How We Built the Global Automation Atlas Website

Global Automation Atlas. Garg, Crosta and Baier, 2026.
automationatlas.org

Purpose. This note describes how we turned the research pipeline behind the Global Automation Atlas into a public website. The website presents the paper's release files as maps, paper figures, methods diagrams, and download tables. The aim was to make the empirical object visible without separating the website from the research pipeline.

Empirical object. The paper classifies O*NET task statements across countries. The core object is a task-country label: one structured classification for a task in a country. The website keeps that object stable across pages. It shows countries, O*NET tasks, task-country labels, occupations, industries, ESCO skills, and the task-skill network.

Main website surfaces.

- **Home** introduces the project through a country map and headline coverage numbers.
- **Atlas** lets readers inspect countries, the task-skill network, occupations, and industries.
- **Paper** contains the paper title, abstract, PDF, prompt protocol, and interactive versions of the main paper figures.
- **Methods** explains the task-country measurement workflow and the task-skill construction workflow.
- **Data** provides the release files and codebook.

Release files. The website reads structured files exported from the research pipeline. Country maps, paper figures, tables, modals, and downloads use those files as their source of record. The files include country summaries, task-country labels, occupation panels, industry panels, skill panels, paper-figure data, prompt protocol files, and task-skill graph files.

Technical stack. We used a conventional web stack, with each tool doing a narrow job:

- Astro, a framework for building fast static websites, provides the page structure.
- React, a library for interactive interface components, powers the Atlas views and paper figures.
- TypeScript adds type checks to the frontend code.
- D3 helps turn data into maps and charts.
- TopoJSON stores country borders in a compact format for web maps.
- SVG and Canvas render figures, methods diagrams, and network views in the browser.
- GitHub records code changes and keeps the work reviewable.
- Cloudflare Pages hosts the deployed website.

AI-assisted coding workflow. We used Claude Code and Codex as coding assistants. They helped read handoff notes, edit components, refactor layouts, generate methods diagrams, run builds, check browser screenshots, and apply visual feedback.

The research team stayed in the loop. The team made the scientific choices, measurement definitions, textual claims, interpretation, page structure, and final design decisions. The coding assistants shortened the implementation loop.

How we worked. The useful workflow was batch-based. We read comments, chose one bounded website problem, made the local code change, built the site, inspected screenshots and browser behavior, iterated, then committed and deployed the finished batch.

This browser review changed the website. We repeatedly checked spacing, oversized titles, redundant cards, unclear controls, stale links, hover behavior, modal design, footer rhythm, and whether a first-time visitor would know where to click.

Design choices. The website improved as it became more restrained. We reduced redundant headings, removed unsupported sections, kept text short, and let figures and interactive views carry more of the explanation. Secondary definitions often moved into tooltips.

The Atlas became the main working surface. Paper, Methods, Data, and About each got a clearer job. Product/trade views, income explorer pages, comparison pages, and task finder pages were removed or deferred when their reader value was too low.

Lessons for another research website.

- Name the empirical object early and build the site around it.
- Export stable web files from the research pipeline.
- Build the main interactive surface before adding extra pages.
- Make paper figures interactive where hover or click changes what readers learn.
- Keep comments, deferred ideas, and implementation batches in plain files.
- Use coding assistants for implementation, refactors, screenshots, and deployment checks.
- Keep claims, interpretation, and design judgment with the research team.
- Review browser screenshots before deployment.

Practical setup checklist. For a similar project, we would start with four files: a country or unit-level summary file, a long file at the main unit of analysis, a figure-data folder, and a codebook. The website can then be built around those files before adding richer features. This prevents the site from becoming a collection of disconnected pages.

The first version should answer three questions for a new reader: what is being measured, where the main object varies, and which files support the claims. More elaborate pages can be added after those questions are clear.

What made it work. The paper had a clear data object, and the feedback process stayed organized. GitHub and Cloudflare made changes testable and deployable. AI coding tools made iteration faster. The research team decided what the website should say, show, remove, and defer.