NEWS · NOVEMBER 3, 2025

Next.js verstehen: Struktur und Architektur (Teil 4 von 5)

By Dr. Holger Flick

Das große Ganze

Sie haben TypeScript (Teil 2) und React Components (Teil 3) gelernt. Jetzt lassen Sie uns verstehen, wie Next.js alles zu einem vollständigen Application Framework zusammenbringt.

Denken Sie an Next.js als das Äquivalent zu Delphis Projektstruktur, Runtime Library und Deployment Tools - alles kombiniert. Es ist nicht nur eine Library - es ist ein vollständiges Framework für das Erstellen von Web-Anwendungen.

Warum Next.js zusätzlich zu React?

React ist eine UI Library - sie hilft Ihnen beim Erstellen von Komponenten. Aber eine echte Anwendung braucht mehr:

- Routing (Navigation zwischen Seiten)
- Data Fetching (Laden von Informationen aus Datenbanken)
- API Endpoints (Backend Logic)
- Deployment und Optimierung

Next.js stellt all das zur Verfügung. Es ist wie der Unterschied zwischen nur der VCL versus der kompletten Delphi IDE und Runtime.

Projektstruktur: Was gehört wohin

In Delphi haben Sie:

- `.dpr` Datei (Projektdatei)
- `.pas` Units (Code Module)
- `.dfm` Forms (UI Definitionen)
- Data Modules (Business Logic)

In Next.js haben Sie:

```
my-app/
 app/                  # Ihre Pages und Routes
    page.tsx          # Home Page
    about/
       page.tsx      # About Page (/about)
    customers/
        page.tsx      # Customers Page (/customers)
 components/           # Wiederverwendbare UI Components
 lib/                  # Business Logic und Utilities
 public/               # Images, Assets
```

Die wichtigste Erkenntnis: Die Ordnerstruktur im `app/` Verzeichnis **ist** Ihr Routing. Erstellen Sie einen Ordner namens `customers`, fügen Sie eine `page.tsx` Datei hinzu, und Sie haben automatisch eine `/customers` Route. Keine Konfiguration erforderlich.

Routing: Einfacher als Sie denken

In Delphi könnten Sie verschiedene Forms zeigen:

```
procedure TMainForm.ShowCustomers;
begin
  CustomersForm.Show;
end;
```

In Next.js basiert Routing auf Ordnern:

```
app/
 page.tsx             '/              (home)
 about/
    page.tsx         '/about
 products/
    page.tsx         '/products
 customers/
     page.tsx         '/customers
     [id]/
         page.tsx     '/customers/123
```

Die `[id]` Notation bedeutet "dynamisches Segment" - wie das Übergeben eines Parameters. Besuchen Sie `/customers/42`, und Ihre Page erhält `42` als Parameter.

Vergleich: Das ist wie verschiedene Forms in Delphi zu haben, aber das Framework übernimmt die Navigation automatisch.

API Routes: Ihre Backend Logic

Hier wird es interessant. Next.js lässt Sie Ihre Backend API im selben Projekt wie Ihr Frontend erstellen:

```
app/
 api/
     customers/
         route.ts    'Erstellt /api/customers Endpoint
```

Diese Datei kann folgendes handhaben:

- GET Requests (Daten abrufen)
- POST Requests (neue Datensätze erstellen)
- PUT Requests (Datensätze aktualisieren)
- DELETE Requests (Datensätze entfernen)

Es ist wie ein Data Module in Delphi zu haben, aber über HTTP zugänglich. Ihre React Components rufen diese API Endpoints auf, und sie handhaben Datenbankoperationen.

Der Vorteil: Frontend und Backend in einer Codebasis, aber ordentlich getrennt.

Datenbankintegration: Wo Ihre Daten leben

Sie kennen Datenbanken. Sie haben mit FireDAC oder anderen Delphi Database Components gearbeitet. Moderne Web-Apps verwenden ähnliche Konzepte, aber andere Tools.

Beliebte Optionen:

- PostgreSQL (wie InterBase/Firebird)
- MySQL (weit verbreitet unterstützt)
- SQL Server (falls Sie es bereits verwenden)

Und um das zu wiederholen, da ich weiß, dass die Delphi Community es viel nutzt: Firebird ist auch eine Option, mit Libraries wie `node-firebird`, die Interaktion mit Firebird Datenbanken von einer Next.js Anwendung aus ermöglichen.

Die ORM Schicht (wie Delphis Components):

- Prisma: Type-safe Database Access
- Drizzle: SQL-ähnliche Syntax
- Direktes SQL, falls Sie es bevorzugen

Beispiel konzeptionell:

```
// Wie TQuery in Delphi
const customers = await prisma.customer.findMany({
  where: { active: true },
  orderBy: { name: 'asc' }
});
```

Ihr SQL Wissen überträgt sich direkt. Die Tools sind anders, aber `SELECT`, `INSERT`, `UPDATE`, `DELETE` funktionieren auf die gleiche Weise.

Datenfluss: Wie alles zusammenhängt

In einer Delphi-Anwendung:

1. Form lädt ’`OnCreate` wird ausgelöst
2. Dataset abfragen ’`FDQuery.Open`
3. Controls befüllen ’ DataSource bindet an Controls
4. User editiert ’ Controls aktualisieren sich
5. User speichert ’`FDQuery.Post`

In einer Next.js-Anwendung:

1. Page lädt ’ fetch von API
2. API fragt Datenbank ab ’ Prisma/SQL
3. JSON zurückgeben ’ API sendet Daten
4. Component rendert ’ React zeigt Daten an
5. User editiert ’ State aktualisiert sich
6. User speichert ’ POST an API

Unterschiedliche Mechanismen, derselbe Ablauf.

Server vs Client: Ein wichtiger Unterschied

Next.js hat zwei Arten von Components:

Server Components (Standard):

- Laufen auf dem Server
- Können direkt auf die Datenbank zugreifen
- Kein JavaScript wird an den Browser gesendet
- Wie serverseitiger Code in Web-Anwendungen

Client Components:

- Laufen im Browser
- Handhaben User-Interaktionen
- Wie Ihr Form-Code in Delphi

Dieser Unterschied existiert in Delphi nicht, weil alles auf dem Client läuft. In Web-Apps wählen Sie aus, wo Code für Performance und Sicherheit läuft.

Was ist mit Echtzeit-Updates?

Ihre Delphi-Apps könnten LiveBindings oder Dataset Events für Echtzeit-Updates verwenden. Web-Apps handhaben das anders:

- **Polling**: Periodisch nach Updates prüfen
- **WebSockets**: Echtzeit bidirektionale Kommunikation
- **Server-Sent Events**: Server sendet Updates

Die Patterns sind anders, aber das Ziel ist dasselbe: das UI mit den Daten synchron halten.

Environment und Konfiguration

Delphi verwendet:

- Project Options
- Konfigurationsdateien
- Registry Settings

Next.js verwendet:

- `.env` Dateien für Secrets (Datenbankpasswörter, API Keys)
- `next.config.js` für Build Settings
- Environment Variables für Deployment

Es ist einfacher und plattformübergreifend standardisierter.

Die Lernkurve verstehen

Hier ist, was sich leicht übertragen lässt:

- Datenbankkonzepte und SQL
- Business Logic und Berechnungen
- Validierungsregeln
- Datenstrukturen

Hier ist, was anders ist:

- Deklaratives UI anstatt imperatives
- Asynchrone Operationen (async/await)
- JSON anstatt Datasets
- Web Architektur Entwurfsmuster

Die gute Nachricht: die unterschiedlichen Teile sind erlernbar. Sie sind nicht schwerer - nur anders.

Ihre Migrationsbedürfnisse bewerten

Beim Betrachten Ihrer Delphi-Anwendungen:

Einfach zu migrieren:

- Standard CRUD Operationen
- Report-Generierung
- Datenvalidierung
- Business-Berechnungen
- User Authentication

Erfordert Überdenken:

- Echtzeit-Datenaktualisierungen
- Komplexe Multi-Window Workflows
- Intensive clientseitige Verarbeitung
- Custom Database Components

Könnte in Delphi bleiben:

- Hardware-Integration
- Windows-spezifische Features
- Legacy Database Compatibility
- Bestehende Integrationen, die funktionieren

Nicht alles muss wechseln. Manchmal ist die richtige Antwort: **Web für neue Sachen, Delphi für das, was funktioniert**.

Die Migrationsstrategie-Frage

Hier bleiben die meisten Delphi-Shops stecken. Sie haben:

- Funktionierende Anwendungen, die Geld verdienen

- Kunden, die nach Web/Mobile-Zugang fragen
- Ein Team, das Delphi gut kennt
- Begrenzte Zeit und Budget für Rewrites

Die Antwort ist nicht "alles neu schreiben". Es ist "strategisch migrieren".

Häufige Ansätze:

1. **Nur neue Features**: Delphi-App behalten, neue Features als Web entwickeln
2. **Modul für Modul**: Ein Subsystem nach dem anderen verschieben
3. **API Wrapper**: Delphi Logic durch Web APIs exponieren
4. **Parallele Entwicklung**: Neue Web-App neben Delphi-App
5. **Schrittweise Migration**: Delphi-Screens über Zeit auslaufen lassen

Jeder hat Kompromisse. Die richtige Wahl hängt von Ihrer spezifischen Situation ab.

Wann Sie professionelle Beratung brauchen

Die Konzepte zu verstehen ist Schritt eins. Eine Migration zu planen und auszuführen ist Schritt zwei. Da zählt Erfahrung.

Ich helfe Delphi-Teams bei:

- Architektur und Planung
- Auswahl der richtigen Migrationsstrategie
- Datenbankmigration und -optimierung
- Team Training und Mentoring
- Code Migration und Refactoring
- Etablierung von Entwicklungsmustern

Das Ziel ist nicht, Ihre Entwicklung zu übernehmen - es ist, Sie zum Erfolg zu führen. Die meisten Engagements sind 4-12 Wochen intensive Arbeit, um das Fundament zu legen, dann macht Ihr Team weiter. Mein Ziel ist es, Ihre internen Fähigkeiten aufzubauen, nicht Abhängigkeit zu schaffen. Andererseits bin ich immer für Follow-up Consulting verfügbar, während Ihr Team mit dem neuen Stack vertrauter wird.

Was als nächstes kommt

In Teil 5 (der finalen Ausgabe) werden wir spezifische Migrationsstrategien und Planung besprechen. Wir werden reden über:

- Wie Sie Ihre Delphi-Anwendung für Migration bewerten
- Was zuerst migriert werden soll (und was in Ruhe gelassen werden soll)
- Einen realistischen Zeitplan erstellen
- Risiko während des Übergangs managen
- Wann professionelle Hilfe hinzugezogen werden sollte

Hier fügen wir alles zusammen und geben Ihnen ein Framework für Entscheidungen über Ihre Anwendungen.

Möchten Sie Ihr spezifisches Next.js Migrationsszenario besprechen? Ob Sie eine einzelne Anwendung oder eine Suite von Delphi-Systemen haben, ich kann Ihnen helfen zu bewerten, wie eine Migration für Ihre Situation aussehen würde. 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 —