",
  "description": "A short paragraph",
  "status": true,
  "stack": ["node", "express", "mongodb"],
  "tags": ["crud", "api", "routing"],
  "images": [
    { "path": "/images/projects/vanilla-js-game/cover.png",
"alt": "Cover image description", "type": "cover" },
    { "path": "/images/projects/vanilla-js-game/screen.png",
"alt": "Screenshot description", "type": "screenshot" }
  ],
  "dates": { "created": "2026-01-08", "updated": "2026-02-01"
}
}
```

Rules

- **IDs must be unique and stable.**
- **Slugs must be unique** and URL-safe (kebab-case).
- `status` is a **boolean**:
  - `true` = active (shown in the list)
  - `false` = inactive (not shown in the list, but still fetchable by id)
- `stack` and `tags` are arrays of strings.
- `images` contains **exactly 2** objects:
  - `type` must be `"cover"` or `"screenshot"`
  - `path` must begin with `/images/` (served from `public`)
  - `alt` must be meaningful (not “image”)
- `dates.created` and `dates.updated` must be ISO format: `YYYY-MM-DD`.

Projects to Include (SSD-aligned)

Your dataset must include these 6 project themes (names can vary, but keep the intent):

- Vanilla JavaScript Game (invent one)
- React Movie Database
- News-Style Landing Page
- Node/Express/Mongo Data-Driven Portfolio
- Ecommerce Site
- One more (you invent) — pick something program-relevant (API, auth, Docker, testing, etc.)

Server Requirements

Middleware (Required)

- request logger (`morgan('dev')` or equivalent)
- `express.static('public')`
- `express.json()` enabled
- `express.urlencoded({ extended: true })` enabled (for contact form)

Router Requirement (Required)

You must use Express Routers:

- `src/server/routes/pages.routes.js` handles all page routes
- `src/server/routes/api.routes.js` handles all `/api` routes

Mount these routers from `server.js`:

- `app.use('/', pagesRouter)`
- `app.use('/api', apiRouter)`

Pages Routes (Required)

Serve these pages from `src/pages` using `res.sendFile()`:

- `GET /` → `src/pages/index.html`
- `GET /about` → `src/pages/about.html`
- `GET /projects` → `src/pages/projects.html`
- `GET /contact` → `src/pages/contact.html`

Implementation rule: No hard-coded absolute paths. Use `path.join()` (or `path.resolve()`) with a safe base directory.

Contact Route (Required)

`POST /contact`

- Accept: `name`, `email`, `message`
- Log submission on the server
- Return a simple success response (200)

(No persistence yet. That's later.)

404 Handling (Required)

- Unknown `/api/*` routes: JSON `{ "error": "Not found" }`
- Other unknown routes: simple text 404 is fine

API Requirements

1) List Projects

`GET /api/projects` — returns an array of projects

Required behavior:

- Return only projects where `status === true`

Optional filter: `GET /api/projects?q=tailwind`

Filter matches any of:

- `title`, `tagline`, `description`
- any item in `stack`
- any item in `tags`

2) Project Details

`GET /api/projects/:id` — returns one project object

If not found: 404 with:

```
None
{ "error": "Project not found" }
```

Response Rules

- `/api/*` always returns JSON
- Correct status codes (200, 404, 500)

Frontend Requirements (Projects Page)

Starter HTML

`src/pages/projects.html` (provided) includes:

- a container element (e.g. `<div id="projectsGrid"></div>`)
- a details area (e.g. `<section id="projectDetails"></section>`)
- a script reference to `/js/projects.js`
- a stylesheet link to `/css/styles.css` (single CSS file)

Required JS Behaviour

`public/js/projects.js` must:

- `fetch('/api/projects')`
- render project cards into the container
- each card shows: `title`, `tagline`, and **stack chips**
- each card includes a “Details” button/link that loads the chosen project

Details Interaction (Required)

When “Details” is clicked:

- `fetch('/api/projects/:id')`
- render details into the details area (or a modal)
- details view must include:
  - `title`, `tagline`, `description`
  - `tags` chips
  - both images using `images[0].path`/`alt` and `images[1].path`/`alt`
  - basic error handling (if fetch fails, show a message)

Styling Requirements (Lightweight)

Plain CSS is fine. You are not graded on high-end design yet.

Minimum:

- Projects list is readable and spaced
- Cards have basic layout and visual separation
- Details area is visually distinct from the list
- Only one CSS file is loaded: `/css/styles.css`

Submission

- GitHub repo link
- Repo must include: `ai-interaction-log.txt`

- `README.md` must include:
  - install steps
  - how to run dev and start
  - list of routes (pages + API)
  - any notes/assumptions

Must run cleanly:

```
None
npm install
npm run dev
```

Marking (100)

- **Core server setup (35)** — middleware, static assets, router usage, page routing via sendFile, 404 handling
- **API (35)** — list (+ active-only), optional filter, detail + 404, correct JSON/status codes
- **Frontend dynamic projects (30)** — fetch + render cards, details fetch + render (incl. images), basic error handling

Stretch Goals (Bonus)

- search input wired to `?q=`
- sort projects (by `dates.updated` or `title`)
- better empty/error UI (loading state, “no matches”, “not found”)

Forward Link

Next week (templating week) you will:

- replace `src/pages/projects.html` with a templated view
- keep the same `projects.json` data contract and API routes
- optionally plug in your A05 UI as the actual portfolio design layer
- optionally introduce a CSS pipeline + multi-process dev workflow (not this week)