NEWS · NOVEMBER 10, 2025

# Migration Strategy and Planning: From Delphi to Next.js (Part 5 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 Complete Picture

We've covered TypeScript fundamentals, React components, and Next.js architecture. Now let's talk about what really matters: how to actually migrate your Delphi applications to the web without destroying your business in the process.

This isn't about code anymore. This is about strategy, risk management, and making smart business decisions.

## The Delphi Developer's Dilemma

You're in a tough spot. You have:

**Assets:**

- Applications that work perfectly
- Years or decades of business logic
- A team that's productive in Delphi
- Clients who are (mostly) happy
- Revenue that depends on these systems

**Pressures:**

- Clients asking for web/mobile access
- Difficulty hiring Delphi developers
- Rising licensing costs
- Modern UX expectations
- Competition using newer platforms

**Concerns:**

- "Can we afford a complete rewrite?"
- "What if the migration fails?"
- "Will our team adapt?"
- "Can we maintain both systems during transition?"

- "What if we lose clients during the change?"

These are legitimate concerns. Let's address them systematically.

## The Myth of the Big Rewrite

Let's be clear: **you probably shouldn't rewrite everything from scratch.**

The big bang rewrite is tempting. Clean slate, modern architecture, fix all the old sins. But history shows that big rewrites often fail:

- **They take longer than estimated** (always)
- **Requirements change during development**
- **Business can't wait for years**
- **Risk is concentrated in one massive project**
- **You're maintaining two codebases** (old and new)

There's a better way: **incremental migration.**

## Strategy 1: The API Wrapper Approach

**Concept:** Keep your Delphi application, but expose its functionality through web APIs. Build new web interfaces that call these APIs.

**How it works:**

1. Your Delphi app continues running and working
2. You add API endpoints to Delphi (TMS XData, DataSnap, RAD Server, or custom)
3. Build Next.js frontend that calls these APIs
4. Users access web interface, but business logic stays in Delphi

Obviously, I am a bit biased here which tool to pick for the API layer as I just think TMS XData is the best fit for Delphi developers, but other options exist as well.

**Advantages:**

- Minimal risk to existing system
- Leverage years of proven business logic
- Web access without full rewrite
- Can migrate UI piece by piece

**Disadvantages:**

- Two systems to maintain as Delphi executable stays
- May hit performance limits
- Keeps licensing costs

**Best for:** Getting web access quickly while planning longer-term migration.

## Strategy 2: Parallel Development

**Concept:** Build new web version alongside Delphi version. Share database, but separate applications.

**How it works:**

1. Both apps access same database
2. Delphi app continues for power users
3. Web app for mobile/remote access
4. Gradually shift features to web version
5. Eventually phase out Delphi (or keep for specific users)

**Advantages:**

- No disruption to current users
- Learn Next.js with real projects
- Can take time to get it right
- Fallback if web version has issues

**Disadvantages:**

- Duplicate effort for shared features
- Database schema conflicts possible
- Business logic duplication
- Extended transition period

**Best for:** Large applications where users need different interfaces (power users vs mobile users).

## Strategy 3: Module-by-Module Migration

**Concept:** Identify discrete modules in your application and migrate them one at a time.

**How it works:**

1. Analyze application into logical modules
2. Pick lowest-risk, highest-value module first
3. Migrate that module to Next.js completely
4. Users access that feature via web
5. Repeat with next module
6. Gradually replace entire system

**Example module order:**

1. Reporting (read-only, high value)
2. Data entry forms (medium complexity)
3. Dashboard/analytics
4. Complex workflows
5. Administrative functions

**Advantages:**

- Continuous delivery of value
- Learn and improve with each module
- Risk is distributed
- Can pause if needed
- Users adapt gradually

**Disadvantages:**

- Integration complexity between old and new
- Some modules depend on others
- Requires careful planning
- Extended timeline

**Best for:** Complex applications with separable modules.

## Strategy 4: New Features Only

**Concept:** Keep existing Delphi app as-is. All new features go into Next.js web app.

**How it works:**

1. Delphi app stays in maintenance mode
2. New development happens in Next.js
3. Share database or integrate via APIs
4. Users use both apps as needed
5. Delphi app slowly becomes legacy

**Advantages:**

- Minimal disruption
- Team learns gradually
- Immediate benefit from web features
- No forced migration of working code
- Low risk

**Disadvantages:**

- Users must use two systems
- Integration overhead
- Delphi app not going away

- May not solve core problems

**Best for:** When existing app works well but needs expansion.

## Assessing Your Application

Before choosing a strategy, understand what you have:

### Questions to Ask:

**Architecture:**

- How modular is your code?
- How much is UI vs business logic?
- Is database access centralized?
- How many dependencies on VCL?

**Business Logic:**

- How complex are your algorithms?
- How much is calculation vs data movement?
- Are validations embedded in UI?
- Could logic be extracted to pure functions?

**Data:**

- What database are you using?
- How normalized is your schema?
- Are there stored procedures?
- How much business logic is in the database?

**Team:**

- How many developers?
- What's their experience level?
- Willingness to learn new tools?
- Available training time?

**Clients:**

- How tolerant are they of change?
- What features do they actually use?
- Which features need web access most?
- Can you run both systems during transition?

## Creating a Realistic Timeline

Here's what migration typically takes (for a medium-complexity application). From experience, the numbers vary widely based on application size, complexity, and team experience. Use them to set expectations, not as guarantees. Also, look at the proportions rather than absolute numbers.

**Assessment Phase: 2-4 weeks**

- Code analysis
- Architecture documentation
- Strategy selection
- Team evaluation

**Foundation Phase: 4-8 weeks**

- Next.js project setup
- Database migration/optimization
- Core components library
- Development patterns established
- Team training

**First Module/Feature: 6-12 weeks**

- Implementation
- Testing
- User acceptance
- Deployment
- Monitoring

**Subsequent Modules: 4-8 weeks each**

- Faster as team learns
- Patterns are established
- Components are reusable

**For a 10-module application:**

- Assessment: 1 month
- Foundation: 2 months
- Modules: 5-7 months (doing 1-2 at a time)
- **Total: 8-10 months** for complete migration

These are realistic timelines with professional guidance. Without guidance, add 50-100% (and even this range is conservative!) more time.

## The Cost Reality

Let's talk money honestly:

**DIY Migration Costs:**

- Developer time (biggest cost)

- Training and learning curve
- Mistakes and rework
- Extended timeline
- Opportunity cost of other features

**Professional Migration Assistance:**

- Upfront consulting fees
- But faster completion
- Established patterns
- Team training included
- Reduced risk
- Often costs less total than DIY when you factor in time

**Example:**

- DIY: 2 developers × 15 months × $8k/month = $240k
- Guided: 2 developers × 9 months × $8k/month + $40k consulting = $184k

The numbers are based on typical developer rates and timelines. Your mileage will vary, but the principle holds: The guided approach often costs less AND delivers faster.

## Risk Management

Every migration has risks. Here's how to manage them:

**Technical Risks:**

- Start with low-risk modules
- Maintain old system during transition
- Thorough testing before switchover
- Have rollback plans
- Monitor performance closely

**Business Risks:**

- Get client buy-in early
- Show progress frequently
- Deliver value incrementally
- Don't break existing workflows
- Provide training and support

**Team Risks:**

- Invest in proper training
- Pair experienced with learning devs
- Document patterns and decisions
- Celebrate small wins
- Accept learning curve exists

# When to Bring in Professional Help

You might need professional migration consulting if:

- **You're not sure where to start**
- **Timeline is critical** (client deadline, competitive pressure)
- **Team has no web development experience**
- **Application is complex or mission-critical**
- **You've tried and gotten stuck**
- **You want patterns established quickly**

What professional help provides:

## Assessment and Strategy

- Analyze your Delphi codebase
- Recommend specific migration approach
- Create realistic project plan
- Identify risks and mitigation

## Architecture and Foundation

- Design Next.js application structure
- Set up database migration
- Create component library
- Establish coding patterns
- Configure deployment pipeline

## Team Training

- TypeScript fundamentals
- React patterns
- Next.js best practices
- Hands-on pair programming
- Code review and guidance

## Execution Support

- Migrate first modules together
- Guide complex business logic translation
- Performance optimization
- Security review
- Production deployment

## Handoff and Independence

- Document architecture decisions
- Create migration playbook
- Train team to continue independently
- Provide ongoing support channel

**Typical engagement:** 3-4 months intensive work, then ongoing support as needed.

# My Migration Consulting Services

I specialize in helping Delphi development teams migrate to modern web platforms. This isn't generic web consulting—this is specifically for Delphi shops making this transition. I offer:

## Initial Assessment

- Comprehensive codebase analysis
- Migration strategy recommendation
- Realistic timeline and cost estimate
- Risk assessment
- No obligation to continue

## Full Migration Partnership

- Complete architecture design
- Team training program
- Hands-on development guidance
- First module migration together
- Ongoing support during full migration
- Goal: Your team continues independently after foundation is set

## Hourly Consulting

- For specific questions or reviews
- Code reviews and architecture advice
- Troubleshooting and debugging
- Team mentoring sessions

## Why Work With Me

- 30 years Delphi experience (I understand your codebase)
- Successfully migrated multiple Delphi applications
- I speak both Delphi and Next.js fluently
- With regards to language, I speak English and German having lived in both countries for many years
- Focus on practical, business-driven solutions
- Goal is your team's independence, not dependency

# The Bottom Line

Migrating from Delphi to Next.js is achievable, but it's not trivial. It requires:

- Strategic planning
- Realistic expectations
- Team commitment
- Adequate time and resources
- Professional guidance (for most teams)

**The good news:** Your Delphi expertise is valuable. The concepts translate. The learning curve is real but manageable.

**The better news:** The end result is worth it. Lower costs, wider reach, modern UX, easier hiring, and the ability to say "yes" when clients ask for web access.

## Next Steps for You

### Assess your situation:

- Which applications need web access?
- How complex are they?
- What's your timeline?
- What's your team's capacity?

### Choose your strategy:

- API wrapper for quick wins?
- Parallel development for safety?
- Module-by-module for thorough migration?
- New features only for gradual transition?

### Decide on guidance:

- DIY with this series as foundation?
- Professional assessment to start?
- Full migration partnership?

### Take action:

- Don't wait for perfect conditions
- Start small and learn
- Build momentum with early wins
- Iterate and improve

## Ready to Start?

If you've read all five parts of this series, you understand:

- Why TypeScript isn't foreign to Delphi developers
- How React components work conceptually
- How Next.js applications are structured

- What migration strategies are available

**Now it's time to apply this to your specific applications.**

I'd love to talk with you about your situation. Even if you're just exploring options, a conversation can help clarify what's involved and what makes sense for your business.

**Contact me for:**

- Free 30-minute initial consultation
- Application assessment discussion
- Migration strategy discussion
- Team training information

### Contact Information

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

## Final Thoughts

The Delphi community has always been pragmatic. We solve problems. We build applications that work. We've adapted to changes before—from DOS to Windows, from BDE to FireDAC, from 16-bit to 64-bit.

This transition to web platforms is another adaptation. Not because Delphi is bad (it's still excellent), but because the market has moved. Our clients need web access. Our applications need to reach more users. Our businesses need to stay competitive.

TypeScript and Next.js aren't replacing Delphi—they're extending what we can build. And because Anders Hejlsberg designed both Delphi and TypeScript, the transition is more natural than you might think.

**You can do this.** Your Delphi expertise is an advantage, not a liability. The concepts you know translate directly.

You just need to learn the syntax and patterns of a new platform.

**You don't have to do it alone.** Many Delphi shops have made this transition successfully. Learn from their

experience. Get guidance where it makes sense. Build on proven patterns.

**The future is web, but the wisdom is Delphi.** Strong typing, component architecture, database exper-

tise—these fundamentals don't change. You're not starting over. You're translating what you know into a new dialect.

Welcome to the next phase of your development career. Let's build something great together.

---

**Thank you for reading this series.** If it helped you understand the path forward, please share it with other

Delphi developers facing similar decisions. Let's help each other navigate this transition successfully. 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 —**