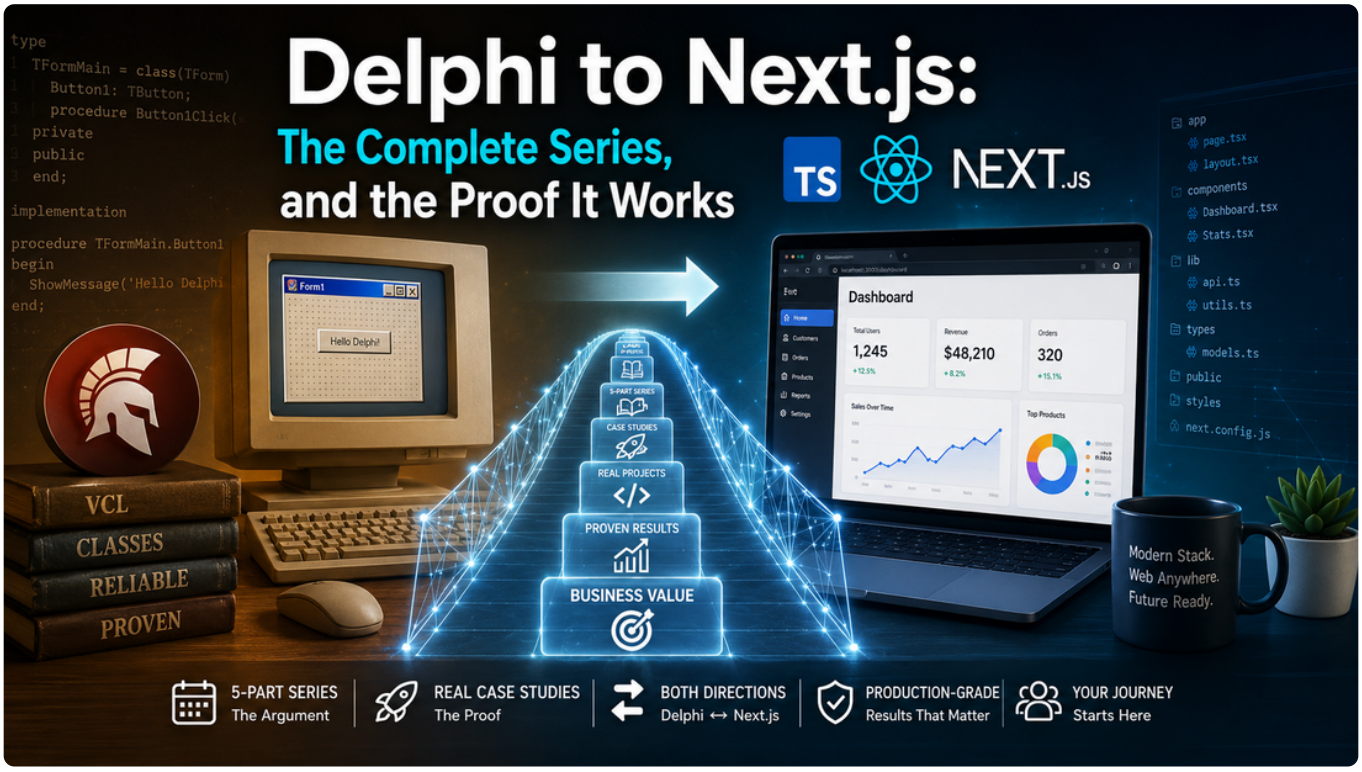
NEWS · JULY 8, 2026

Delphi to Next.js: The Complete Series, and the Proof It Works

By Dr. Holger Flick



In late 2025 I published a five-part series arguing that a Delphi developer — someone with decades of Object Pascal muscle memory — could pick up TypeScript, React, and Next.js faster than they feared, and that doing so was a sound business decision, not a betrayal of a beloved tool. The series went on to become the most-read content I have ever published on this blog.

Back then, the series was an *argument*. It rested on concept mappings — Delphi interfaces to TypeScript interfaces, VCL components to React components, project structure to framework structure — and on migration strategies I had used in consulting work. What it could not yet include was public, documented proof.

Throughout 2026, that changed. Between February and July I published a string of case studies that put the series' claims through production-grade reality: a Delphi 7 legacy application rebuilt as a full Next.js web platform in a one-week sprint, a six-year-old Delphi multi-tier app migrated to a modern Next.js stack in about four hours — and, running in the other direction, projects that prove you can *keep* your Delphi frontend and still gain modern superpowers.

This post is the whole story in one place: what the five parts claimed, what the case studies proved, and where to start if you're joining now.

The argument the series made

The five parts were designed as a single arc, from *why* through *what* to *how*, and each one stands on its own if you only need that piece.

The five-part arc: business case ’ language ’ UI model ’ framework ’ strategy

Notice that only the first and last boxes are highlighted: the series begins and ends with *business* questions, and the three technical parts in the middle exist to serve them. Here is what each part claimed.

Part 1 — The TypeScript opportunity

Part 1 made the case that expanding into the web stack is a business decision, not a technology fashion statement. Clients increasingly expect applications that run on any device without installation, and the entire destination toolchain — TypeScript, React, Next.js, editors, databases — is available at no licensing cost. The post was explicit about something I still stand behind: **Delphi remains phenomenal** at what it does best, especially Windows desktop and database-heavy enterprise software. The argument was never "abandon Delphi"; it was "add the web to your reach."

One thread made the whole series feel less foreign: Anders Hejlsberg, the original author of Turbo Pascal and chief architect of Delphi, is the lead architect of TypeScript. The language you would be learning was shaped by the same mind that shaped the one you already know.

Part 2 — TypeScript for Delphi developers

Part 2 walked through the language side by side with Object Pascal: strong typing, interfaces, classes, generics, enums — concept for concept. The claim was that a Delphi developer reads TypeScript with a sense of recognition rather than alienation, and that the genuinely new material (union types, structural typing, first-class functions) extends what you know instead of replacing it.

Part 3 — React and the component model

Part 3 translated React into the mental model every Delphi developer already has: components. Drop a `TButton` on a form, set properties, handle events — React is the same compositional idea for the web, with props playing the role of published properties and state playing the role of the data your form manages. The post's purpose was evaluation, not tutorial: understand the model well enough to judge whether your application's UI translates.

Part 4 — Next.js structure and architecture

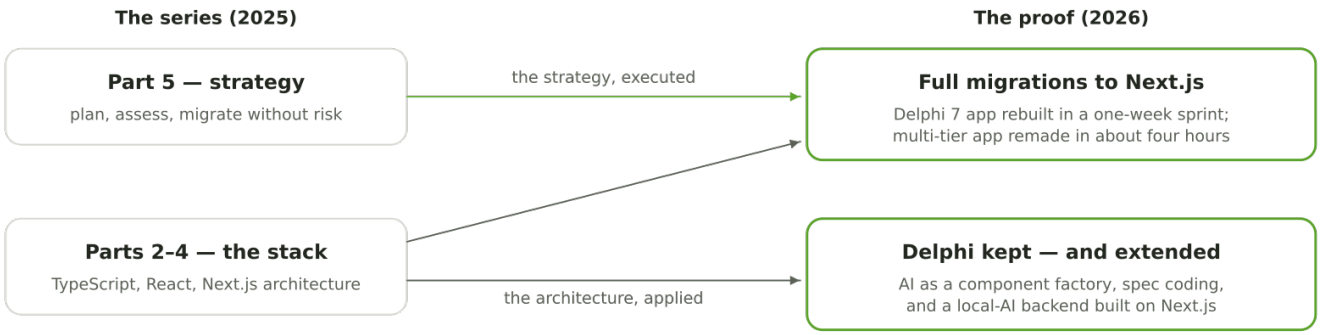
Part 4 positioned Next.js as the equivalent of Delphi's project structure, runtime library, and deployment tooling combined: file-based routing, API routes for backend logic, database access, and the server/client distinction. The key takeaway was architectural — a Next.js application can be the *whole* system, frontend and backend in one deployable project.

Part 5 — Migration strategy and planning

Part 5 was the part written for business owners rather than compilers: four migration strategies (API wrapper, parallel development, module-by-module, new-features-only), how to assess an application honestly, and why the "big rewrite" is a myth to be avoided. Its core message: migration is risk management, and the right strategy depends on your application, your team, and your customers — not on anyone's enthusiasm for a framework.

From argument to evidence

In the months after the series, five published case studies put its central claims to a real test — and they run in *both* directions: some replace a Delphi system with Next.js outright, others deliberately keep Delphi in place and extend it with modern capabilities.



How the 2026 case studies map back to the 2025 series

Read the diagram left to right: the strategy thinking of Part 5 drove the full migrations, while the stack knowledge of Parts 2–4 is the raw material of every project on the right — including the ones that deliberately *keep* Delphi.

The road forward: two full migrations to Next.js

The heaviest claim in the series was Part 5's: that a real, data-carrying Delphi application can move to the web without a multi-year death march. Two published projects back it up.

Adapt or Disappear: How AI Turned a 2-Year Project Into a 1-Week Sprint documents a customer's legacy application built in **Delphi 7** — released in 2002, no Unicode, no HiDPI — running on Firebird 1.5, rebuilt as a full-stack TypeScript platform on Next.js with Prisma and a responsive, mobile-capable UI. Authentication, role-based access control, complex relational data, and a pipeline migrating real production data — roughly 16,000 lines of hand-crafted TypeScript delivered in a one-week sprint, a 15–25× productivity multiplier against the industry-standard estimates the post walks through in detail. Customers had been requesting that modernization for over twenty years.

Four Hours to Migrate a 6-Year-Old Delphi Multi-Tier App pushes the same result further: a production application — a TMS WEB Core frontend, a TMS XData REST backend, and a Firebird database with tens of thousands of rows of real user history — rebuilt on Next.js 16, React 19, Prisma 7, and PostgreSQL. New schema, new authentication, recreated charts, a live data-migration tool, deployment scripts. About four hours, start to finish.

Both posts are deliberately honest about why the headline numbers are *not* the story. The speed was the payoff of months spent learning to drive AI-assisted development as a serious engineering tool: prepared project conventions, controlled access to the legacy sources, and constant expert review of everything the AI produced. The catch-and-correct loop, not the stopwatch, is the real finding.

> **Where the headline numbers don't generalize**
>
> Don't read those case studies as "any Delphi app migrates in an afternoon." Each was migrated by someone who knew both the source system and the target stack deeply, after months of deliberate preparation of AI tooling and conventions. What they prove is the *ceiling*: with the series' concepts internalized and modern AI assistance, the cost curve Part 5 described in 2025 has shifted dramatically downward. Your first migration will not take four hours — but it no longer needs to take years, either.

The road back: keeping Delphi and adding superpowers

The series' other promise — the one in Part 1's very first paragraphs — was that Delphi remains phenomenal and that nobody is asking you to abandon it. Three published projects prove that the same modern toolbox makes your *existing* Delphi work stronger.

AI Won't Replace Delphi Developers. But... starts from a forum thread where a developer had spent days hunting for a bitmap-to-grayscale component — and shows AI producing a clean, dependency-free VCL solution in about two minutes. The lesson generalizes: AI turns "find and evaluate a third-party library" problems into "generate exactly the code I need" problems, entirely inside Delphi.

Spec Coding with Delphi takes that a step further: a native tool that scans a community-curated database of over 12,000 PC games, detects installations via the Windows registry, and backs up save files with optional S3 upload — built in one afternoon with Claude Code, and compiled to a single 4 MB native executable. The methodology the post names *spec coding* — write precise specifications, let the AI implement, review rigorously — is the same discipline behind the big migrations, applied to brand-new Delphi code.

Give Your Delphi App a Brain — Without Sending a Single Byte to the Cloud closes the loop by combining both worlds: a battle-tested Delphi VCL frontend stays exactly where it is, and a small Next.js web service beside it runs a local AI model — in the worked example, turning an uploaded invoice into structured, reviewable expense records. The Delphi side needs little more than a REST call, and the backend ships a web management UI for free. This is Part 4's architecture (Next.js as a self-contained frontend-plus-backend) and Part 5's "new features only" strategy in one artifact.

Together these three answer the fear that started the series: adding this stack to your toolbox doesn't mean abandoning fifteen years of working Delphi code. Sometimes it means extending it with capabilities no amount of `TStringList` parsing will ever deliver.

What 2026 added to the 2025 argument

The series' concept mappings have aged well — TypeScript still reads like a familiar language to a Delphi developer, and the component model still translates. My opinion, and I'll defend it: **TypeScript is the most natural second language a Delphi developer can learn today.** The strongest supporting evidence remains the lineage — Hejlsberg architected both — plus everything Part 2 demonstrated side by side.

> **The other side**

> Scope that opinion honestly: it holds for the *language*, not the *ecosystem*. The npm world moves faster than the Delphi world, build tooling has more moving parts than a single `.dproj`, and the asynchronous programming model is genuinely new territory. The language will feel like home within weeks; fluency with the ecosystem takes months. The case studies above were done *after* that fluency was earned, not before.

What the series could not have fully anticipated is how much AI-assisted development would change the economics. Part 5's timelines assumed hand migration; the one-week and four-hour results show what becomes possible when a developer who understands both stacks supervises an AI that does the mechanical translation. The judgment — schema design, catching wrong or ugly output, knowing what "correct" looks like — is still entirely human, and it is exactly the knowledge the five parts teach.

> **Alternatives**
> Everything on the destination side of this series is open source and free to use: Next.js, React, and TypeScript are all publicly developed on GitHub. Next.js is also not the only credible path — Nuxt (Vue) and SvelteKit solve the same problem excellently, and the concepts in Parts 2–5 transfer to them almost unchanged. And if your team wants the web while staying in Pascal, TMS WEB Core compiles Delphi code to JavaScript and is a genuinely productive option — the four-hour case study migrated *from* it not because it failed, but because that project's goals had outgrown its fit.

Where to start

If you're arriving at this series now, there is a natural reading order.

1. Part 1 — The TypeScript Opportunity for the business case and the honest framing of what Delphi still does best.
2. Part 2 — TypeScript, Part 3 — React, and Part 4 — Next.js in sequence, for the language, the UI model, and the framework — each mapped to Delphi concepts you already own.
3. Part 5 — Migration Strategy before you write a single line of a real project, because strategy failures are more expensive than syntax errors.
4. Then the proof, in both directions: the one-week Delphi 7 migration and the four-hour multi-tier migration for what a full move looks like — and the two-minute grayscale solution, spec coding a native tool in an afternoon, and the local-AI document scanner for what *not* moving, and extending instead, looks like.

The Bottom Line

The 2025 series claimed that a Delphi developer's hard-won concepts transfer to the modern web stack, that the transition is a manageable business decision rather than a leap of faith, and that Delphi and Next.js can coexist in one product strategy. The 2026 case studies turned each of those claims from argument into documented, reproducible practice — some by migrating real production systems, others by proving no migration was needed at all.

> The series taught the map. The case studies drove the road — in both directions.

If you're weighing this decision for your own application, the assessment framework in Part 5 is the place to start — and if you'd like an experienced guide for the trip, let's talk.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —