NEWS · NOVEMBER 3, 2025

Understanding Next.js: Structure and Architecture (Part 4 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.

The Big Picture

You've learned TypeScript (Part 2) and React components (Part 3). Now let's understand how Next.js brings it all together into a complete application framework.

Think of Next.js as the equivalent of Delphi's project structure, runtime library, and deployment tools all combined. It's not just a library—it's a complete framework for building web applications.

Why Next.js on Top of React?

React is a UI library—it helps you build components. But a real application needs more:

- Routing (navigating between pages)
- Data fetching (loading information from databases)
- API endpoints (backend logic)
- Deployment and optimization

Next.js provides all of this. It's like the difference between having just the VCL versus having the complete Delphi IDE and runtime.

Project Structure: What Goes Where

In Delphi, you have:

- `.dpr` file (project file)
- `.pas` units (code modules)
- `.dfm` forms (UI definitions)
- Data modules (business logic)

In Next.js, you have:

```
my-app/
 app/                  # Your pages and routes
    page.tsx          # Home page
    about/
       page.tsx      # About page (/about)
    customers/
        page.tsx      # Customers page (/customers)
 components/           # Reusable UI components
 lib/                  # Business logic and utilities
 public/               # Images, assets
```

The key insight: The folder structure in the `app/` directory **is** your routing. Create a folder called `customers`, add a `page.tsx` file, and you automatically have a `/customers` route. No configuration needed.

Routing: Simpler Than You Think

In Delphi, you might show different forms:

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

In Next.js, routing is based on folders:

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

The `[id]` notation means "dynamic segment"—like passing a parameter. Visit `/customers/42`, and your page gets `42` as a parameter.

Comparison: This is like having different forms in Delphi, but the framework handles navigation automatically.

API Routes: Your Backend Logic

Here's where it gets interesting. Next.js lets you build your backend API in the same project as your frontend:

```
app/
 api/
     customers/
         route.ts    'Creates /api/customers endpoint
```

This file can handle:

- GET requests (fetch data)
- POST requests (create new records)

- PUT requests (update records)
- DELETE requests (remove records)

It's like having a data module in Delphi, but accessible via HTTP. Your React components call these API endpoints, and they handle database operations.

The advantage: Frontend and backend in one codebase, but properly separated.

Database Integration: Where Your Data Lives

You know databases. You've worked with FireDAC or other Delphi database components. Modern web apps use similar concepts but different tools.

Popular choices:

- PostgreSQL (like InterBase/Firebird)
- MySQL (widely supported)
- SQL Server (if you're already using it)

And to repeat this, as I know the Delphi community uses it a lot: Firebird is also an option, with libraries like `node-firebird` allowing interaction with Firebird databases from a Next.js application.

The ORM layer (like Delphi's components):

- Prisma: Type-safe database access
- Drizzle: SQL-like syntax
- Direct SQL if you prefer

Example conceptually:

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

Your SQL knowledge transfers directly. The tools are different, but `SELECT`, `INSERT`, `UPDATE`, `DELETE` work the same way.

Data Flow: How It All Connects

In a Delphi application:

1. Form loads ’`OnCreate` fires
2. Query dataset ’`FDQuery.Open`
3. Populate controls ’ DataSource binds to controls
4. User edits ’ Controls update
5. User saves ’`FDQuery.Post`

In a Next.js application:

1. Page loads ’ fetch from API
2. API queries database ’ Prisma/SQL

3. Return JSON ’ API sends data
4. Component renders ’ React displays data
5. User edits ’ State updates
6. User saves ’ POST to API

Different mechanics, same flow.

Server vs Client: An Important Distinction

Next.js has two types of components:

Server Components (default):

- Run on the server
- Can access database directly
- No JavaScript sent to browser
- Like server-side code in web applications

Client Components:

- Run in the browser
- Handle user interactions
- Like your form code in Delphi

This distinction doesn't exist in Delphi because everything runs on the client. In web apps, you choose where code runs for performance and security.

What About Real-Time Updates?

Your Delphi apps might use LiveBindings or dataset events for real-time updates. Web apps handle this differently:

- **Polling**: Check for updates periodically
- **WebSockets**: Real-time two-way communication
- **Server-Sent Events**: Server pushes updates

The patterns are different, but the goal is the same: keep the UI in sync with data.

Environment and Configuration

Delphi uses:

- Project options
- Configuration files
- Registry settings

Next.js uses:

- `.env` files for secrets (database passwords, API keys)
- `next.config.js` for build settings

- Environment variables for deployment

It's simpler and more standard across platforms.

Understanding the Learning Curve

Here's what translates easily:

- Database concepts and SQL
- Business logic and calculations
- Validation rules
- Data structures

Here's what's different:

- Declarative UI instead of imperative
- Asynchronous operations (async/await)
- JSON instead of datasets
- Web architecture patterns

The good news: the different parts are learnable. They're not harder—just different.

Evaluating Your Migration Needs

When looking at your Delphi applications:

Easy to migrate:

- Standard CRUD operations
- Report generation
- Data validation
- Business calculations
- User authentication

Requires rethinking:

- Real-time data updates
- Complex multi-window workflows
- Heavy client-side processing
- Custom database components

Might stay in Delphi:

- Hardware integration
- Windows-specific features
- Legacy database compatibility
- Existing integrations that work

Not everything needs to move. Sometimes the right answer is: **web for new stuff, Delphi for what works**.

The Migration Strategy Question

This is where most Delphi shops get stuck. You have:

- Working applications that make money
- Clients asking for web/mobile access
- A team that knows Delphi well
- Limited time and budget for rewrites

The answer isn't "rewrite everything." It's "migrate strategically."

Common approaches:

1. **New features only**: Keep Delphi app, build new features as web
2. **Module by module**: Move one subsystem at a time
3. **API wrapper**: Expose Delphi logic through web APIs
4. **Parallel development**: New web app alongside Delphi app
5. **Gradual migration**: Phase out Delphi screens over time

Each has tradeoffs. The right choice depends on your specific situation.

When You Need Professional Guidance

Understanding the concepts is step one. Planning and executing a migration is step two. That's where experience matters.

I help Delphi teams with:

- Architecture assessment and planning
- Choosing the right migration strategy
- Database migration and optimization
- Team training and mentorship
- Code migration and refactoring
- Establishing development patterns

The goal isn't to take over your development—it's to set you up for success. Most engagements are 4-12 weeks of intensive work to establish the foundation, then your team continues. My aim is to build your internal capabilities, not create dependency. On the other hand, I’m always available for follow-up consulting as your team grows more comfortable with the new stack.

What's Next

In Part 5 (the final installment), we'll discuss specific migration strategies and planning. We'll talk about:

- How to assess your Delphi application for migration
- What to migrate first (and what to leave alone)
- Creating a realistic timeline
- Managing risk during transition
- When to bring in professional help

This is where we put it all together and give you a framework for making decisions about your applications.

Want to discuss your specific Next.js migration scenario? Whether you have a single application or a suite of Delphi systems, I can help you evaluate what migration would look like for your situation. 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 —