NEWS · JANUARY 2, 2026

# From 30 Years of Delphi to the Modern Web: My Journey Into 2026

By Dr. Holger Flick

*A personal reflection on three decades of desktop development, and what comes next*

## The Landscape Has Changed

I started with Delphi before most developers today wrote their first line of code. Over 30 years, I watched Borland rise, worked inside the company, and built a career helping others master the platform. I've created hundreds of videos, spoken at conferences worldwide, and earned recognition in the Delphi community.

But I'm not writing this to reminisce. I'm writing because the world has fundamentally shifted, and pretending otherwise helps no one.

The traditional setup—a developer at a desk, building Windows applications that run on office PCs—is no longer the default. People work from phones on trains. They access business systems from tablets in warehouses. Executives approve invoices from airport lounges.

The desktop isn't dead. But it's no longer the center of gravity.

## The Numbers Don't Lie

Consider what's happened in enterprise computing over the past decade:

- **Mobile and tablet usage** for business applications has grown exponentially
- **Cloud-based services** have become the expectation, not the exception
- **Remote and hybrid work** demands applications that work anywhere, on any device
- **Browser-based tools** now handle workloads that once required dedicated desktop software

This isn't a prediction. It's already reality. The question isn't whether your application needs to reach the web—it's when and how.

## Why Delphi Developers Have an Advantage

Here's what many people don't realize: Delphi developers are *exceptionally* well-prepared for modern web development. The skills transfer more directly than you might expect.

### TypeScript Is Pascal's Spiritual Successor

When I started learning TypeScript seriously, I was struck by the similarities. This isn't coincidental—Anders Hejlsberg, who designed Turbo Pascal and Delphi's Object Pascal, also created TypeScript.

Consider these parallels:

**Strong Typing**

```
// TypeScript
interface Customer {
  id: number;
  name: string;
  active: boolean;
}

function processCustomer(customer: Customer): void {
  // Type-safe operations
}
```

```
// Delphi
type
  TCustomer = record
    ID: Integer;
    Name: string;
    Active: Boolean;
  end;

procedure ProcessCustomer(const Customer: TCustomer);
begin
  // Type-safe operations
end;
```

The concepts map directly. If you understand Delphi's type system, you understand TypeScript's. The learning curve isn't a cliff—it's a gentle slope.

## The New RAD Experience

Delphi developers love RAD—Rapid Application Development. Drop components on a form, wire up events, connect a database, ship the application. It's efficient. It's satisfying. It works.

The modern web stack delivers the same experience, differently:

- **Tailwind CSS** gives you a component-based approach to styling—think of it as dropping visual properties onto elements
- **shadcn/ui** provides pre-built, customizable components that you compose into interfaces
- **Prisma ORM** handles database connectivity with the same declarative clarity as Delphi's data components
- **Next.js** provides the framework that ties everything together—routing, API endpoints, server-side rendering

Here's what connecting to a database and querying data looks like with Prisma:

```
// Define your model (like a Delphi table component)
model Product {
  id        Int      @id @default(autoincrement())
  name      String
  price     Decimal
  inStock   Boolean  @default(true)
  createdAt DateTime @default(now())
}

// Query it (like Delphi dataset operations)
const products = await prisma.product.findMany({
  where: { inStock: true },
  orderBy: { name: 'asc' }
});
```

If you've spent years with Delphi datasets, this feels natural. The abstractions are different, but the thinking is the same.

## My Journey: Lessons Learned

I'll be transparent about my path here because I think it's instructive.

My first step toward bridging Delphi and the web was TMS WEB Core with TMS XData. On paper, it seemed like a natural fit—write Pascal, deploy to the web. I invested significant time, created content, and spoke about it publicly. I still value these technologies and will continue working with them, including presenting on TMS FlexCel with TMS WEB Core at the 2026 TMS Days.

But through that experience, I gained something equally important: a deep understanding of how web applications actually work—the layers, the architecture, the fundamentals. TMS WEB Core was my entry point into this world.

What I've come to realize, however, is that for many projects, the broader JavaScript ecosystem offers advantages that are hard to ignore. The npm registry contains over two million packages. The community support, documentation, and tooling for React and TypeScript are extensive. And when you add AI-assisted development to the equation—tools that understand and generate JavaScript and TypeScript fluently—the development velocity is remarkable.

TMS WEB Core remains a valid option, especially for teams deeply invested in Pascal. But for projects requiring maximum flexibility and ecosystem access, the native web stack has become my preferred path forward.

I also need to address a persistent misconception: I never was employed by TMS. I was a contractor. I'm independent, and I'm available for projects. Some people have assumed otherwise, which has affected opportunities. So let me be clear: I work for myself, through FlixEngineering, and I'm always taking on new clients.

## What I'm Offering Now

My position is simple: I'm a Delphi expert who also masters modern web technologies.

This isn't about abandoning anything. Companies with Delphi applications have options:

**Desktop Maintenance and Enhancement** Your Delphi application works. It might need maintenance, new features, or modernization within the Windows ecosystem. I can help with that.

**Web Migration Strategy** You've decided your application needs to reach the web. You need someone who understands your existing codebase AND knows how to build its replacement. That's exactly what I offer.

**Hybrid Approaches** Delphi can serve as a backend, exposing APIs that a modern web frontend consumes.

This preserves your investment in business logic while delivering a contemporary user experience.

**Incremental Modernization** Not every migration needs to be a complete rewrite. Sometimes you can extract components, modernize specific workflows, or build new features on the web while maintaining the core desktop application.

## The AI Factor

I would be doing you a disservice if I didn't mention how AI has transformed development. Tools like GitHub Copilot, Claude, and others understand TypeScript and React deeply. They generate boilerplate, suggest implementations, catch errors, and accelerate development dramatically.

This isn't a gimmick. It's a genuine multiplier. Work that once took days now takes hours. The combination of modern frameworks and AI assistance creates a development velocity that legacy approaches simply cannot match.

I use these tools daily. They're part of how I deliver value to clients.

## Looking Forward

As we enter 2026, I'm focused on several things:

1. **Continuing to serve Delphi clients** who need expertise in their existing systems
2. **Helping organizations migrate** to web technologies with someone who speaks both languages
3. **Creating content** that helps other Delphi developers make this transition
4. **Building applications** using the modern stack I've come to appreciate

The future isn't about choosing sides. It's about building bridges.

If you have a legacy Delphi application and you're wondering what's next—whether that's maintaining what you have, preparing for migration, or executing a full modernization—I'd welcome the conversation.

I'm available, I'm experienced, and I understand both worlds.

Let's build something together.

---

*Holger Flick is a software consultant and CEO of FlixEngineering LLC, based in Estero, Florida. With over 30 years of Delphi experience and expertise in modern web technologies including TypeScript, React, and Next.js, he helps organizations bridge legacy desktop applications with contemporary web platforms. He speaks English and German fluently and is available for consulting engagements worldwide.*

---


- **Contact:** holger(at)flixengineering.com
- **Website:** https://www.holgerscode.com
- **LinkedIn:** https://www.linkedin.com/in/hflick

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 —**