Building dbenger.com: An AI-Assisted Portfolio Case Study

How I Built an Interactive Resume Web App in 3 Days with Claude Code

Dominik Benger — February 2026

Table of Contents

1. Project Overview

The Problem

Static PDF resumes are limited. They're flat, one-dimensional documents that force 10+ years of career experience into bullet points on a page. Recruiters scan them in 30 seconds. There's no way to filter by relevance, explore career progression visually, or engage with the content interactively.

I wanted something better.

The Vision

Replace my PDF resume with a fully interactive web application — a living document where visitors could:

- **Explore** my career trajectory through animated SVG visualizations
- **Filter** my experience by industry, role type, or technical capability
- **Interact** with 77 skills across 7 categories in a filterable tag grid
- **Connect** directly through an integrated contact form
- **Download** a traditional PDF when they needed one

The result is dbenger.com — an animated, responsive web app built with Next.js 15, Tailwind CSS, Framer Motion, and custom SVG visualizations.

The Constraint

I built the entire project in approximately **3 days** using Claude Code as my primary development partner. No design team, no Figma mockups, no prior React/Next.js project in this domain. Just a 4-page resume PDF, a product requirements document, and an AI coding assistant.

What It Became

The final site features:

- **3 interactive visualizations** — Career Path (animated SVG with year-proportional node positioning), Skills & Tech Stack (filterable tag grid with 77 skills), and Industry Verticals (proportional bar chart with detail cards)
- **Filterable experience timeline** — 8 career entries with expandable sections, company logos, technology tags, and key projects
- **Collaboration page** — 10 service offerings with accordion deliverables, 3 packages, and a tooling showcase
- **Contact form** — Formspree-powered with honeypot spam protection
- **Full accessibility** — skip-to-content, ARIA tablist/tabpanel, keyboard navigation, `prefers-reduced-motion` support
- **Mobile-first responsive design** — tested at 320px, 375px, 768px, and 1024px+

---

2. Tech Stack & Tools

Core Framework & Libraries

Layer	Technology	Version	Purpose
Framework	Next.js	15.x	App Router, SSG, Turbopack dev server
Language	TypeScript	5.7	Type safety across all data and components
UI Library	React	19.x	Component model
Styling	Tailwind CSS	3.4	Utility-first CSS with custom design tokens
Animations	Framer Motion	11.x	Scroll animations, page transitions, SVG path drawing
Visualizations	Custom SVG	—	Hand-built, no charting library
Analytics	Vercel Analytics	1.x	Page views and Web Vitals
Contact Form	Formspree	—	Serverless form handling
Font	Plus Jakarta Sans	—	Loaded via `next/font/google`

Development & Deployment

Tool	Purpose
ChatPRD	AI product management — helped draft the initial Product Requirements Document
Codex	AI research assistant — helped refine PRD structure, user stories, and requirements
Claude Code	AI pair-programming — wrote the vast majority of code, docs, and config
Git / GitHub	Version control, hosted at github.com/Ninety2UA/web-app-resume
Vercel	Hosting, CDN, SSL, custom domain management

Tool	Purpose
npm	Package management
ESLint	Code quality with `eslint-config-next`

What I Didn't Use (and Why)

- **No charting library** (Chart.js, Recharts, D3) — Custom SVG gave me full control over the Career Path visualization's aesthetics and animations without bundle bloat.
- **No CMS** — Career data lives in TypeScript files under `src/data/`. For a single-user site, a CMS adds complexity without value.
- **No test framework** — The site is entirely presentational with no business logic. Visual QA via browser and Playwright snapshots was sufficient.
- **No database** — Everything is statically generated at build time. Zero runtime data fetching.

> **Key Insight:** The "no charting library" decision saved significant bundle size and gave me pixel-perfect control over the Career Path visualization — including year-proportional node positioning, custom bezier curves, and per-node label placement. A charting library would have fought me on all of these.

---

3. Project Setup & Configuration

Project Initialization

The project started from a standard Next.js 15 scaffold with TypeScript, Tailwind CSS, and the App Router enabled:

```
{
  "dependencies": {
    "next": "^15.1.0",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "framer-motion": "^11.15.0",
    "@vercel/analytics": "^1.4.0"
  }
}
```

Five production dependencies. That's the entire runtime footprint.

Tailwind Configuration

The design system was encoded directly in `tailwind.config.ts` with a warm color palette:

```
colors: {
  warm: {
    50: '#FDFCFA',   // Page background
    100: '#F5F0EB',  // Card surfaces
    200: '#EDE6DE',  // Borders
    600: '#6B6560',  // Secondary text
    900: '#1A1814',  // Primary text
  },
  coral:    { DEFAULT: '#E07A5F' },  // Primary accent
  teal:     { DEFAULT: '#4A9B8E' },  // Secondary accent
  amber:    { DEFAULT: '#E6B35A' },  // Tertiary accent
  lavender: { DEFAULT: '#7C6FB0' },  // Quaternary accent
}
```

Custom box shadows, animation keyframes, and the Plus Jakarta Sans font family completed the config. The entire Tailwind setup fits in 80 lines.

Next.js Configuration

Minimal and intentional:

```
const nextConfig: NextConfig = {
  images: { formats: ['image/avif', 'image/webp'] },
  reactStrictMode: true,
  poweredByHeader: false,
}
```

The `poweredByHeader: false` removes the `X-Powered-By: Next.js` header — a small security best practice. AVIF image support was enabled for future optimization.

Global Styles

The global CSS (`globals.css` , 70 lines) handles:

- Smooth scrolling, font smoothing
- Custom selection color (coral tint)

- Focus-visible outlines using the coral accent
- Custom scrollbar styling (warm palette)
- A `scrollbar-hide` utility for clean horizontal scroll areas
- A `prefers-reduced-motion` media query that forces near-zero animation durations globally

```
@media (prefers-reduced-motion: reduce) {
  *, *::before, *::after {
    animation-duration: 0.01ms !important;
    transition-duration: 0.01ms !important;
  }
}
```

> **Design Decision:** Using `0.01ms` instead of `0ms` prevents browsers from skipping `animationend` / `transitionend` events entirely. Framer Motion animation props still fire — they just complete instantly.

---

4. Architecture & Design Decisions

Page Architecture

The site is a single-page scrollable app with two additional routes:

```
/                     → Hero + Visualizations + Filters + Experience + Contact +
/collaboration        → Service offerings, packages, working style
/portfolio            → Project cards (hidden from nav — preserved for v2)
```

All three routes are **statically generated** (SSG) at build time. There are no API routes, no server-side data fetching, and no client-side data fetching. The entire site is served as pre-rendered HTML with hydrated React components for interactivity.

Component Tree

```
Layout (all pages)
├── FloatingNav (fixed, always visible, z-50)
├── Skip-to-content link (z-[100])
└── Vercel Analytics

/ (Home)
├── HeroSection
│   ├── VisualizationToggle (tablist: 3 tabs)
│   ├── RoleEvolution (SVG + aligned HTML timeline)
│   ├── SkillsTechStack (interactive tag grid)
│   ├── IndustryVerticals (stacked bar + cards)
│   └── TimelineMarkers (shown for non-Career-Path tabs)
├── FilterPills (sticky, inclusive OR toggle chips)
├── ExperienceSection
│   ├── ExperienceCard × N (main roles)
│   ├── ExperienceCard × N (additional, compact)
│   └── Education
├── ContactSection (Formspree POST)
└── Footer

/collaboration
├── OfferingsGrid → OfferingCard (accordion deliverables)
├── PackageCards (3-tier comparison + add-ons)
└── WorkingStyleSection (principles + tool pills)
```

Data Architecture

All career data lives in `src/data/` as typed TypeScript objects:

File	Contents	Records
`experience.ts`	Career entries, education, filter tags	8 entries, 7 filter tags
`skills.ts`	Skill categories, industry data, role progression nodes	7 categories, 77 skills, 7 role nodes
`offerings.ts`	Service offerings, packages, add-ons, tool categories	10 offerings, 3 packages
`portfolio.ts`	Project cards (placeholder)	6 projects

The `ExperienceEntry` interface captures the full structure:

```
interface ExperienceEntry {
  id: string
  company: string
  role: string
  team?: string
  location: string
```

```
    startDate: string          // "YYYY-MM"
    endDate: string | null     // null = "Present"
    summary: string
    sections: { title: string; bullets: string[] }[]
    keyProjects?: { title: string; description: string; link?: string }[]
    technologies: string[]
    industries: string[]       // maps to filter tag IDs
    roleType: string[]         // maps to filter tag IDs
    logo?: string              // path to company logo
    isAdditional?: boolean     // renders in compact "Additional Experience" sect
  }
```

Why TypeScript over JSON? TypeScript data files give us type checking at build time. If I misspell a filter tag ID, the build catches it. JSON would require runtime validation.

Design System: The Warm Palette

The entire visual language is built around a "warm & approachable" aesthetic:

Token	Hex	Role
Background	`#FDFCFA`	Warm off-white page background
Surface	`#F5F0EB`	Card and panel backgrounds
Primary text	`#1A1814`	Warm near-black for headings
Secondary text	`#6B6560`	Muted text for metadata
Terra Cotta	`#E07A5F`	Primary accent — CTAs, active states, links
Sage Teal	`#4A9B8E`	Data/Analytics accent
Amber	`#E6B35A`	Marketing accent
Lavender	`#7C6FB0`	AI/Cloud accent

Three extended colors (Sky `#5B8DB8`, Rose `#D4697A`, Emerald `#4A9B6E`) are used exclusively in the Skills & Tech Stack visualization as inline styles — not in the Tailwind config — to keep the global palette lean.

Z-Index Strategy

A deliberate stacking order prevents overlap issues:

Element	Z-Index	Position
Sticky filter bar	`z-30`	`sticky top-[68px]`
FloatingNav	`z-50`	`fixed top-4`
Skip-to-content	`z-[100]`	Absolute on focus

The filter bar uses `top-[68px]` to sit below the FloatingNav (which is ~44px tall at `top-4`). This relationship required careful measurement — getting it wrong caused the filter bar to overlap the nav on initial scroll.

---

5. Development Workflow with Claude Code

How the Project Was Actually Built

This project was built using **Claude Code** (Anthropic's CLI for Claude) as the primary development tool. I provided the direction, requirements, and design feedback; Claude Code wrote the code, config, documentation, and handled debugging.

The Document-First Approach

Before writing a single line of code, I created three foundational documents:

1. **PRD (Product Requirements Document)** — ~900 lines covering goals, user stories, functional requirements, UX flows, success metrics, and technical considerations. I used **ChatPRD** and **Codex** to draft and refine the initial PRD — ChatPRD helped structure the product thinking (goals, user stories, success metrics), while Codex helped research best practices and sharpen the requirements.
2. **Technical Specification** — ~380 lines defining the tech stack, data models, design system tokens, component specs, and performance budgets
3. **Implementation Plan** — ~470 lines breaking the project into 32+ tasks across 8 phases with explicit dependencies

These documents served as the "prompt" for the entire project. When I started a Claude Code session, the first thing it did was read these files to understand the full scope.

> **Key Strategy:** Front-loading the documentation paid massive dividends. Using ChatPRD and Codex for the PRD meant the requirements were well-structured before any code was written. Claude Code could then reference the spec when making

> implementation decisions, and the task tracker gave both of us a shared understanding of progress and priorities.

The CLAUDE.md File

The project's `CLAUDE.md` file (the project-level instruction file for Claude Code) grew to ~200 lines over the course of development. It served as persistent memory across sessions and included:

- **Startup ritual** — mandatory steps to read status docs and git state before making changes
- **Tech stack reference** — quick lookup for versions, libraries, and patterns
- **Architecture decisions** — documented choices so they wouldn't be revisited
- **Pitfalls section** — known gotchas (z-index relationships, SVG text rendering, `.gitignore` exceptions)
- **Accessibility patterns** — ARIA patterns used across the app
- **Session continuity notes** — what was done last, what's deployed, what's next

This file was the single most important artifact for multi-session development. Without it, every new session would have started from scratch.

Task Breakdown and Phasing

The project was organized into 15 phases:

Phase	Scope	Tasks
0	Project setup (Next.js, Tailwind, fonts, PDF)	T01–T05
1	Data architecture (experience, skills, portfolio)	T06–T09
2	Layout & navigation (root layout, nav, footer)	T10–T12
3	Hero section (header, viz toggle, 4 visualizations, timeline)	T13–T19
4	Filters & experience (pills, scroll animation, cards)	T20–T23
5	Contact & footer	T24
6	Portfolio page	T25–T27
7	Main page assembly (wire everything up)	T28
8	Polish (responsive, accessibility, performance, QA)	T29–T32
9	Collaboration page	T34

Phase	Scope	Tasks
10	UI rework (chart positioning, nav cleanup)	U06–U10
11	UI tweaks (logo positioning, padding, tooling)	U11–U13
12	Documentation (README, CLAUDE.md sync)	D01–D02
13	Deployment (Vercel, custom domain)	L03
14	Mobile layout fixes	U14–U15

Phases 0–7 were completed in the **first commit** (`fb6a036`) — over 10,000 lines of code across 41 files. This was the "autonomous full build" phase where Claude Code worked through the plan end-to-end.

Multi-Agent Parallelism

For Phase 8 (Polish & Optimization), I used Claude Code's agent system to parallelize work across 4 agents:

- **Agent 1**: Responsive design for hero components
- **Agent 2**: Responsive design for experience and contact components
- **Agent 3**: Accessibility audit across all layouts
- **Agent 4**: Performance analysis and optimization

This parallel approach had a key lesson: **agents blocked on file writes when permission prompts were unavailable**. The solution was to split files by exclusive ownership (zero overlap between agents) to avoid merge conflicts.

The Iteration Loop

After the initial build, the workflow settled into a tight feedback loop:

1. **Review the live site** (via `npm run dev`)
2. **Identify issues** (visual, behavioral, content)
3. **Describe the issue to Claude Code** (e.g., "the chart labels overlap on nodes 1 and 2")
4. **Claude Code fixes the code** and explains the change
5. **Verify** in the browser
6. **Commit** when a batch of fixes is stable

Each post-launch commit addressed a cluster of related issues: UI overlap fixes, chart label rework, visualization merges, mobile layout fixes. The commit history tells the story of progressive refinement.

6. Key Features Built

Feature 1: Career Path Visualization (RoleEvolution)

The centerpiece of the site — an animated SVG chart showing career progression from 2014 to 2025.

How it works:

```
roleNodes data → yearToX() positioning → SVG bezier path → Framer Motion animat
```

Each career milestone is a node positioned proportionally by year:

```
const yearToX = (year: number) =>
  padding.left + ((year - minYear) / yearRange) * chartW
```

The path connecting nodes uses cubic bezier curves:

```
const pathData = nodes.reduce((acc, node, i) => {
  if (i === 0) return `M ${node.x} ${node.y}`
  const prev = nodes[i - 1]
  const cpx = (prev.x + node.x) / 2
  return `${acc} C ${cpx} ${prev.y} ${cpx} ${node.y} ${node.x} ${node.y}`
}, '')
```

The path draws itself using Framer Motion's `pathLength` animation:

```
<motion.path
  d={pathData}
  initial={{ pathLength: 0 }}
  animate={{ pathLength: 1 }}
  transition={{ duration: 1.5, ease: [0.37, 0, 0.63, 1] }}
/>
```

Nodes appear sequentially with staggered delays (0.3s base + 0.18s per node), creating a "filling in" effect that follows the path.

Label placement was the hardest part. Early-career nodes (2014–2018) are bunched in the left ~36% of the chart, so labels needed custom diagonal offsets to avoid overlapping circles and curves:

```
const labelPlacements = [
  { dx: 0,   titleDy: 38,  anchor: 'start' },   // Node 0: below
  { dx: 24,  titleDy: 30,  anchor: 'start' },   // Node 1: below-right
  { dx: 24,  titleDy: 30,  anchor: 'start' },   // Node 2: below-right
  { dx: -24, titleDy: -44, anchor: 'end' },     // Node 3: above-left
  { dx: 24,  titleDy: 30,  anchor: 'start' },   // Node 4: below-right
  { dx: 0,   titleDy: -50, anchor: 'middle' },  // Node 5: above
  { dx: 0,   titleDy: -50, anchor: 'end' },     // Node 6: above-right
]
```

An aligned HTML timeline sits below the SVG, sharing the same `yearToX()` percentages for pixel-perfect alignment. This was moved from inside the SVG to HTML for better text rendering and responsive behavior.

> **Pitfall Discovered:** SVG `textAnchor="end"` near the left edge of the viewBox clips text off-screen. Node 1 had to use `textAnchor="start"` with a positive `dx` offset instead of `"end"` with negative `dx` .

Feature 2: Skills & Tech Stack (Interactive Tag Grid)

A filterable grid of 77 skills across 7 color-coded categories.

Categories:

- Data / Analytics (13 skills) — teal
- Marketing Platforms (8 skills) — coral
- Cloud / Infrastructure (10 skills) — amber
- AI / LLM Tools (8 skills) — lavender
- Developer Tools (22 skills) — sky
- Workflow Automation (6 skills) — rose
- Generative Media (10 skills) — emerald

Clicking a category filter isolates that category; clicking again (or "All") resets. The transition uses Framer Motion's `AnimatePresence` with staggered tag animations:

```
<motion.span
  initial={{ opacity: 0, scale: 0.85 }}
  animate={{ opacity: 1, scale: 1 }}
  transition={{
    duration: 0.25,
    delay: catIdx * 0.06 + skillIdx * 0.02,
  }}
/>
```

Colors are defined as inline styles via a `colorMap` object, not Tailwind classes. This was deliberate — the extended palette (sky, rose, emerald) didn't need to exist in the global Tailwind config:

```
const colorMap = {
  teal:    { dot: '#4A9B8E', bg: '#EFF8F6', text: '#3B7D72', border: '#4A9B8E30
  sky:     { dot: '#5B8DB8', bg: '#EEF4F9', text: '#4A7396', border: '#5B8DB830
  // ... 5 more
}
```

Feature 3: Filterable Experience Section

Experience cards filter dynamically using pill toggle buttons with inclusive OR logic:

- Clicking "Apps & Gaming" shows all entries tagged with that industry
- Clicking multiple pills shows entries matching **any** selected filter
- Clicking "All" resets

Filter state lives at the page level and flows down:

```
export default function Home() {
  const [activeFilters, setActiveFilters] = useState<string[]>([])
  return (
    <>
      <FilterPills activeFilters={activeFilters} onFilterChange={setActiveFilte
      <ExperienceSection activeFilters={activeFilters} />
    </>
  )
}
```

The sticky filter bar sits below the floating nav with a careful `top-[68px]` offset, `z-30` stacking, and a `backdrop-blur-sm` frosted glass effect.

Feature 4: Scroll-Triggered Animations

A custom `useScrollAnimation` hook drives reveal animations:

```
export function useScrollAnimation(threshold = 0.2) {
  const ref = useRef<HTMLDivElement>(null)
  const [isVisible, setIsVisible] = useState(false)

  useEffect(() => {
    if (window.matchMedia('(prefers-reduced-motion: reduce)').matches) {
      setIsVisible(true)
      return
    }

    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          setIsVisible(true)
          observer.unobserve(entry.target)  // trigger once only
        }
      },
      { threshold, rootMargin: '0px 0px -80px 0px' }
    )
    // ...
  }, [threshold])

  return { ref, isVisible }
}
```

Key design choices:

- **Fires once** — `observer.unobserve()` after first intersection prevents re-animation on scroll up
- **Respects reduced motion** — immediately sets `isVisible = true`, bypassing the observer entirely
- **80px bottom margin** — elements appear before they're fully in viewport, creating a natural reveal

---

Feature 5: Contact Form with Formspree

A client-side form that POSTs to Formspree's API:

```
const res = await fetch('https://formspree.io/f/mojnqgnq', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(formData),
})
```

Four states: `idle` → `submitting` (spinner) → `success` (checkmark) → `error` (fallback mailto link). A hidden honeypot field provides spam protection:

```
<input type="text" name="_gotcha" style={{ display: 'none' }}
       tabIndex={-1} aria-hidden="true" autoComplete="off" />
```

Feature 6: Collaboration Page

A full service offerings page (`/collaboration`) built from a content document (`docs/Offering.md`):

- **10 service offerings** with accordion-style deliverables using Framer Motion `AnimatePresence` and `height: 0 → auto` animation
- **3 packages** (Audit / Build / Operate) in a comparison card layout
- **Working style section** with principle cards and tool category pills

The accordion pattern:

```
<AnimatePresence initial={false}>
  {isOpen && (
    <motion.div
      initial={{ height: 0, opacity: 0 }}
      animate={{ height: 'auto', opacity: 1 }}
      exit={{ height: 0, opacity: 0 }}
      transition={{ duration: 0.3, ease: [0.04, 0.62, 0.23, 0.98] }}
      className="overflow-hidden"
    >
      {/* deliverables list */}
    </motion.div>
  )}
</AnimatePresence>
```

Feature 7: Floating Navigation

An always-visible pill-shaped nav bar, fixed at the top center:

```
<nav className="fixed top-4 left-1/2 -translate-x-1/2 z-50">
  <div className="flex items-center gap-0.5 sm:gap-1 bg-warm-50/80
    backdrop-blur-md border border-warm-200 rounded-full
    px-1.5 sm:px-2 py-1 sm:py-1.5 shadow-nav">
    {/* links + PDF download button */}
  </div>
</nav>
```

This went through several iterations:

1. **v1**: Hidden on load, appeared on scroll with `AnimatePresence` wrapper
2. **v2**: Always visible, but with a hamburger menu on mobile
3. **v3 (final)**: Always visible, same horizontal pill layout at all breakpoints with responsive text/padding sizes

The final version removed ~87 lines of code (scroll listener, useState, AnimatePresence, hamburger menu, dropdown). Simpler was better.

---

7. Challenges & Solutions

Challenge 1: SVG Label Overlap in Year-Proportional Chart

Problem: When nodes are positioned by actual year (2014–2025), early-career roles (2014–2018) bunch in the left ~36% of the chart. Labels for adjacent nodes overlap each other and the bezier curves.

Solution: A per-node `labelPlacements` array with diagonal offsets. Bunched nodes alternate between above-left and below-right placement, keeping text clear of both circles and the rising curve. Node 1 was a special case — too close to the SVG left edge for `textAnchor="end"`, requiring below-right placement with `textAnchor="start"`.

Iterations: This took 3 revisions (U04, U05, then the full U06–U10 rework) to get right.

Challenge 2: Mobile Chart Horizontal Scroll

Problem: The `RoleEvolution` SVG had `min-w-[500px]` to prevent it from becoming unreadably small on narrow viewports. This forced a horizontal scroll container — a poor mobile experience.

Solution: Removed the min-width entirely and let the SVG scale naturally via `viewBox="0 0 900 440"` + `preserveAspectRatio="xMidYMid meet"`. The chart becomes small on 320px screens but remains legible and eliminates the jarring horizontal scroll.

> **Pitfall Recorded:** SVG `min-w-[500px]` forces horizontal scroll on mobile — prefer letting `viewBox` + `preserveAspectRatio` handle scaling naturally.

Challenge 3: Nav/Filter Bar Z-Index Overlap

Problem: After launch, the sticky filter bar overlapped the floating nav when scrolling. The filter bar used `sticky top-0`, placing it at the same position as the FloatingNav.

Solution: Changed the filter bar to `sticky top-[68px]` — calculated by measuring the FloatingNav height (~44px) plus its `top-4` offset (16px) plus breathing room. Documented the z-index stacking order in both `CLAUDE.md` and `README.md`.

Challenge 4: `next/image` Dimension Warnings with Company Logos

Problem: Company logos have varying aspect ratios. `next/image` requires explicit `width` and `height` props, and mismatches produce console warnings.

Solution: Used plain `<img>` tags with an ESLint disable/enable block:

```
{/* eslint-disable @next/next/no-img-element */}
<img src={logo} alt="" className="h-7 w-auto max-w-[80px] object-contain" />
{/* eslint-enable @next/next/no-img-element */}
```

The logos are small static PNGs (3–153KB) that don't benefit from Next.js's image optimization pipeline. The build size actually **decreased** from 162kB to 156kB by removing the `next/image` module import.

Challenge 5: Cross-Page Navigation with Anchor Links

Problem: `#section` hrefs in the FloatingNav didn't work when navigating from `/collaboration` back to the home page. The anchor tried to find the element on the current page.

Solution: Prefix all anchor links with `/` — using `/#top` , `/#experience` instead of `#top` , `#experience` . Next.js handles the route change to `/` first, then scrolls to the anchor.

Challenge 6: ESLint Disable in JSX Conditional Expressions

Problem: `eslint-disable-next-line` doesn't work inside conditional expressions (`&&`) when the target element is on a non-adjacent line.

Solution: Use `eslint-disable` / `eslint-enable` block pairs instead of `eslint-disable-next-line` .

Challenge 7: Merging Two Visualizations into One

Problem: The original design had 4 visualization tabs: Career Path, Skills, Industries, Tech Stack. The Skills (bar chart) and Tech Stack (timeline) tabs felt incomplete individually and didn't showcase the full breadth of 77 skills.

Solution: Merged them into a single "Skills & Tech Stack" interactive tag grid with category filters. This reduced tabs from 4 to 3, created a more engaging interaction pattern, and showcased all 77 skills at once. The merge deleted two components (`SkillsProgression.tsx` , `TechStack.tsx`) and replaced them with `SkillsTechStack.tsx` .

---

8. Documentation & Project Management

The Document Stack

The project maintained 6 documentation files that served as the "control plane" for development:

Document	Lines	Purpose
`PRD.md`	~890	Requirements, user stories, success metrics

Document	Lines	Purpose
`Spec.md`	~380	Tech stack, data models, component specs, performance budgets
`Plan.md`	~470	Phased task breakdown with dependencies
`tasks.md`	~270	Task tracker with completion status and detail sections
`STATUS.md`	~140	Current state summary, decisions made, pitfalls
`CLAUDE.md`	~200	AI assistant instructions, startup ritual, patterns

How They Were Used

PRD → Written first, before any code. Defined what "done" looks like. Referenced throughout for scope decisions (e.g., "shareable filter URLs" was explicitly cut from v1 scope).

Spec → The bridge between requirements and implementation. Data model interfaces were copy-pasted from the spec into TypeScript files. Design system tokens were transferred directly into `tailwind.config.ts`.

Plan → The execution roadmap. Tasks were organized with explicit dependencies (`T28` depends on `T13–T24`), which enabled Claude Code to work through them in the correct order during the initial autonomous build.

tasks.md → The living task tracker. Every completed task got a "Detail" section documenting what was done, what changed, and what to watch for. This turned the tracker into a changelog and audit trail.

STATUS.md → The "executive summary" — quick reference for project state, recent commits, and known issues. Updated at the end of each session.

CLAUDE.md → The most novel document. It's essentially a persistent instruction set for the AI assistant, including a mandatory startup ritual:

```
## Startup Ritual
At the start of every new session, before taking any action:
1. Read `docs/STATUS.md` and `docs/tasks.md` (in parallel)
2. Check git state (status, log, diff)
3. Summarize current status to the user
4. Wait for instructions before making changes
```

This ritual prevented the most common failure mode: an AI assistant making changes without understanding the current state.

Session Continuity

Each development session ended with a "session wrap" that updated the docs:

1. Update `STATUS.md` with what was accomplished
2. Update `tasks.md` with completed task details
3. Update `CLAUDE.md` if new patterns or pitfalls were discovered
4. Commit the doc updates

This created a chain of context that survived across sessions. The AI didn't need to re-discover the project state — it read the docs and picked up where it left off.

> **Key Takeaway:** Documentation isn't overhead when working with AI — it's infrastructure. The docs were the mechanism that made multi-session development coherent.

---

9. Lessons Learned

On AI-Assisted Development

> **"Front-load your docs."** The more context you provide upfront (PRD, spec, plan), the less course-correction you need during implementation. The initial autonomous build (10,000+ lines in one commit) was possible because the spec was thorough enough for Claude Code to make correct decisions without asking.

> **"CLAUDE.md is your most important file."** It's the persistent memory that makes sessions composable. Every pitfall documented, every pattern recorded, every architectural decision captured — they compound across sessions.

> **"Review everything."** AI-generated code is not reviewed code. Every commit in this project was manually verified in the browser at multiple viewport sizes. The AI got the structure right ~90% of the time; the remaining 10% was the difference between "works" and "polished."

On Architecture

> **"Fewer dependencies, fewer problems."** Five production dependencies. Zero API routes. Zero database. Every decision to not add something made the project simpler to build, debug, and deploy.

> **"Custom SVG beats chart libraries for bespoke visualizations."** The Career Path chart needed year-proportional positioning, custom bezier curves, per-node label placement, and Framer Motion integration. No charting library offers all of that without fighting the API.

> **"Static TypeScript data is underrated."** Type-checked data files with no runtime overhead. Changes are caught at build time. No CMS complexity for a single-user site.

On Process

> **"Ship fast, then iterate."** The initial build was deployed live, then refined over several focused sessions: UI overlap fixes, chart rework, visualization merge, mobile layout, logo integration. Each iteration addressed a specific cluster of issues.

> **"Document your pitfalls."** The pitfalls section in `CLAUDE.md` prevented the same mistakes from recurring. SVG `textAnchor="end"` near edges, `.gitignore` blocking PNGs, `next/image` dimension warnings — each was encountered once and never again.

> **"Parallel agents need exclusive file ownership."** When running multiple AI agents, split files so no two agents write to the same file. Merge conflicts with AI-generated code are painful to resolve.

On Design

"Warm palettes create approachability." The deliberate choice of `#FDFCFA` (warm off-white) over pure white, and `#1A1814` (warm near-black) over pure black, creates a softer, more inviting feel. Small HSL shifts make a large perceptual difference.

"Navigation simplicity wins." The FloatingNav went through 3 iterations before arriving at the simplest version: a fixed horizontal pill bar with no scroll triggers, no hamburger menu, no animation wrappers. The final version is 47 lines of code.

---

10. Final Results & Reflections

By the Numbers

Metric	Value
Development time	~3 days (Feb 8–10, 2026)
Total commits	18
Source files	~30 TSX/TS files
Lines of code (initial commit)	10,581
Production dependencies	5
First Load JS (home)	158 kB (gzipped)
CSS bundle	25 kB (Tailwind purge)
Pages generated (SSG)	3
Build warnings	0
Skills showcased	77 across 7 categories
Experience entries	8 (with education)
Visualizations	3 interactive views

What Worked

1. **The document-first approach** — PRD → Spec → Plan → Build. The upfront investment in docs made the autonomous build phase possible.
2. **Claude Code for rapid prototyping** — Going from zero to a deployed, responsive, accessible web app in 3 days would not have been feasible without AI assistance. The tool excelled at translating structured requirements into working components.
3. **Custom SVG visualizations** — They're the differentiating feature of the site. Visitors engage with the Career Path chart in a way they never would with a static timeline.
4. **Progressive refinement** — The "ship, then iterate" approach kept momentum high. Each commit improved specific aspects without trying to achieve perfection in one pass.
5. **Persistent documentation** — The docs survived across sessions and prevented context loss. They also serve as the foundation for this very ebook.

What I'd Do Differently

1. **Start with mobile** — The first build was desktop-primary. Mobile fixes came later (commit `1f47fd8`). Starting mobile-first would have avoided the chart horizontal scroll issue entirely.
2. **Design in the browser sooner** — Some label placement issues (RoleEvolution nodes 1-4) required 3 iterations to resolve. A quicker feedback loop between code changes and visual inspection would have caught these earlier.
3. **Add Lighthouse CI** — While the build passes clean, automated Lighthouse scoring on each commit would catch performance regressions before they ship.

What's Next

- **Portfolio page v2** — The `/portfolio` route exists with placeholder content. Real case studies and project showcases are planned.
- **Blog / writing section** — Expanding the site from a resume into a personal brand platform.
- **Analytics refinement** — Custom event tracking for visualization interactions (which tabs are viewed most, how deep do visitors scroll).
- **Performance optimization** — Dynamic imports for visualizations, font weight subsetting, lazy loading below-fold content.

Final Thought

This project started as a replacement for a 4-page PDF. It became a case study in how AI-assisted development changes the economics of personal projects. What would have been a multi-week effort for a solo developer became a weekend sprint. The tools didn't write the vision — they executed it. The taste, the design decisions, the "should this be a chart or a grid?" choices — those remain human. But the implementation velocity? That's transformed.

The site is live at dbenger.com. The code is open at github.com/Ninety2UA/web-app-resume.

Appendix A: Commit Timeline

Date	Hash	Description
Feb 8	`fb6a036`	Initial commit: complete interactive resume (Phases 0–7, 10,581 lines)
Feb 8	`34e5062`	Phase 8: responsive design, accessibility, performance, launch prep
Feb 8	`53bdd97`	Fix post-launch UI issues (nav/filter overlap, chart labels, padding)
Feb 8	`47ba309`	Merge Skills + Tech Stack visualizations into single interactive tab
Feb 8	`18a2ea5`	Add Collaboration page with 10 service offerings
Feb 9	`b5e26ea`	UI rework: year-proportional chart, aligned timeline, nav cleanup
Feb 9	`2d65169`	Add company logos, full resume content, nav fix, README
Feb 10	`6863c84`	Comprehensive README rewrite, CLAUDE.md sync
Feb 10	`e8f9cae`	Sync project docs with latest commit history
Feb 10	`78bb8f6`	Mark deployment complete — live at dbenger.com
Feb 10	`ec931fc`	Update RIT logo with new version
Feb 10	`1f47fd8`	Fix mobile layout: remove chart scroll, inline pill nav
Feb 10	`648994d`	Fix Google intern dates to Jan 2017 – Aug 2017

Date	Hash	Description
Feb 10	`d083605`	Update downloadable resume PDF to V3

Appendix B: File Inventory

Source Files (src/)

```
src/app/
├── layout.tsx                    Root layout (fonts, metadata, analytics
├── page.tsx                      Home page (Hero + Filters + Experience
├── globals.css                   Global styles + Tailwind directives
├── icon.svg                      Favicon (DB monogram)
├── opengraph-image.png           OG image (1200×630)
├── collaboration/page.tsx        /collaboration route
└── portfolio/page.tsx            /portfolio route (hidden)

src/components/
├── layout/FloatingNav.tsx        Always-visible pill navigation
├── layout/Footer.tsx             Footer with social links
├── hero/HeroSection.tsx          Hero wrapper (viz toggle, viz area, tim
├── hero/VisualizationToggle.tsx  3-tab toggle buttons
├── hero/TimelineMarkers.tsx      Era markers (non-Career-Path tabs)
├── hero/visualizations/RoleEvolution.tsx      SVG career path chart
├── hero/visualizations/SkillsTechStack.tsx    Interactive skill tag grid
├── hero/visualizations/IndustryVerticals.tsx  Stacked bar + detail cards
├── experience/ExperienceSection.tsx  Filtered experience container
├── experience/ExperienceCard.tsx     Individual experience card
├── filters/FilterPills.tsx       Toggle filter chips
├── contact/ContactSection.tsx    Formspree contact form
├── portfolio/PortfolioGrid.tsx   Project card grid
├── portfolio/ProjectCard.tsx     Individual project card
├── collaboration/OfferingCard.tsx    Service offering with accordion
├── collaboration/OfferingsGrid.tsx   2-column offering grid
├── collaboration/PackageCards.tsx    3-tier package comparison
└── collaboration/WorkingStyleSection.tsx  Principles + tool pills

src/data/
├── experience.ts                 8 career entries + education + filter t
├── skills.ts                     7 skill categories + industry data + ro
├── offerings.ts                  10 offerings + 3 packages + tools
└── portfolio.ts                  6 project cards (placeholder)
```

```
src/hooks/
└── useScrollAnimation.ts              IntersectionObserver scroll trigger

src/lib/
└── utils.ts                           cn() classname merger + formatDateRange
```

Configuration Files

```
package.json          Dependencies and scripts
next.config.ts        Next.js config (images, strict mode)
tailwind.config.ts    Design system tokens (colors, shadows, animations)
tsconfig.json         TypeScript config (strict, path aliases)
postcss.config.mjs    PostCSS + Autoprefixer
.eslintrc.json        ESLint config
.gitignore            Ignores with PNG exceptions
```

Documentation Files (docs/)

```
PRD.md                          Product Requirements Document (~890 lines)
Spec.md                         Technical Specification (~380 lines)
Plan.md                         Implementation Plan (~470 lines)
tasks.md                        Task Tracker (~270 lines)
STATUS.md                       Project Status (~140 lines)
Dominik_Benger_Resume_4Page.md  Source resume content
Offering.md                     Service offerings source
resume-file/                    Original PDF resume
```

Built with Next.js 15, Tailwind CSS, Framer Motion, and Claude Code. Live at dbenger.com | Source at github.com/Ninety2UA/web-app-resume