NEWS · AUGUST 21, 2025

Why Web development is difficult for Delphi developers...

By Dr. Holger Flick

I have been fairly quiet over the past few months, focusing on several projects that required my full attention. Last year, I got to know React and I reported about the challenges.

Coming from a Delphi background, I have learned a great deal about web development and have applied my knowledge of **TypeScript** and **React** in real-world scenarios. Some backend functionality is written in Delphi

using TMS XData, allowing me to leverage Delphi's powerful database capabilities while utilizing modern web technologies on the frontend.

When building web applications with **Next.js**, the gap between frontend and backend is narrowing. I can now

develop full-stack applications using a single language, TypeScript, which greatly improves efficiency. The learning curve has been steep, but the rewards are significant.

As Delphi developers, we can quickly grasp TypeScript concepts since they are similar to Object Pascal. The tooling is as mature as the Delphi IDE, and the community is huge, with many resources available to help you get started.

The challenges...

The biggest challenge, as I mentioned in my previous blog post, is transitioning to React. Its component-based architecture differs from what we are used to in Delphi, but once you understand it, it becomes a powerful way to build user interfaces.

However, the main hurdle is not the component-based architecture itself, but the fact that React is a JavaScript framework. JavaScript is very different from Object Pascal—the syntax, coding style, and debugging process all differ. TypeScript helps bridge the gap, but learning a new language and framework remains a challenge.

...but they are not too difficult! A new programming paradigm.

React is a declarative, component-based framework for building user interfaces. Instead of writing imperative code to manipulate the DOM, you write declarative code that describes what the UI should look like. This represents a fundamental shift in how we approach UI development.

Consider a pizza restaurant: you order a pizza, and the restaurant prepares it for you. You don't care how the pizza is made; you just want to eat it. In React, you describe what the pizza should look like, and React handles the preparation. In Delphi, it's different—you must instruct the restaurant on how to make the pizza, what ingredients to use, and how to cook it. You control the entire process.

This is known as the **declarative programming paradigm**. It is a powerful way to build user interfaces, but it

requires a different mindset. For me, understanding this paradigm was the most challenging part of learning React. Once you grasp it, the rest is just learning the syntax and framework. The benefit is that you can build complex user interfaces with less code and reduced complexity. Imperative code is often more complex and harder to maintain, while declarative code is easier to read and understand. Today, I would not be able to build a complex UI without the declarative approach.

UI is difficult

Building user interfaces is challenging and requires great attention to detail. In Delphi, we have the VCL and FireMonkey frameworks, which provide a rich set of components for UI development. In React, however, you must build your own components or use third-party libraries, which can be daunting if you are unfamiliar with the framework.

Fortunately, two 'tools' make building user interfaces in React easier:

- Tailwind CSS: A utility-first CSS framework that enables you to build user interfaces quickly and easily. It offers a set of predefined classes for styling components, making it simple to create responsive, modern UIs without writing extensive custom CSS.
- Shadcn UI: A component library that provides pre-built components for React applications. Built on top of Tailwind CSS, it offers components that are easy to use and customize, reducing the need for custom code.

If you are a Delphi developer looking to get into React, I highly recommend using Tailwind CSS and Shadcn UI. They will make your life much easier and help you build user interfaces efficiently.

Writing backend plus frontend code is tedious

Writing backend code in Delphi and frontend code in React can be tedious. You must switch between two different languages, paradigms, and frameworks, which is challenging if you are not familiar with both. Even if you write your backend in TypeScript, you still need to manage two separate projects.

Next.js simplifies full-stack development by allowing you to write both backend and frontend code in the same project using TypeScript. This approach increases efficiency, enables code sharing between backend and frontend, and reduces duplication. You can call backend code from the frontend using **Server Actions** or **API routes**, making it easy to build full-stack applications with a single language. Everything is encapsulated in one project, simplifying management and deployment.

Conclusion

In summary, React is a powerful framework for building modern, responsive user interfaces. It requires a new way of thinking about application development, but once you understand it, React offers an efficient approach to UI design. The declarative programming paradigm is the most challenging aspect, but it is also the key to React's power. With Next.js, you can build full-stack applications using TypeScript, streamlining development. Delphi, with TMS XData, provides robust database features that complement modern web technologies. If you prefer, you can also write your backend using Next.js.

Are you ready to dip your toes into something new?

If you’re interested in exploring web technologies, learning React, or building your next application for the web, I’m here to help. Whether you want to modernize parts of your existing solution or start a new project, I offer consulting to guide you through the process and help you master these tools. Let’s connect and take your development skills to the next level!

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 —