NEWS · OCTOBER 27, 2025

Expanding Your Delphi Expertise: The TypeScript Opportunity (Part 1 of 5)

By Dr. Holger Flick

> **Update (July 2026): the full series recap — with proof**
> This series now has a companion post that condenses all five parts into one page and pairs them with real-world 2026 case studies — including a production Delphi multi-tier app rebuilt on Next.js in about four hours: Delphi to Next.js: The Complete Series, and the Proof It Works.

After 30 years of Delphi development, I never thought I'd write this. But here we are, and I need to share why understanding TypeScript, React, and Next.js might be the most important business decision you make this year.

Why This Series Exists

Let me be clear upfront: **Delphi is still phenomenal**. For Windows desktop applications and database-heavy enterprise software, it remains one of the most productive environments ever created. The VCL is battle-tested, the database components are unmatched, and the compile times are lightning fast.

But the world has changed. Your clients are asking for web applications. They want to access their systems from iPads, Chromebooks, and smartphones. They want applications that don't require installation. They want modern interfaces that look like the apps they use every day. And honestly? Building true cross-platform web applications with Next.js is not just more efficient—it's dramatically more cost-effective than trying to make Delphi work for the web.

The Anders Hejlsberg Connection

Here's something that should immediately catch your attention: **TypeScript was created by Anders Hejlsberg**, the same brilliant engineer who gave us Turbo Pascal and Delphi. When I learned this, everything clicked.

Think about it. The strong typing we love in Delphi? It's there. Interfaces? Check. Generics? Yes. Classes and inheritance? Absolutely. Anders took everything that made Delphi's type system robust and brought it to the JavaScript ecosystem.

This isn't a coincidence. This is a master craftsman applying three decades of language design experience to solve modern problems. If you've ever appreciated Delphi's type safety catching bugs at compile time, you'll feel right at home with TypeScript.

The Economic Reality Nobody Talks About

Let's talk about development costs. I'm a business owner, and so are many of you. Here's something remarkable about the modern web development ecosystem:

The Complete Development Stack is Available at No Cost

- **TypeScript**: Free, open-source language with enterprise-grade tooling
- **React**: Free, backed by Meta with massive community support
- **Next.js**: Free, production-ready framework used by Fortune 500 companies
- **Node.js**: Free runtime with an enormous ecosystem
- **Visual Studio Code**: Free, professional-grade IDE
- **All essential libraries and tools**: Available at no licensing cost

Optional Enhancements

- **AI coding assistants**: Available for those who want them
- **Premium hosting**: Many excellent free tiers available
- **Advanced tooling**: Mostly free, with premium options if desired

The entire development stack adds nothing to your per-developer costs.

This means you can expand your team's capabilities and explore new market opportunities without additional licensing overhead. Your development budget can focus on what matters most: talent, infrastructure, and business growth.

But here's what really matters to your business: your clients don't care what you build with. They care that their application works on their phone, their tablet, and their laptop without installing anything.

The Web Reality: Why Delphi Struggles Here

Delphi can technically do web development. We have WebBroker, WebStencils, and various frameworks. But let's be honest with each other—they feel like we're trying to make a desktop framework do something it wasn't really designed for. It's like using a Formula 1 car to plow a field. Sure, it can pull a plow, but that's not what it was built for.

Next.js was designed from the ground up for modern web applications. Pages load instantly. They work on phones, tablets, and desktops without different code. Search engines can find your content. Users don't install anything—they just click a link.

Your Delphi desktop application might be perfect. But when your biggest client says "we need this to work on any device with a web browser for our sales team without having to install it on each device," what do you tell them?

A Familiar Example: Strong Typing

This is what should give you confidence. In Delphi, you write:

```
type
  TCustomer = class
  private
    FName: string;
    FEmail: string;
  public
    property Name: string read FName write FName;
    property Email: string read FEmail write FEmail;
  end;
```

In TypeScript, this looks remarkably similar:

```
interface Customer {
  name: string;
  email: string;
}
```

If you try to assign the wrong type, TypeScript catches it before you even run the code. Just like Delphi. The same compile-time safety you've relied on for decades.

TypeScript isn't a completely different world. It's a familiar world with a different accent.

Why You're Reading This Series

I'm not going to pretend this series will make you a Next.js expert. It won't. What it will do is show you that the transition is possible, that your Delphi knowledge translates, and that there's a path forward for your business.

Here's the reality: **Most Delphi shops need help with this transition**. You have:

- 10, 15, 20 years of business logic in Delphi code
- Applications that work perfectly and make money
- Clients asking for web and mobile versions
- A team that knows Delphi inside and out but has never touched JavaScript

You can't just stop everything and rewrite from scratch. You need a strategy. You need to understand what's possible. You need to know what to keep in Delphi and what to move to the web.

That's where professional migration consulting comes in. Over this series, I'll show you enough to understand the landscape. When you're ready to actually make the move—when you have a client project that demands web access or when you're planning your next major version—you'll want experienced help.

I help Delphi development shops migrate their applications to modern web platforms. Not by throwing away your investment, but by:

- Analyzing your existing Delphi application architecture
- Identifying what should move to the web and what should stay desktop
- Creating a realistic migration plan that doesn't halt your business
- Training your Delphi developers in TypeScript and Next.js
- Building the initial framework so your team can take over
- Ensuring your data layer transitions smoothly

This isn't about abandoning Delphi. It's about expanding what you can offer your clients. Desktop where it makes sense. Web where they need it. Your code, your business logic, your competitive advantage—preserved and extended.

If you're reading this and thinking "I need to have this conversation about our applications," reach out. Even if you're not ready to start a project, I'm happy to discuss your specific situation and what a migration might look like for your business.

What This Series Will Cover

Over the next four articles, we'll explore the concepts that matter:

Part 2: TypeScript for Delphi Developers

- Why the type system feels so familiar
- How interfaces and classes translate
- What's different (and what's better)
- Understanding the core concepts without diving into implementation

Part 3: Understanding React Components

- Components as building blocks (like VCL controls)
- The concept of props and state
- How event handling works
- Why this approach works for web applications

Part 4: The Next.js Application Model

- How projects are structured
- Routing and navigation concepts
- Where your business logic lives
- How databases integrate

Part 5: Migration Strategy and Planning

- Assessing your Delphi application
- What moves to web, what stays desktop
- Planning a realistic transition
- When to bring in professional help

Each part will compare concepts to what you know in Delphi. The goal isn't to teach you to code Next.js—the goal is to help you understand if this is right for your business and what the migration path looks like.

The Bottom Line for Your Business

You've invested decades in Delphi, and that expertise isn't going away. Your applications work. Your clients are happy. But the market is changing, and you're hearing requests that Delphi wasn't designed to handle.

Here's what you need to know:

Your Delphi background is an advantage. You already understand strong typing, component architecture, event-driven programming, and database operations. These concepts don't change—only the syntax does.

The transition is achievable. Not trivial, but achievable. With the right guidance, your team can learn these technologies. More importantly, you don't have to migrate everything at once.

The business case is compelling. Lower costs, wider reach, modern user experience, and the ability to say "yes" when clients ask for web or mobile access.

You don't have to do it alone. Many Delphi shops have successfully made this transition with professional guidance. The key is having a clear strategy and realistic expectations.

Next Steps

In Part 2, we'll look at TypeScript itself. You'll see why Anders Hejlsberg's design decisions make this feel familiar. We'll explore the type system, look at interfaces and classes, and understand why Delphi developers pick this up faster than developers from dynamic languages.

You don't need to install anything yet. Just read and understand the concepts. If you decide this is the direction for your business, that's when you'll want to start getting hands-on—either on your own or with professional help.

The journey from Delphi to Next.js is about business strategy as much as it is about technology. Let's explore that strategy together.

Want to discuss your specific migration scenario? I work with Delphi development teams to plan and execute transitions to modern web platforms. Whether you're exploring options or ready to start a project, let's talk about what makes sense for your applications and your business. Contact me!

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 —