The Shed, explained

MapLarge Developer Test Project | built by Steven Stout | 2026-09-29

Live theshed.stevenstout.biz Code github.com/SteveStout/TheYard/tree/main/samples/maplarge Resume and About on the site

A file and folder browser on the starter you sent, built the way I build my own project: shape and tests first, every decision written down, and a live site.

What it is

An ASP.NET Core 10 API over one configurable home directory and one page in TypeScript with no framework, where the address bar is the whole state. Browse, search by name or wildcard, upload with a progress bar, download, make folders, move, copy and delete, inside a native dialog with one trigger. Every failure is a sentence the page shows, never a status code. Twelve decision records are served from the running app with the code they decided about read from the build as you read, and the site runs beside my own project at no added cost.

```
dotnet run               # localhost:5120, browsing sample-home/
dotnet test              # 111 xUnit tests, warnings as errors
npm install && npm test  # 21 node tests over the compiled modules
```

Or skip the build: **/?path=reports/2026** opens the dialog in a folder, **/?q=*.csv&sort=size&dir=desc** is a sorted search as a link, **/?view=docs&doc=adr-003-the-line-a-path-cannot-cross** is the security record with its code live. The README is the map; ADR-003, ADR-005 and ADR-009 are the three to read first.

The brief, and where each item is

You asked	Where it is
Builds in VS, Rider, VS Code, CLI	TestProject.sln, your file kept by name, on .NET 10; the app references no package.
JSON API: browse and search	FilesController; snake_case; one RFC 9457 problem document on every failure, with a trace id.
Home directory by variable	Files:Home or FILES__HOME. HomePath is the one class that turns a request path into a real one.
Deep-linkable state in the URL	Six query keys, one parser and serializer, one navigate(). Back, reload and a pasted link take the same path.
Vanilla SPA, client rendered	TypeScript compiled by tsc alone; src/lib is pure, src/ui renders; no innerHTML anywhere, and a test proves it.
Upload and download	Multipart with a bar per file, 409 answered in place; downloads stream with range requests.
Counts and sizes per view	Totals on every reply, computed on the server for exactly what is listed, plus took_ms.
Bonus: dialog, writes, performance	A native dialog and one button; delete, move and copy ask in the row; a 10,000-file measurement.

How it was built and shipped

- **Green before every commit.** 132 tests, warnings as errors, a headless Chrome pass of the running page (deep links, search, Back, upload, delete, Escape) with zero console errors.
- **Live at no added cost.** A third web app on the App Service plan my project runs on, its own free managed certificate, the image in the same registry, rolled by a workflow whose identity holds Website Contributor on that one site and nothing wider. The Bicep and the workflow are in the folder.
- **Built with AI, and written down (ADR-011).** I decided the shape and the test list first, Claude drafted against them, the build ran on my machine, and I read everything before it shipped. My first push had a folder-name case collision Windows hides and Linux does not; the container found it, and the link test now checks case exactly.

By the numbers

132 tests green before every commit

< 1 ms to browse 100 files, measured by a test

47 ms to search 10,000 files whole; capped, 6 ms

0 warnings, analyzers at their strictest

12 records served live with their code

$0 added hosting, on a plan I already pay for

Measured over 10,000 files on the build machine (ADR-008)

Six choices, and why

1 Your starter, kept
Your names, your controller shape, your one page. Three lines added to the project file (ADR-001).

2 Dependencies inward
One project, four folders. A test reads every using and fails on one that points the wrong way (ADR-002).

3 One pure security class
Three string refusals before any filesystem touch, then containment; the same tests on both OSes (ADR-003).

4 The address is the state
Nothing is kept anywhere else. The first keystroke of a search pushes; the rest replace (ADR-005).

5 Measured, not claimed
Search is lazy, stops at a cap and says so; a test prints the timings on every run (ADR-008).

6 Rules beside their tests
A table of every standing rule and the test that holds it; a test reads the table (ADR-009).

Taking it further: my larger project is the worked example

TheYard (theyard.stevenstout.biz) is the same practice at full size: .NET 10 and React, Azure SQL and Cosmos DB, 86 records, about 1,930 test runs per gate, two sites on one plan. Each next step is something the Yard already does, so the cost is known.

Next on the Shed	How the Yard already does it
A committed browser test	138 Playwright specs in the gate, located by role and name; axe holds nine views to WCAG 2.1 AA.
Paging behind the search cap	The inventory API pages 100,000 vehicles behind Load more; the page never asks for the whole set.
Logs that outlive the container	Every request and error kept in Cosmos DB for three years, read on the Admin tab (ADR-071, 073).
The path guard as a Files card	The Admin tab is already a product: health, machines, timing, kept logs, each a card (ADR-080).

The pitch. The Shed is your brief, delivered on your starter **(1)** with dependencies a test keeps pointing inward **(2)**, a security model small enough to read in one sitting **(3)**, deep links that are the whole state rather than a copy of it **(4)**, performance that is measured on every run instead of promised **(5)**, and every rule kept beside the test that holds it **(6)**: 132 tests green, live at no added cost, and the same working method that runs my own production site. Happy to walk any file with the team.