# **Technical Implementation Guide**

## **Third-Party Tool Replacement for Legacy Concierge**

**Version**: 1.0 **Date**: August 18, 2025 **Team**: Development Team

## **Quick Start Implementation**

### **Immediate Actions (Week 1)**

1. **Security Risk Mitigation**

# Remove misplaced WordPress core files

rm /Users/darcher/dev/legacyConcierge/wp-content/about.php

rm /Users/darcher/dev/legacyConcierge/wp-content/includes/class-wp-site-health.php

# Run security scan

docker exec legacy-concierge\_app wp core verify-checksums --allow-root

1. **Development Environment Setup**

# Create new project structure

mkdir -p legacy-concierge-custom/{frontend,backend,infrastructure}

cd legacy-concierge-custom

# Initialize Next.js frontend

npx create-next-app@latest frontend --typescript --tailwind --app

# Initialize Node.js backend

mkdir backend && cd backend

npm init -y

npm install express typescript prisma @prisma/client

### **Phase 1: Custom Foundation (Weeks 2-12)**

#### **1. Database Schema Design**

-- Core content tables

CREATE TABLE pages (

id SERIAL PRIMARY KEY,

slug VARCHAR(255) UNIQUE NOT NULL,

title VARCHAR(255) NOT NULL,

content TEXT,

meta\_description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE media (

id SERIAL PRIMARY KEY,

filename VARCHAR(255) NOT NULL,

alt\_text TEXT,

file\_size INTEGER,

mime\_type VARCHAR(100),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE forms (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

fields JSON NOT NULL,

settings JSON,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE form\_submissions (

id SERIAL PRIMARY KEY,

form\_id INTEGER REFERENCES forms(id),

data JSON NOT NULL,

ip\_address INET,

submitted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

#### **2. Backend API Structure**

// backend/src/routes/pages.ts

import express from "express";

import { PrismaClient } from "@prisma/client";

const router = express.Router();

const prisma = new PrismaClient();

// Get all pages

router.get("/pages", async (req, res) => {

const pages = await prisma.page.findMany();

res.json(pages);

});

// Get page by slug

router.get("/pages/:slug", async (req, res) => {

const page = await prisma.page.findUnique({

where: { slug: req.params.slug },

});

res.json(page);

});

// Create/update page (admin only)

router.post("/pages", async (req, res) => {

const { slug, title, content, metaDescription } = req.body;

const page = await prisma.page.upsert({

where: { slug },

update: { title, content, metaDescription },

create: { slug, title, content, metaDescription },

});

res.json(page);

});

export default router;

#### **3. Frontend Components**

// frontend/components/PageBuilder.tsx

import React from "react";

import { DndProvider } from "react-dnd";

import { HTML5Backend } from "react-dnd-html5-backend";

interface Component {

id: string;

type: "text" | "image" | "form" | "gallery";

props: Record<string, any>;

}

export const PageBuilder: React.FC = () => {

const [components, setComponents] = useState<Component[]>([]);

const addComponent = (type: Component["type"]) => {

const newComponent: Component = {

id: `${type}-${Date.now()}`,

type,

props: {},

};

setComponents([...components, newComponent]);

};

return (

<DndProvider backend={HTML5Backend}>

<div className="flex h-screen">

{/\* Component Palette \*/}

<div className="w-64 bg-gray-100 p-4">

<h3 className="font-bold mb-4">Components</h3>

<button onClick={() => addComponent("text")}>Add Text Block</button>

<button onClick={() => addComponent("image")}>Add Image</button>

<button onClick={() => addComponent("form")}>Add Form</button>

</div>

{/\* Canvas \*/}

<div className="flex-1 p-4">

{components.map((component) => (

<ComponentRenderer key={component.id} component={component} />

))}

</div>

</div>

</DndProvider>

);

};

### **Phase 2: Core Feature Migration (Weeks 13-24)**

#### **1. Form System Replacement**

// backend/src/services/FormService.ts

export class FormService {

static async processSubmission(formId: string, data: any, ip: string) {

// Anti-spam validation

if (await this.isSpam(data, ip)) {

throw new Error("Submission flagged as spam");

}

// Save to database

const submission = await prisma.formSubmission.create({

data: {

formId: parseInt(formId),

data,

ipAddress: ip,

},

});

// Send notification email

await EmailService.sendFormNotification(formId, data);

return submission;

}

private static async isSpam(data: any, ip: string): Promise<boolean> {

// Rate limiting check

const recentSubmissions = await prisma.formSubmission.count({

where: {

ipAddress: ip,

submittedAt: {

gte: new Date(Date.now() - 60000), // Last minute

},

},

});

return recentSubmissions > 3;

}

}

#### **2. Image Optimization Pipeline**

// backend/src/services/ImageService.ts

import sharp from "sharp";

import { promises as fs } from "fs";

export class ImageService {

static async processUpload(file: Express.Multer.File) {

const filename = `${Date.now()}-${file.originalname}`;

const basePath = `/uploads/${filename}`;

// Generate optimized versions

const variants = await Promise.all([

this.createVariant(file.buffer, "thumbnail", 200, 200),

this.createVariant(file.buffer, "medium", 800, 600),

this.createVariant(file.buffer, "large", 1200, 900),

this.createWebP(file.buffer, "webp", 1200, 900),

]);

// Save to database

const media = await prisma.media.create({

data: {

filename,

altText: "",

fileSize: file.size,

mimeType: file.mimetype,

variants: variants.reduce(

(acc, variant) => ({

...acc,

[variant.size]: variant.path,

}),

{}

),

},

});

return media;

}

private static async createVariant(

buffer: Buffer,

size: string,

width: number,

height: number

) {

const processed = await sharp(buffer)

.resize(width, height, { fit: "inside", withoutEnlargement: true })

.jpeg({ quality: 85 })

.toBuffer();

const path = `/uploads/${size}/${Date.now()}.jpg`;

await fs.writeFile(`./public${path}`, processed);

return { size, path };

}

}

#### **3. SEO Engine Implementation**

// frontend/utils/seo.ts

export interface SEOConfig {

title: string;

description: string;

keywords?: string[];

ogImage?: string;

canonicalUrl?: string;

}

export const generateSEOTags = (config: SEOConfig) => {

return {

title: config.title,

description: config.description,

keywords: config.keywords?.join(", "),

openGraph: {

title: config.title,

description: config.description,

images: config.ogImage ? [{ url: config.ogImage }] : [],

url: config.canonicalUrl,

},

twitter: {

card: "summary\_large\_image",

title: config.title,

description: config.description,

image: config.ogImage,

},

};

};

// Auto-generate sitemap

export const generateSitemap = async () => {

const pages = await fetch("/api/pages").then((r) => r.json());

const sitemap = `<?xml version="1.0" encoding="UTF-8"?>

<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

${pages

.map(

(page: any) => `

<url>

<loc>https://legacyconcierge.com/${page.slug}</loc>

<lastmod>${page.updatedAt}</lastmod>

<priority>0.8</priority>

</url>

`

)

.join("")}

</urlset>`;

return sitemap;

};

### **Phase 3: Advanced Features (Weeks 25-36)**

#### **1. Custom Analytics System**

// backend/src/services/AnalyticsService.ts

export class AnalyticsService {

static async trackPageView(

path: string,

userAgent: string,

ip: string,

referrer?: string

) {

// Privacy-compliant tracking (no personal data)

const event = {

type: "pageview",

path,

timestamp: new Date(),

browser: this.parseBrowser(userAgent),

country: await this.getCountryFromIP(ip),

referrer: this.cleanReferrer(referrer),

};

await prisma.analyticsEvent.create({ data: event });

}

static async getAnalytics(dateRange: { start: Date; end: Date }) {

const pageViews = await prisma.analyticsEvent.groupBy({

by: ["path"],

where: {

type: "pageview",

timestamp: {

gte: dateRange.start,

lte: dateRange.end,

},

},

\_count: true,

orderBy: { \_count: "desc" },

});

return {

pageViews,

totalViews: pageViews.reduce((sum, page) => sum + page.\_count, 0),

uniquePages: pageViews.length,

};

}

}

#### **2. Custom Caching Strategy**

// backend/src/middleware/cache.ts

import Redis from "ioredis";

const redis = new Redis(process.env.REDIS\_URL);

export const cacheMiddleware = (ttl: number = 300) => {

return async (req: any, res: any, next: any) => {

const key = `cache:${req.method}:${req.originalUrl}`;

try {

const cached = await redis.get(key);

if (cached) {

return res.json(JSON.parse(cached));

}

// Override res.json to cache the response

const originalJson = res.json;

res.json = function (data: any) {

redis.setex(key, ttl, JSON.stringify(data));

return originalJson.call(this, data);

};

next();

} catch (error) {

console.error("Cache error:", error);

next();

}

};

};

### **Phase 4: Security & Monitoring (Weeks 37-48)**

#### **1. Security Framework**

// backend/src/middleware/security.ts

import rateLimit from "express-rate-limit";

import helmet from "helmet";

// Rate limiting

export const createRateLimit = (windowMs: number, max: number) =>

rateLimit({

windowMs,

max,

message: "Too many requests from this IP",

standardHeaders: true,

legacyHeaders: false,

});

// Security headers

export const securityHeaders = helmet({

contentSecurityPolicy: {

directives: {

defaultSrc: ["'self'"],

styleSrc: ["'self'", "'unsafe-inline'"],

scriptSrc: ["'self'"],

imgSrc: ["'self'", "data:", "https:"],

connectSrc: ["'self'"],

fontSrc: ["'self'"],

objectSrc: ["'none'"],

mediaSrc: ["'self'"],

frameSrc: ["'none'"],

},

},

hsts: {

maxAge: 31536000,

includeSubDomains: true,

preload: true,

},

});

// Input sanitization

export const sanitizeInput = (req: any, res: any, next: any) => {

const sanitize = (obj: any): any => {

if (typeof obj === "string") {

return obj.replace(/<script.\*?>.\*?<\/script>/gi, "");

}

if (typeof obj === "object" && obj !== null) {

const sanitized: any = {};

for (const [key, value] of Object.entries(obj)) {

sanitized[key] = sanitize(value);

}

return sanitized;

}

return obj;

};

req.body = sanitize(req.body);

req.query = sanitize(req.query);

next();

};

#### **2. Monitoring & Alerting**

// backend/src/services/MonitoringService.ts

export class MonitoringService {

static async logError(error: Error, context: any) {

await prisma.errorLog.create({

data: {

message: error.message,

stack: error.stack,

context: JSON.stringify(context),

timestamp: new Date(),

},

});

// Send alert for critical errors

if (this.isCritical(error)) {

await this.sendAlert(error, context);

}

}

static async checkSystemHealth() {

const checks = await Promise.allSettled([

this.checkDatabase(),

this.checkRedis(),

this.checkDiskSpace(),

this.checkMemoryUsage(),

]);

const healthStatus = {

status: checks.every((c) => c.status === "fulfilled")

? "healthy"

: "unhealthy",

checks: checks.map((check, index) => ({

name: ["database", "redis", "disk", "memory"][index],

status: check.status,

message: check.status === "rejected" ? check.reason.message : "OK",

})),

timestamp: new Date(),

};

return healthStatus;

}

}

## **Deployment Strategy**

### **Docker Configuration**

# frontend/Dockerfile

FROM node:20-alpine AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --only=production

COPY . .

RUN npm run build

FROM node:20-alpine AS runner

WORKDIR /app

COPY --from=builder /app/.next ./.next

COPY --from=builder /app/public ./public

COPY --from=builder /app/package.json ./package.json

COPY --from=builder /app/node\_modules ./node\_modules

EXPOSE 3000

CMD ["npm", "start"]

# docker-compose.custom.yml

version: "3.8"

services:

frontend:

build: ./frontend

ports:

- "3000:3000"

environment:

- API\_URL=http://backend:4000

depends\_on:

- backend

backend:

build: ./backend

ports:

- "4000:4000"

environment:

- DATABASE\_URL=postgresql://user:password@postgres:5432/legacyconcierge

- REDIS\_URL=redis://redis:6379

depends\_on:

- postgres

- redis

postgres:

image: postgres:15-alpine

environment:

- POSTGRES\_DB=legacyconcierge

- POSTGRES\_USER=user

- POSTGRES\_PASSWORD=password

volumes:

- postgres\_data:/var/lib/postgresql/data

redis:

image: redis:7-alpine

volumes:

- redis\_data:/data

volumes:

postgres\_data:

redis\_data:

## **Migration Checklist**

### **Pre-Migration**

1. Complete backup of WordPress site
2. Content audit and cleanup
3. URL mapping for SEO preservation
4. Performance baseline measurements

### **During Migration**

1. Set up parallel development environment
2. Implement content migration scripts
3. Test all functionality thoroughly
4. Configure monitoring and alerting

### **Post-Migration**

1. DNS cutover to new system
2. Monitor performance and errors
3. Verify all redirects working
4. Security audit and penetration testing

## **Maintenance Procedures**

### **Daily**

1. Monitor system health dashboard
2. Review error logs
3. Check performance metrics

### **Weekly**

1. Database optimization
2. Security scan
3. Backup verification
4. Performance tuning

### **Monthly**

1. Security audit
2. Dependency updates
3. Capacity planning review
4. Disaster recovery testing

**Implementation Timeline**: 12 months **Team Size**: 3-4 developers **Budget**: $140,000 - $210,000 **Expected ROI**: 6-9 years