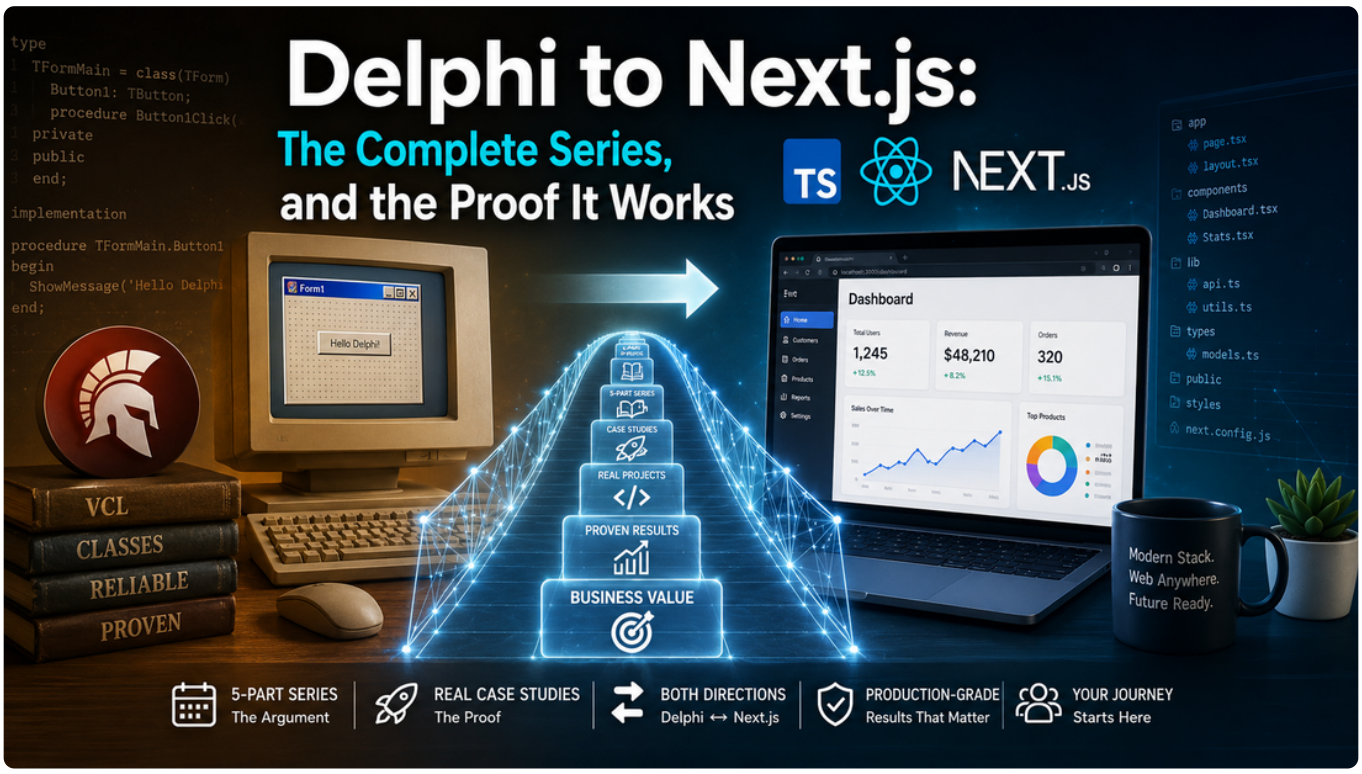
NEWS · JULY 8, 2026

Von Delphi zu Next.js: Die komplette Serie – und der Beweis, dass es funktioniert

By Dr. Holger Flick



Ende 2025 habe ich eine fünfteilige Serie veröffentlicht, in der ich argumentierte, dass ein Delphi-Entwickler — jemand mit Jahrzehnten an Object-Pascal-Muskelgedächtnis — TypeScript, React und Next.js schneller erlernen kann, als er befürchtet, und dass dieser Schritt eine solide unternehmerische Entscheidung ist — kein Verrat an einem geliebten Werkzeug. Die Serie wurde zum meistgelesenen Inhalt, den ich je auf diesem Blog veröffentlicht habe.

Damals war die Serie ein *Argument*. Sie stützte sich auf Konzept-Übersetzungen — Delphi-Interfaces zu TypeScript-Interfaces, VCL-Komponenten zu React-Komponenten, Projektstruktur zu Framework-Struktur — und auf Migrationsstrategien, die ich in meiner Beratungsarbeit eingesetzt hatte. Was sie noch nicht enthalten konnte, war ein öffentlicher, dokumentierter Beweis.

Im Laufe des Jahres 2026 änderte sich das. Zwischen Februar und Juli habe ich eine Reihe von Fallstudien veröffentlicht, die die Behauptungen der Serie einem Realitätstest auf Produktionsniveau unterzogen: eine Delphi-7-Legacy-Anwendung, in einem einwöchigen Sprint als vollständige Next.js-Web-Plattform neu aufgebaut; eine sechs Jahre alte mehrschichtige Delphi-Anwendung, in rund vier Stunden auf einen modernen Next.js-Stack migriert — und, in die entgegengesetzte Richtung, Projekte, die beweisen, dass Sie Ihr Delphi-Frontend *behalten* und trotzdem moderne Superkräfte gewinnen können.

Dieser Beitrag erzählt die ganze Geschichte an einem Ort: was die fünf Teile behaupteten, was die Fallstudien bewiesen haben und wo Sie am besten einsteigen, wenn Sie jetzt dazustoßen.

Das Argument der Serie

Die fünf Teile waren als ein einziger Bogen angelegt, vom *Warum* über das *Was* zum *Wie* — und jeder Teil steht für sich, wenn Sie nur dieses eine Stück brauchen.



Der fünfteilige Bogen: Business Case ’ Sprache ’ UI-Modell ’ Framework ’ Strategie

Beachten Sie, dass nur der erste und der letzte Kasten hervorgehoben sind: Die Serie beginnt und endet mit *unternehmerischen* Fragen, und die drei technischen Teile in der Mitte existieren, um ihnen zu dienen. Hier ist, was jeder Teil behauptete.

Teil 1 — Die TypeScript-Chance

Teil 1 argumentierte, dass die Erweiterung in den Web-Stack eine unternehmerische Entscheidung ist, kein modisches Technologie-Statement. Kunden erwarten zunehmend Anwendungen, die ohne Installation auf jedem Gerät laufen, und die gesamte Ziel-Toolchain — TypeScript, React, Next.js, Editoren, Datenbanken — ist ohne Lizenzkosten verfügbar. Der Beitrag war explizit in einem Punkt, zu dem ich nach wie vor stehe: **Delphi bleibt phänomenal** in dem, was es am besten kann, insbesondere Windows-Desktop- und datenbanklastige Unternehmenssoftware. Das Argument lautete nie „geben Sie Delphi auf“; es lautete „erweitern Sie Ihre Reichweite um das Web“.

Ein roter Faden ließ die ganze Serie weniger fremd wirken: Anders Hejlsberg, der ursprüngliche Autor von Turbo Pascal und Chefarchitekt von Delphi, ist der leitende Architekt von TypeScript. Die Sprache, die Sie lernen würden, wurde vom selben Kopf geformt wie die, die Sie bereits kennen.

Teil 2 — TypeScript für Delphi-Entwickler

Teil 2 ging die Sprache Seite an Seite mit Object Pascal durch: starke Typisierung, Interfaces, Klassen, Generics, Enums — Konzept für Konzept. Die Behauptung war, dass ein Delphi-Entwickler TypeScript mit einem Gefühl des Wiedererkennens statt der Entfremdung liest, und dass das wirklich Neue (Union Types, strukturelle Typisierung, Funktionen als First-Class-Werte) Ihr Wissen erweitert, statt es zu ersetzen.

Teil 3 — React und das Komponentenmodell

Teil 3 übersetzte React in das mentale Modell, das jeder Delphi-Entwickler bereits besitzt: Komponenten.

Einen `TButton` auf ein Formular ziehen, Eigenschaften setzen, Ereignisse behandeln — React ist dieselbe kompositorische Idee für das Web, wobei Props die Rolle der published Properties spielen und State die Rolle der Daten, die Ihr Formular verwaltet. Der Zweck des Beitrags war Bewertung, nicht Tutorial: das Modell gut genug verstehen, um zu beurteilen, ob sich die UI Ihrer Anwendung übersetzen lässt.

Teil 4 — Struktur und Architektur von Next.js

Teil 4 positionierte Next.js als das Äquivalent von Delphis Projektstruktur, Laufzeitbibliothek und Deployment-Werkzeugen in einem: dateibasiertes Routing, API-Routen für Backend-Logik, Datenbankzugriff und

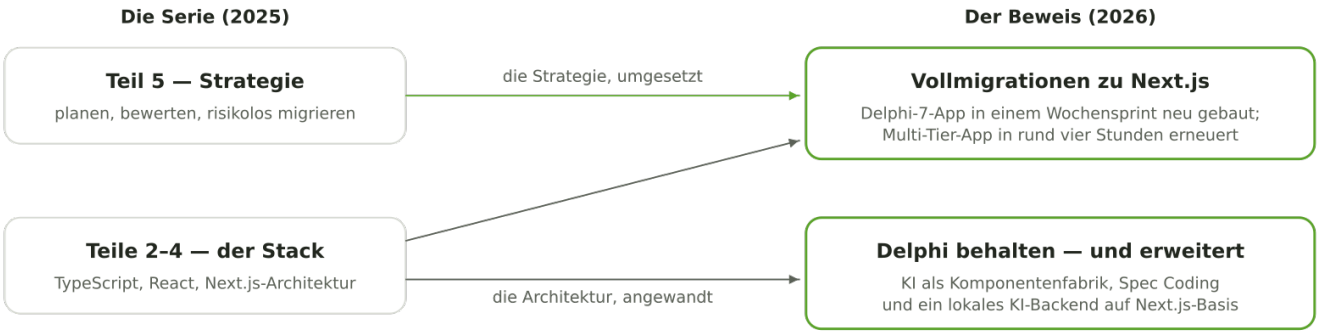
die Unterscheidung zwischen Server und Client. Die zentrale Erkenntnis war architektonischer Natur — eine Next.js-Anwendung kann das *gesamte* System sein, Frontend und Backend in einem einzigen deploybaren Projekt.

Teil 5 — Migrationsstrategie und Planung

Teil 5 war der Teil, der für Unternehmer statt für Compiler geschrieben wurde: vier Migrationsstrategien (API-Wrapper, parallele Entwicklung, Modul für Modul, nur neue Features), wie man eine Anwendung ehrlich bewertet und warum das „große Rewrite“ ein Mythos ist, den es zu vermeiden gilt. Die Kernbotschaft: Migration ist Risikomanagement, und die richtige Strategie hängt von Ihrer Anwendung, Ihrem Team und Ihren Kunden ab — nicht von irgendjemandes Begeisterung für ein Framework.

Vom Argument zum Beweis

In den Monaten nach der Serie stellten fünf veröffentlichte Fallstudien ihre zentralen Behauptungen auf eine echte Probe — und sie verlaufen in *beide* Richtungen: Einige ersetzen ein Delphi-System vollständig durch Next.js, andere behalten Delphi bewusst bei und erweitern es um moderne Fähigkeiten.



Wie die Fallstudien von 2026 auf die Serie von 2025 zurückverweisen

Lesen Sie das Diagramm von links nach rechts: Das strategische Denken aus Teil 5 trieb die Vollmigrationen an, während das Stack-Wissen aus den Teilen 2–4 das Rohmaterial jedes Projekts auf der rechten Seite ist — auch derjenigen, die Delphi bewusst *behalten*.

Der Weg nach vorn: zwei Vollmigrationen zu Next.js

Die gewichtigste Behauptung der Serie stand in Teil 5: dass eine echte, datentragende Delphi-Anwendung ins Web umziehen kann, ohne zum mehrjährigen Todesmarsch zu werden. Zwei veröffentlichte Projekte belegen das.

Adapt or Disappear: How AI Turned a 2-Year Project Into a 1-Week Sprint dokumentiert die Legacy-Anwendung eines Kunden, gebaut in **Delphi 7** — erschienen 2002, kein Unicode, kein HiDPI — auf Firebird 1.5, neu aufgebaut als Full-Stack-TypeScript-Plattform auf Next.js mit Prisma und einer responsiven, mobilfähigen Oberfläche. Authentifizierung, rollenbasierte Zugriffskontrolle, komplexe relationale Daten und eine Pipeline zur Migration echter Produktionsdaten — rund 16.000 Zeilen handwerklich sauberes TypeScript, geliefert in einem einwöchigen Sprint, ein Produktivitätsfaktor von 15–25× gegenüber den branchenüblichen Schätzungen, die der Beitrag im Detail durchgeht. Kunden hatten diese Modernisierung über zwanzig Jahre lang gewünscht.

Four Hours to Migrate a 6-Year-Old Delphi Multi-Tier App treibt dasselbe Ergebnis noch weiter: eine Produktionsanwendung — ein TMS WEB Core-Frontend, ein TMS XData-REST-Backend und eine Firebird-Datenbank mit Zehntausenden Zeilen echter Nutzerhistorie — neu aufgebaut auf Next.js 16, React 19, Prisma 7 und PostgreSQL. Neues Schema, neue Authentifizierung, nachgebaute Diagramme, ein Live-Datenmigrationswerkzeug, Deployment-Skripte. Rund vier Stunden, von Anfang bis Ende.

Beide Beiträge sind bewusst ehrlich darin, warum die Schlagzeilenzahlen *nicht* die eigentliche Geschichte sind. Die Geschwindigkeit war der Lohn von Monaten, in denen KI-gestützte Entwicklung als ernsthaftes Engineering-Werkzeug erlernt wurde: vorbereitete Projektkonventionen, kontrollierter Zugriff auf die Legacy-Quellen und permanente fachkundige Überprüfung von allem, was die KI produzierte. Die Schleife aus Erkennen und Korrigieren, nicht die Stoppuhr, ist der eigentliche Befund.

> **Wo die Schlagzeilenzahlen nicht verallgemeinerbar sind**
> Lesen Sie diese Fallstudien nicht als „jede Delphi-App migriert an einem Nachmittag“. Jede wurde von jemandem migriert, der sowohl das Quellsystem als auch den Ziel-Stack in der Tiefe kannte — nach Monaten gezielter Vorbereitung von KI-Werkzeugen und Konventionen. Was sie beweisen, ist die *Obergrenze*: Mit verinnerlichten Konzepten aus der Serie und moderner KI-Unterstützung hat sich die Kostenkurve, die Teil 5 im Jahr 2025 beschrieb, dramatisch nach unten verschoben. Ihre erste Migration wird nicht vier Stunden dauern — aber sie muss auch nicht mehr Jahre dauern.

Der Weg zurück: Delphi behalten und Superkräfte hinzufügen

Das andere Versprechen der Serie — das aus den allerersten Absätzen von Teil 1 — lautete, dass Delphi phänomenal bleibt und niemand von Ihnen verlangt, es aufzugeben. Drei veröffentlichte Projekte beweisen, dass derselbe moderne Werkzeugkasten Ihre *bestehende* Delphi-Arbeit stärker macht.

AI Won't Replace Delphi Developers. But... beginnt mit einem Forenthread, in dem ein Entwickler tagelang nach einer Komponente für die Umwandlung von Bitmaps in Graustufen gesucht hatte — und zeigt, wie KI in etwa zwei Minuten eine saubere, abhängigkeitsfreie VCL-Lösung liefert. Die Lehre lässt sich verallgemeinern: KI verwandelt Probleme vom Typ „eine Drittanbieter-Bibliothek finden und bewerten“ in Probleme vom Typ „genau den Code generieren, den ich brauche“ — vollständig innerhalb von Delphi.

Spec Coding with Delphi geht noch einen Schritt weiter: ein natives Werkzeug, das eine communitygepflegte Datenbank mit über 12.000 PC-Spielen durchsucht, Installationen über die Windows-Registry erkennt und Spielstände sichert, optional mit S3-Upload — gebaut an einem Nachmittag mit Claude Code und kompiliert zu einer einzigen nativen 4-MB-Executable. Die Methodik, die der Beitrag „spec coding“ nennt — präzise Spezifikationen schreiben, die KI implementieren lassen, rigoros überprüfen — ist dieselbe Disziplin, die hinter den großen Migrationen steht, angewandt auf brandneuen Delphi-Code.

Give Your Delphi App a Brain — Without Sending a Single Byte to the Cloud schließt den Kreis, indem er beide Welten verbindet: Ein bewährtes Delphi-VCL-Frontend bleibt exakt, wo es ist, und ein kleiner Next.js-Webservice daneben betreibt ein lokales KI-Modell — im durchgearbeiteten Beispiel verwandelt er eine hochgeladene Rechnung in strukturierte, überprüfbare Spesendatensätze. Die Delphi-Seite braucht kaum mehr als einen REST-Aufruf, und das Backend liefert eine Web-Verwaltungsoberfläche gratis mit. Das ist die Architektur aus Teil 4 (Next.js als in sich geschlossenes Frontend plus Backend) und die „Nur neue Features“-Strategie aus Teil 5 in einem einzigen Artefakt.

Zusammen beantworten diese drei die Angst, mit der die Serie begann: Diesen Stack in Ihren Werkzeugkasten aufzunehmen bedeutet nicht, fünfzehn Jahre funktionierenden Delphi-Code aufzugeben. Manchmal bedeutet es, ihn um Fähigkeiten zu erweitern, die kein noch so ausgefeiltes `TStringList`-Parsing je liefern wird.

Was 2026 dem Argument von 2025 hinzugefügt hat

Die Konzept-Übersetzungen der Serie sind gut gealtert — TypeScript liest sich für einen Delphi-Entwickler nach wie vor wie eine vertraute Sprache, und das Komponentenmodell lässt sich weiterhin übertragen. Meine Meinung, und ich verteidige sie: **TypeScript ist die natürlichste Zweitsprache, die ein Delphi-Entwickler heute lernen kann.** Der stärkste Beleg bleibt die Abstammung — Hejlsberg hat beide Sprachen architektonisch geprägt — plus alles, was Teil 2 Seite an Seite demonstriert hat.

> **Die andere Seite**
> Grenzen Sie diese Meinung ehrlich ein: Sie gilt für die *Sprache*, nicht für das *Ökosystem*. Die npm-Welt bewegt sich schneller als die Delphi-Welt, das Build-Tooling hat mehr bewegliche Teile als eine einzelne `.dproj`, und das asynchrone Programmiermodell ist wirklich Neuland. Die Sprache wird sich innerhalb von Wochen wie Zuhause anfühlen; die Vertrautheit mit dem Ökosystem braucht Monate. Die obigen Fallstudien entstanden, *nachdem* diese Vertrautheit erarbeitet war — nicht davor.

Was die Serie nicht vollständig vorhersehen konnte, ist, wie stark KI-gestützte Entwicklung die Ökonomie verändern würde. Die Zeitpläne von Teil 5 gingen von manueller Migration aus; die Ergebnisse von einer Woche und vier Stunden zeigen, was möglich wird, wenn ein Entwickler, der beide Stacks versteht, eine KI beaufsichtigt, die die mechanische Übersetzung übernimmt. Das Urteilsvermögen — Schema-Design, falsche oder unschöne Ausgaben erkennen, wissen, wie „korrekt“ aussieht — ist weiterhin vollständig menschlich, und es ist genau das Wissen, das die fünf Teile vermitteln.

> **Alternativen**
> Alles auf der Zielseite dieser Serie ist Open Source und kostenlos nutzbar: Next.js, React und TypeScript werden allesamt öffentlich auf GitHub entwickelt. Next.js ist auch nicht der einzige glaubwürdige Weg — Nuxt (Vue) und SvelteKit lösen dasselbe Problem hervorragend, und die Konzepte aus den Teilen 2–5 lassen sich nahezu unverändert auf sie übertragen. Und wenn Ihr Team das Web will, aber bei Pascal bleiben möchte: TMS WEB Core kompiliert Delphi-Code zu JavaScript und ist eine wirklich produktive Option — die Vier-Stunden-Fallstudie migrierte *von* ihm weg, nicht weil es versagt hätte, sondern weil die Ziele jenes Projekts über seinen Zuschnitt hinausgewachsen waren.

Wo Sie anfangen sollten

Wenn Sie jetzt zu dieser Serie stoßen, gibt es eine natürliche Lesereihenfolge.

1. Teil 1 — Die TypeScript-Chance für den Business Case und die ehrliche Einordnung dessen, was Delphi nach wie vor am besten kann.
2. Teil 2 — TypeScript, Teil 3 — React und Teil 4 — Next.js der Reihe nach, für die Sprache, das UI-Modell und das Framework — jeweils auf Delphi-Konzepte abgebildet, die Sie bereits besitzen.
3. Teil 5 — Migrationsstrategie, bevor Sie eine einzige Zeile eines echten Projekts schreiben, denn Strategiefehler sind teurer als Syntaxfehler.
4. Danach der Beweis, in beide Richtungen: die einwöchige Delphi-7-Migration und die Vier-Stunden-Migration der Multi-Tier-App dafür, wie ein vollständiger Umzug aussieht — und die Zwei-Minuten-Graustufenlösung,

Spec Coding eines nativen Tools an einem Nachmittag und der lokale KI-Dokumentenscanner dafür, wie es aussieht, *nicht* zu migrieren und stattdessen zu erweitern.

Das Fazit

Die Serie von 2025 behauptete, dass die hart erarbeiteten Konzepte eines Delphi-Entwicklers auf den modernen Web-Stack übertragbar sind, dass der Übergang eine beherrschbare unternehmerische Entscheidung ist statt eines Sprungs ins Ungewisse, und dass Delphi und Next.js in einer gemeinsamen Produktstrategie koexistieren können. Die Fallstudien von 2026 haben jede dieser Behauptungen vom Argument in dokumentierte, reproduzierbare Praxis verwandelt — teils durch die Migration echter Produktionssysteme, teils durch den Nachweis, dass gar keine Migration nötig war.

> Die Serie lehrte die Landkarte. Die Fallstudien sind die Straße gefahren — in beide Richtungen.

Wenn Sie diese Entscheidung für Ihre eigene Anwendung abwägen, ist das Bewertungsframework in Teil 5 der richtige Ausgangspunkt — und wenn Sie einen erfahrenen Begleiter für die Reise möchten, sprechen Sie mich an.

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 —