Migrationsstrategie und Planung: Von Delphi zu Next.js (Teil 5 von 5)

By Dr. Holger Flick

Das große Ganze

Wir haben TypeScript-Grundlagen, React-Komponenten und die Next.js-Architektur behandelt. Jetzt sprechen wir über das, was wirklich zählt: Wie Sie Ihre Delphi-Anwendungen tatsächlich ins Web migrieren, ohne dabei Ihr Unternehmen zu zerstören.

Hier geht es nicht mehr um Code. Es geht um Strategie, Risikomanagement und kluge Geschäftsentscheidungen.

Das Dilemma des Delphi-Entwicklers

Sie befinden sich in einer schwierigen Lage. Sie haben:

Vermögenswerte:

- Anwendungen, die perfekt funktionieren
- Jahre oder Jahrzehnte der Geschäftslogik
- Ein Team, das in Delphi produktiv ist
- Kunden, die (meistens) zufrieden sind
- Umsätze, die von diesen Systemen abhängen

Druck:

- Kunden fragen nach Web-/Mobilzugang
- Schwierigkeiten beim Einstellen von Delphi-Entwicklern
- Steigende Lizenzkosten
- Moderne UX-Erwartungen
- Konkurrenz mit neueren Plattformen

Sorgen:

- "Können wir uns eine komplette Neuentwicklung leisten?"
- "Was ist, wenn die Migration fehlschlägt?"
- "Wird sich unser Team anpassen?"
- "Können wir beide Systeme während der Übergangszeit pflegen?"
- "Was ist, wenn wir Kunden während der Umstellung verlieren?"

Das sind berechtigte Sorgen. Lassen Sie uns sie systematisch angehen.

Der Mythos der großen Neuentwicklung

Seien wir klar: **Sie sollten wahrscheinlich nicht alles von Grund auf neu entwickeln.**

Die komplette Neuentwicklung ist verlockend. Reiner Tisch, moderne Architektur, alle alten Sünden korrigieren. Aber die Geschichte zeigt, dass große Neuentwicklungen oft scheitern:

- **Sie dauern länger als geschätzt** (immer)
- **Anforderungen ändern sich während der Entwicklung**
- **Das Unternehmen kann nicht jahrelang warten**
- **Das Risiko ist in einem riesigen Projekt konzentriert**
- **Sie pflegen zwei Codebasen** (alt und neu)

Es gibt einen besseren Weg: **schrittweise Migration.**

Strategie 1: Der API-Wrapper-Ansatz

Konzept: Behalten Sie Ihre Delphi-Anwendung, aber stellen Sie ihre Funktionalität über Web-APIs zur Verfügung. Erstellen Sie neue Web-Interfaces, die diese APIs aufrufen.

Wie es funktioniert:

1. Ihre Delphi-App läuft weiter und funktioniert
2. Sie fügen API-Endpunkte zu Delphi hinzu (TMS XData, DataSnap, RAD Server oder benutzerdefiniert)
3. Erstellen Sie Next.js-Frontend, das diese APIs aufruft
4. Benutzer greifen auf Web-Interface zu, aber Geschäftslogik bleibt in Delphi

Offensichtlich bin ich etwas voreingenommen, welches Tool für die API-Schicht zu wählen ist, da ich denke, dass TMS XData am besten für Delphi-Entwickler geeignet ist, aber andere Optionen existieren auch.

Vorteile:

- Minimales Risiko für bestehendes System
- Nutzen Sie Jahre bewährter Geschäftslogik
- Web-Zugang ohne vollständige Neuentwicklung
- UI kann schrittweise migriert werden

Nachteile:

- Zwei Systeme zu pflegen, da Delphi-Executable bestehen bleibt
- Kann Performance-Grenzen erreichen
- Beibehaltung der Lizenzkosten

Am besten für: Schnellen Web-Zugang erhalten, während langfristige Migration geplant wird.

Strategie 2: Parallele Entwicklung

Konzept: Erstellen Sie eine neue Web-Version neben der Delphi-Version. Teilen Sie die Datenbank, aber separate Anwendungen.

Wie es funktioniert:

1. Beide Apps greifen auf dieselbe Datenbank zu
2. Delphi-App für Power-User weiterhin
3. Web-App für mobilen/Remote-Zugang
4. Schrittweise Verlagerung von Features zur Web-Version
5. Eventuell Delphi ausmustern (oder für spezifische Benutzer behalten)

Vorteile:

- Keine Störung für aktuelle Benutzer
- Next.js mit echten Projekten lernen
- Zeit nehmen, um es richtig zu machen
- Rückfall, falls Web-Version Probleme hat

Nachteile:

- Doppelter Aufwand für gemeinsame Features
- Datenbank-Schema-Konflikte möglich
- Geschäftslogik-Duplikation
- Verlängerte Übergangszeit

Am besten für: Große Anwendungen, bei denen Benutzer verschiedene Interfaces benötigen (Power-User vs. mobile Benutzer).

Strategie 3: Modul-für-Modul-Migration

Konzept: Identifizieren Sie diskrete Module in Ihrer Anwendung und migrieren Sie sie nacheinander.

Wie es funktioniert:

1. Analysieren Sie die Anwendung in logische Module
2. Wählen Sie das risikoärmste, wertvollste Modul zuerst
3. Migrieren Sie dieses Modul vollständig zu Next.js
4. Benutzer greifen auf diese Funktion über das Web zu
5. Wiederholen Sie mit dem nächsten Modul
6. Ersetzen Sie schrittweise das gesamte System

Beispiel Modul-Reihenfolge:

1. Berichte (nur-lesend, hoher Wert)
2. Dateneingabeformulare (mittlere Komplexität)
3. Dashboard/Analytics
4. Komplexe Workflows
5. Administrative Funktionen

Vorteile:

- Kontinuierliche Wertlieferung

- Lernen und Verbessern mit jedem Modul
- Risiko ist verteilt
- Kann bei Bedarf pausiert werden
- Benutzer gewöhnen sich schrittweise

Nachteile:

- Integrationskomplexität zwischen alt und neu
- Einige Module hängen von anderen ab
- Erfordert sorgfältige Planung
- Verlängerte Zeitlinie

Am besten für: Komplexe Anwendungen mit trennbaren Modulen.

Strategie 4: Nur neue Features

Konzept: Behalten Sie die bestehende Delphi-App wie sie ist. Alle neuen Features gehen in die Next.js-Web-App.

Wie es funktioniert:

1. Delphi-App bleibt im Wartungsmodus
2. Neue Entwicklung findet in Next.js statt
3. Teilen Sie Datenbank oder integrieren Sie über APIs
4. Benutzer verwenden beide Apps nach Bedarf
5. Delphi-App wird langsam zum Legacy-System

Vorteile:

- Minimale Störung
- Team lernt schrittweise
- Sofortiger Nutzen von Web-Features
- Keine erzwungene Migration von funktionierendem Code
- Geringes Risiko

Nachteile:

- Benutzer müssen zwei Systeme verwenden
- Integrations-Overhead
- Delphi-App verschwindet nicht
- Löst möglicherweise Kernprobleme nicht

Am besten für: Wenn bestehende App gut funktioniert, aber Erweiterung benötigt.

Bewertung Ihrer Anwendung

Bevor Sie eine Strategie wählen, verstehen Sie, was Sie haben:

Fragen zu stellen:

Architektur:

- Wie modular ist Ihr Code?
- Wie viel ist UI vs. Geschäftslogik?
- Ist Datenbankzugang zentralisiert?
- Wie viele Abhängigkeiten von VCL?

Geschäftslogik:

- Wie komplex sind Ihre Algorithmen?
- Wie viel ist Berechnung vs. Datenbewegung?
- Sind Validierungen in UI eingebettet?
- Könnte Logik in reine Funktionen extrahiert werden?

Daten:

- Welche Datenbank verwenden Sie?
- Wie normalisiert ist Ihr Schema?
- Gibt es Stored Procedures?
- Wie viel Geschäftslogik ist in der Datenbank?

Team:

- Wie viele Entwickler?
- Welche Erfahrungsstufe haben sie?
- Bereitschaft, neue Tools zu lernen?
- Verfügbare Trainingszeit?

Kunden:

- Wie tolerant sind sie gegenüber Veränderungen?
- Welche Features nutzen sie tatsächlich?
- Welche Features brauchen am meisten Web-Zugang?
- Können Sie beide Systeme während des Übergangs betreiben?

Erstellen einer realistischen Zeitlinie

Hier ist, was Migration typischerweise dauert (für eine mittel-komplexe Anwendung). Aus Erfahrung variieren die Zahlen stark basierend auf Anwendungsgröße, Komplexität und Team-Erfahrung. Verwenden Sie sie, um Erwartungen zu setzen, nicht als Garantien. Schauen Sie auch auf die Verhältnisse statt absolute Zahlen.

Bewertungsphase: 2-4 Wochen

- Code-Analyse
- Architektur-Dokumentation
- Strategieauswahl
- Team-Bewertung

Grundlagenphase: 4-8 Wochen

- Next.js Projekt-Setup
- Datenbank-Migration/Optimierung
- Kern-Komponentenbibliothek
- Entwicklungsmuster etabliert
- Team-Training

Erstes Modul/Feature: 6-12 Wochen

- Implementierung
- Testing
- Benutzerakzeptanz
- Deployment
- Monitoring

Nachfolgende Module: 4-8 Wochen jeweils

- Schneller, da Team lernt
- Muster sind etabliert
- Komponenten sind wiederverwendbar

Für eine 10-Modul-Anwendung:

- Bewertung: 1 Monat
- Grundlagen: 2 Monate
- Module: 5-7 Monate (1-2 gleichzeitig)
- **Gesamt: 8-10 Monate** für komplette Migration

Das sind realistische Zeitlinien mit professioneller Führung. Ohne Führung, fügen Sie 50-100% (und selbst dieser Bereich ist konservativ!) mehr Zeit hinzu.

Die Kosten-Realität

Sprechen wir ehrlich über Geld:

DIY-Migrationskosten:

- Entwicklerzeit (größter Kostenfaktor)
- Training und Lernkurve
- Fehler und Nacharbeit
- Verlängerte Zeitlinie
- Opportunitätskosten anderer Features

Professionelle Migrations-Unterstützung:

- Vorher-Beratungsgebühren
- Aber schnellere Fertigstellung
- Etablierte Muster

- Team-Training inklusive
- Reduziertes Risiko
- Kostet oft weniger gesamt als DIY, wenn Sie Zeit berücksichtigen

Beispiel:

- DIY: 2 Entwickler × 15 Monate × 8k€/Monat = 240k€
- Geführt: 2 Entwickler × 9 Monate × 8k€/Monat + 40k€ Beratung = 184k€

Die Zahlen basieren auf typischen Entwicklerpreisen und Zeitlinien. Ihr Erfolg kann variieren, aber das Prinzip gilt: Der geführte Ansatz kostet oft weniger UND liefert schneller.

Risikomanagement

Jede Migration hat Risiken. So managen Sie sie:

Technische Risiken:

- Mit risikoarmen Modulen beginnen
- Altes System während Übergang beibehalten
- Gründliches Testen vor Umstellung
- Rückfallpläne haben
- Performance genau überwachen

Geschäftsrisiken:

- Kundenakzeptanz früh erhalten
- Fortschritt häufig zeigen
- Wert schrittweise liefern
- Bestehende Workflows nicht brechen
- Training und Support bieten

Team-Risiken:

- In ordentliches Training investieren
- Erfahrene mit lernenden Entwicklern paaren
- Muster und Entscheidungen dokumentieren
- Kleine Erfolge feiern
- Lernkurve akzeptieren

Wann professionelle Hilfe hinzuziehen

Sie könnten professionelle Migrations-Beratung benötigen, wenn:

- **Sie nicht sicher sind, wo Sie anfangen sollen**
- **Zeitlinie ist kritisch** (Kundentermin, Konkurrenzendruck)
- **Team hat keine Web-Entwicklungserfahrung**
- **Anwendung ist komplex oder mission-kritisch**

- **Sie haben es versucht und stecken fest**
- **Sie wollen Muster schnell etabliert**

Was professionelle Hilfe bietet:

Bewertung und Strategie

- Analysieren Sie Ihre Delphi-Codebase
- Empfehlen Sie spezifischen Migrationsansatz
- Erstellen Sie realistischen Projektplan
- Identifizieren Sie Risiken und Milderung

Architektur und Grundlagen

- Next.js Anwendungsstruktur entwerfen
- Datenbank-Migration einrichten
- Komponentenbibliothek erstellen
- Coding-Muster etablieren
- Deployment-Pipeline konfigurieren

Team-Training

- TypeScript Grundlagen
- React-Muster
- Next.js Best Practices
- Hands-on Pair Programming
- Code-Review und Führung

Ausführungsunterstützung

- Erste Module zusammen migrieren
- Komplexe Geschäftslogik-Übersetzung führen
- Performance-Optimierung
- Sicherheitsreview
- Produktions-Deployment

Übergabe und Unabhängigkeit

- Architekturentscheidungen dokumentieren
- Migrations-Playbook erstellen
- Team für unabhängige Fortsetzung trainieren
- Laufenden Support-Kanal bieten

Typisches Engagement: 3-4 Monate intensive Arbeit, dann laufende Unterstützung nach Bedarf.

Meine Migrations-Beratungsdienstleistungen

Ich spezialisiere mich darauf, Delphi-Entwicklungsteams bei der Migration zu modernen Web-Plattformen zu helfen. Das ist keine generische Web-Beratung—das ist speziell für Delphi-Shops, die diesen Übergang machen. Ich biete:

Erstbewertung

- Umfassende Codebase-Analyse
- Migrationsstrategie-Empfehlung
- Realistische Zeit- und Kostenschätzung
- Risikobewertung
- Keine Verpflichtung zum Fortfahren

Vollständige Migrationspartnerschaft

- Komplettes Architekturdesign
- Team-Trainingsprogramm
- Hands-on Entwicklungsführung
- Erste Modulimigration zusammen
- Laufende Unterstützung während vollständiger Migration
- Ziel: Ihr Team setzt nach Grundlagenlegung unabhängig fort

Stunden-Beratung

- Für spezifische Fragen oder Reviews
- Code-Reviews und Architekturberatung
- Fehlerbehebung und Debugging
- Team-Mentoring-Sitzungen

Warum mit mir arbeiten

- 30 Jahre Delphi-Erfahrung (ich verstehe Ihre Codebase)
- Erfolgreich mehrere Delphi-Anwendungen migriert
- Ich spreche sowohl Delphi als auch Next.js fließend
- Bezüglich Sprache spreche ich Englisch und Deutsch, nachdem ich in beiden Ländern viele Jahre gelebt habe
- Fokus auf praktische, geschäftsorientierte Lösungen
- Ziel ist die Unabhängigkeit Ihres Teams, nicht Abhängigkeit

Das Fazit

Die Migration von Delphi zu Next.js ist machbar, aber nicht trivial. Sie erfordert:

- Strategische Planung
- Realistische Erwartungen
- Team-Engagement
- Angemessene Zeit und Ressourcen
- Professionelle Führung (für die meisten Teams)

Die gute Nachricht: Ihre Delphi-Expertise ist wertvoll. Die Konzepte übersetzen sich. Die Lernkurve ist real, aber bewältigbar.

Die bessere Nachricht: Das Endergebnis ist es wert. Niedrigere Kosten, größere Reichweite, moderne UX, einfachere Einstellung und die Fähigkeit, "ja" zu sagen, wenn Kunden nach Web-Zugang fragen.

Nächste Schritte für Sie

Bewerten Sie Ihre Situation:

- Welche Anwendungen brauchen Web-Zugang?
- Wie komplex sind sie?
- Was ist Ihre Zeitlinie?
- Was ist die Kapazität Ihres Teams?

Wählen Sie Ihre Strategie:

- API-Wrapper für schnelle Erfolge?
- Parallele Entwicklung für Sicherheit?
- Modul-für-Modul für gründliche Migration?
- Nur neue Features für schrittweisen Übergang?

Entscheiden Sie über Führung:

- DIY mit dieser Serie als Grundlage?
- Professionelle Bewertung zum Start?
- Vollständige Migrationspartnerschaft?

Handeln Sie:

- Warten Sie nicht auf perfekte Bedingungen
- Klein anfangen und lernen
- Momentum mit frühen Erfolgen aufbauen
- Iterieren und verbessern

Bereit zu starten?

Wenn Sie alle fünf Teile dieser Serie gelesen haben, verstehen Sie:

- Warum TypeScript für Delphi-Entwickler nicht fremd ist
- Wie React-Komponenten konzeptionell funktionieren
- Wie Next.js-Anwendungen strukturiert sind
- Welche Migrationsstrategien verfügbar sind

Jetzt ist es Zeit, dies auf Ihre spezifischen Anwendungen anzuwenden.

Ich würde gerne mit Ihnen über Ihre Situation sprechen. Auch wenn Sie nur Optionen erkunden, kann ein Gespräch helfen zu klären, was involviert ist und was für Ihr Unternehmen Sinn macht.

Kontaktieren Sie mich für:

- Kostenlose 30-minütige Erstberatung
- Anwendungsbewertungsdiskussion
- Migrationsstrategiediskussion
- Team-Training-Informationen

Kontaktinformationen

- **E-Mail:** `holger(at)flixengineering.com`
- **LinkedIn:** Holger Flick

Abschließende Gedanken

Die Delphi-Community war schon immer pragmatisch. Wir lösen Probleme. Wir erstellen Anwendungen, die funktionieren. Wir haben uns schon vorher an Veränderungen angepasst—von DOS zu Windows, von BDE zu FireDAC, von 16-Bit zu 64-Bit.

Dieser Übergang zu Web-Plattformen ist eine weitere Anpassung. Nicht weil Delphi schlecht ist (es ist immer noch ausgezeichnet), sondern weil sich der Markt bewegt hat. Unsere Kunden brauchen Web-Zugang. Unsere Anwendungen müssen mehr Benutzer erreichen. Unsere Unternehmen müssen wettbewerbsfähig bleiben.

TypeScript und Next.js ersetzen Delphi nicht—sie erweitern, was wir bauen können. Und weil Anders Hejlsberg sowohl Delphi als auch TypeScript entworfen hat, ist der Übergang natürlicher, als Sie denken könnten.

Sie können das schaffen. Ihre Delphi-Expertise ist ein Vorteil, keine Belastung. Die Konzepte, die Sie kennen, übersetzen sich direkt. Sie müssen nur die Syntax und Muster einer neuen Plattform lernen.

Sie müssen es nicht alleine machen. Viele Delphi-Shops haben diesen Übergang erfolgreich gemacht. Lernen Sie aus ihrer Erfahrung. Holen Sie sich Führung, wo es Sinn macht. Bauen Sie auf bewährten Mustern auf.

Die Zukunft ist Web, aber die Weisheit ist Delphi. Starke Typisierung, Komponentenarchitektur, Datenbankexpertise—diese Grundlagen ändern sich nicht. Sie fangen nicht von vorne an. Sie übersetzen, was Sie wissen, in einen neuen Dialekt.

Willkommen in der nächsten Phase Ihrer Entwicklerlaufbahn. Lassen Sie uns zusammen etwas Großartiges bauen.

Danke fürs Lesen dieser Serie. Wenn sie Ihnen geholfen hat, den Weg nach vorn zu verstehen, teilen Sie sie bitte mit anderen Delphi-Entwicklern, die ähnliche Entscheidungen treffen. Lassen Sie uns einander dabei helfen, diesen Übergang erfolgreich zu navigieren. Kontaktieren Sie mich!

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 —