Complete Scope of Work: Pure n8n Architecture

AI Content Copilot Platform (No Copilot Studio)

Executive Summary

What Changed: Complete elimination of Microsoft Copilot Studio. The new architecture is:

- **Frontend:** Custom React dashboard (fully under your control)
- **Backend:** n8n automation engine (all intelligence + orchestration)
- **Data:** SharePoint (existing setup, no changes)

Why This is Better:

- Save $500+/month in licensing costs
- 100% customizable UI/UX
- Better performance and debugging
- No Microsoft ecosystem lock-in
- Modern, industry-standard tech stack

Timeline: 8 weeks
Estimated Effort: 200-220 hours

Project Scope

What We're Building

1. Custom Web Frontend (React Dashboard)

- User authentication
- Analyze keyword form
- Content backlog table view
- Brief generation interface
- Content production page
- Published content gallery
- Responsive design (mobile-friendly)

2. n8n Backend API (10 Workflows)

- Brand management endpoints
- Content analysis with AI
- PAVE scoring automation
- Brief generation
- Full content production (HTML + Images)
- Publishing workflow
- Status management

3. SharePoint Integration

- Read/write to Brand Configuration
- Full CRUD on Content Pipeline
- File operations (HTML storage)

4. AI Service Integration

- Bing Search API (web search)
- Azure OpenAI GPT-4 (content generation)
- DALL-E 3 (image generation)

Deliverables Checklist

Frontend Deliverables

- [ ] React application source code
- [ ] Responsive UI (desktop + mobile)
- [ ] User authentication system
- [ ] 6 main pages:
  - [ ] Dashboard/Home
  - [ ] Analyze Keyword
  - [ ] Content Backlog
  - [ ] Create Brief

    - [ ] Produce Content
    - [ ] Published Gallery
- [ ] Deployed to Vercel/Netlify
- [ ] Production-ready build

Backend Deliverables (n8n)

- [ ] 10 complete workflows (JSON exports)
- [ ] API documentation (OpenAPI/Swagger)
- [ ] Authentication workflow (JWT)
- [ ] Error handling & retry logic
- [ ] State persistence for long-running tasks
- [ ] Monitoring & logging setup
- [ ] Deployed to n8n Cloud or self-hosted

Documentation Deliverables

- [ ] System architecture diagrams
- [ ] API reference guide
- [ ] Frontend setup guide
- [ ] n8n workflow documentation
- [ ] User manual (how to use the system)
- [ ] Admin guide (maintenance & updates)
- [ ] Deployment playbook

Testing Deliverables

- [ ] Unit tests for critical workflows
- [ ] End-to-end test suite
- [ ] Performance test results
- [ ] UAT completion report

Phase-by-Phase Breakdown

Phase 0: Foundation Setup (Week 1) - 16 hours

0.1 Infrastructure Setup (8 hours)

n8n Deployment:

- [ ] Choose hosting option (n8n Cloud vs Self-hosted)
- [ ] Deploy n8n instance
- [ ] Configure PostgreSQL database
- [ ] Set up SSL/HTTPS
- [ ] Configure environment variables

Frontend Setup:

- [ ] Create React project (Vite + TypeScript)
- [ ] Install dependencies:
    - React Router
    - React Query
    - Axios
    - TailwindCSS
    - Shadcn/ui components
- [ ] Set up project structure
- [ ] Configure ESLint + Prettier
- [ ] Set up Git repository

0.2 API Credentials Configuration (4 hours)

Azure OpenAI:

- [ ] Create Azure OpenAI resource
- [ ] Get API key and endpoint
- [ ] Test GPT-4 connection
- [ ] Test DALL-E 3 access

Bing Search API:

- [ ] Get Bing Search API key
- [ ] Configure rate limits
- [ ] Test search functionality

SharePoint:

- [ ] Set up OAuth 2.0 app registration
- [ ] Get client ID + secret
- [ ] Test SharePoint connection
- [ ] Verify list access permissions

0.3 SharePoint Verification (4 hours)

Verify Existing Setup:

- [ ] Brand Configuration list schema
- [ ] Content Pipeline list schema
- [ ] Test data creation
- [ ] Lookup field relationships
- [ ] File storage location

Create Test Data:

- [ ] 2-3 sample brands
- [ ] 5-10 sample content items
- [ ] Sample CSS wrapper file

Phase 1: Backend Development (n8n Workflows) (Weeks 2-4) - 80 hours

Week 2: Foundation Workflows (24 hours)

Flow 1: Authentication System (6 hours)

Endpoint: `POST /api/auth/login`

Functionality:

- User authentication (SharePoint users or custom DB)
- JWT token generation
- Token validation middleware
- Session management

n8n Nodes:

1. Webhook trigger (POST /api/auth/login)
2. IF node: Validate credentials
3. Code node: Generate JWT token
4. Respond to webhook (success or error)

Testing:

- [ ] Valid login
- [ ] Invalid credentials
- [ ] Token expiration
- [ ] Token refresh

Flow 2: Get Brands List (4 hours)

Endpoint: `GET /api/brands/list`

Functionality:

- Fetch all active brands from SharePoint
- Return simple array of {id, name}

n8n Nodes:

1. Webhook trigger (GET)
2. Validate JWT token
3. SharePoint: Get Items (Brand Configuration)
4. Code: Format as array
5. Respond with JSON

Error Handling:

- SharePoint timeout → Retry once
- No brands found → Return empty array

Flow 3: Get Brand Details (4 hours)

Endpoint: `GET /api/brands/{id}`

Functionality:

- Get complete brand configuration
- Return persona, tone, context, CSS link, disclaimer, banned tags

n8n Nodes:

1. Webhook trigger with ID parameter
2. SharePoint: Get Item by ID
3. Code: Extract all fields
4. Respond with brand object

Flow 4: Content Analysis - PAVE Scorer (10 hours)

Endpoint: `POST /api/content/analyze`

This is the most complex flow in Week 2

Functionality:

1. Receive brand_id + keyword
2. Get brand details from SharePoint
3. Perform Bing web search
4. Summarize top 10 results
5. Call GPT-4 with PAVE scoring prompt
6. Validate JSON response
7. Return typed scores + reasonings

n8n Workflow Structure:

```
Webhook (POST /api/content/analyze)
  ↓
Validate Input (brand_id, keyword)
  ↓
SharePoint: Get Brand Details
  ↓
Bing Search API Request
  ↓
Code: Summarize Search Results (top 10)
  ↓
Code: Build GPT Prompt (with brand context)
  ↓
OpenAI: GPT-4 API Call
  ↓
Code: Validate JSON Schema
  ↓
IF: Valid?
  ├─ Yes → Return PAVE Scores
  └─ No → Retry (max 3 times)
         └─ Return Fallback Scores
```

Detailed Implementation:

Step 1: Bing Search

```
// HTTP Request Node
const keyword = $json.keyword;
const apiKey = $env.BING_API_KEY;

return {
  method: 'GET',
  url: 'https://api.bing.microsoft.com/v7.0/search',
  headers: {
    'Ocp-Apim-Subscription-Key': apiKey
  },
  qs: {
    q: keyword,
    count: 10,
    mkt: 'en-US',
    safeSearch: 'Moderate'
  }
};
```

Step 2: Summarize Results

```
// Code Node
const results = $json.webPages.value;

const summary = results.map((result, index) => {
  return `${index + 1}. ${result.name}
   Snippet: ${result.snippet}
   URL: ${result.url}`;
}).join('\n\n');

return {
  search_summary: summary.substring(0, 2000) // Limit to 2000 chars
};
```

Step 3: GPT-4 PAVE Scoring

```
// OpenAI Node Configuration
{
  "model": "gpt-4",
  "messages": [
    {
      "role": "system",
      "content": "You are an SEO strategist. Return ONLY valid JSON (no markdown)."
    },
    {
      "role": "user",
      "content": `Brand: {{$json.brand_name}}
Persona: {{$json.brand_persona}}
Product context: {{$json.brand_product_context}}
Keyword: {{$json.keyword}}
Search summary: {{$json.search_summary}}

Rules:
- Keep each reasoning <= 15 words.
- volume_score must be 0 and volume_reasoning must be "Manual check required".

Output JSON:
{
  "profitability_score": 1-5,
  "profitability_reasoning": "...",
  "authority_score": 1-5,
  "authority_reasoning": "...",
  "volume_score": 0,
  "volume_reasoning": "Manual check required",
  "effort_score": 1-5,
  "effort_reasoning": "..."
}`
    }
  ],
  "temperature": 0.3,
  "max_tokens": 500
}
```

Step 4: Validation & Retry Logic

```
// Code Node: Validate Schema
const response = $json.choices[0].message.content;

// Try to parse JSON
let parsed;
try {
  // Remove markdown code blocks if present
  const cleaned = response.replace(/```json|```/g, '').trim();
  parsed = JSON.parse(cleaned);
} catch (e) {
  return {
    valid: false,
    error: 'JSON parse failed',
    retry_needed: true
  };
}

// Validate schema
const requiredFields = [
  'profitability_score', 'profitability_reasoning',
  'authority_score', 'authority_reasoning',
  'volume_score', 'volume_reasoning',
  'effort_score', 'effort_reasoning'
];

const missing = requiredFields.filter(field => !(field in parsed));

if (missing.length > 0) {
  return {
    valid: false,
    missing_fields: missing,
    retry_needed: true
  };
}

// Validate data types
if (typeof parsed.profitability_score !== 'number' ||
    typeof parsed.authority_score !== 'number' ||
    typeof parsed.effort_score !== 'number') {
  return {
    valid: false,
    error: 'Scores must be numbers',
    retry_needed: true
  };
}

// Success!
return {
  valid: true,
  pave_scores: parsed
};
```

Error Handling Strategy:

- Retry 1: Same prompt with "CRITICAL: Return ONLY JSON, no other text"
- Retry 2: Simplified prompt with example JSON
- Retry 3 (final): Return fallback scores:

```
{
  "profitability_score": 3,
  "profitability_reasoning": "Unable to analyze",
  "authority_score": 3,
  "authority_reasoning": "Unable to analyze",
  "volume_score": 0,
  "volume_reasoning": "Manual check required",
  "effort_score": 3,
  "effort_reasoning": "Unable to analyze",
  "_error": true,
  "_message": "AI analysis failed after 3 attempts"
}
```

Testing Checklist:

- ☐ Valid keyword → Returns proper scores
- ☐ Invalid keyword → Handles gracefully
- ☐ Bing API timeout → Retry logic works
- ☐ GPT returns invalid JSON → Retries and recovers
- ☐ All retries fail → Returns fallback scores
- ☐ Performance: < 30 seconds total

Week 3: Content Management Workflows (28 hours)

Flow 5: Save Scored Idea (6 hours)

Endpoint: `POST /api/content/save`

Functionality:

- Receive keyword + brand + PAVE scores
- Calculate total score (P+A+V+E)
- Auto-approval: IF total >= 18 → "Approved for Briefing", ELSE "PAVE Scored"
- Create item in SharePoint Content Pipeline

n8n Workflow:

```
Webhook (POST /api/content/save)
  ↓
Code: Calculate Total Score
  ↓
IF: Total >= 18?
  ├─ Yes → status = "Approved for Briefing"
  └─ No → status = "PAVE Scored"
  ↓
SharePoint: Get Brand ID (lookup)
  ↓
SharePoint: Create Item in Content Pipeline
  ↓
Respond: {success, item_id, status}
```

Field Mapping:

```
{
  "Title": keyword,
  "Target_BrandId": brand_id, // Lookup field
  "Status": status,
  "PAVE_Score_P": P,
  "PAVE_Score_A": A,
  "PAVE_Score_V": V,
  "PAVE_Score_E": E,
  "Search_Summary": search_summary,
  "AI_Reasoning": ai_reasoning
}
```

Flow 6: Fetch Backlog Ideas (4 hours)

Endpoint: `GET /api/content/backlog?status=Approved&limit=10`

Functionality:

- Get items from Content Pipeline
- Filter by status (default: "Approved for Briefing")
- Sort by PAVE_Score_P descending
- Limit results (default: 10)

Response Format:

```
{
  "items": [
    {
      "id": 42,
      "title": "best vitamin D supplements",
      "brand": "Vit Cornu",
      "pave_scores": {"P": 4, "A": 3, "V": 4, "E": 4},
      "total_score": 15,
      "status": "Approved for Briefing",
      "created_date": "2026-02-01"
    }
  ],
  "total": 5
}
```

Flow 7: Fetch Item Details (4 hours)

Endpoint: `GET /api/content/details/{id}`

Functionality:

- Get single item by ID
- Return all fields including authority_brief

Flow 8: Brief Recommendation (6 hours)

Endpoint: `POST /api/brief/recommend`

Functionality:

- Use GPT-4 to analyze keyword intent
- Recommend content format (Comparison, Guide, Listicle, etc.)
- Based on search summary and keyword

GPT Prompt:

```
 Analyze this keyword and search context:
Keyword: "{{keyword}}"
Search Summary: "{{search_summary}}"

Based on search intent, recommend ONE content format:
1. Comparison Page (product/service comparisons)
2. In-Depth Guide (educational, how-to)
3. Listicle (top 10, best of)
4. Landing Page (tool/resource)
5. Research Report (data-driven)

Return JSON:
{
  "format": "Comparison Page",
  "reasoning": "High commercial intent with product focus",
  "alternative": "In-Depth Guide"
}
```

Flow 9: Generate Authority Brief (8 hours)

Endpoint: `POST /api/brief/generate`

Functionality:

- Get item details + brand context
- Use GPT-4 to generate structured brief
- Save to SharePoint
- Update status to "Briefed"

GPT Prompt Structure:

```
You are a content strategist creating an authority brief.

Keyword: {{keyword}}
Brand: {{brand_name}}
Persona: {{brand_persona}}
Tone: {{brand_tone}}
Format: {{selected_format}}
Search Summary: {{search_summary}}
Differentiator: {{user_differentiator}}
SME Input: {{sme_input}}

Create a detailed authority brief with these sections:
1. Content Goal
2. Target Audience
3. Content Outline (H2/H3 structure)
4. Key Points to Cover
5. Sources & Research Requirements
6. Success Metrics

Output in Markdown format.
```

n8n Implementation:

```
Webhook
  ↓
SharePoint: Get Item Details
  ↓
SharePoint: Get Brand Details
  ↓
Code: Build Brief Generation Prompt
  ↓
OpenAI: GPT-4
  ↓
SharePoint: Update Item
  - Authority_Brief = generated_text
  - Status = "Briefed"
  ↓
Respond: {success, brief_text}
```

Week 4: Content Production Workflows (28 hours)

Flow 10: Produce Content (The Master Workflow) (20 hours)

Endpoint: `POST /api/content/produce`

This is the most complex workflow in the entire system.

Functionality:

1. Get item + brand details (personas, tone, constraints)
2. Generate HTML article with GPT-4
3. Validate HTML (no banned tags)
4. Generate DALL-E image prompt with GPT-4
5. Create cover image with DALL-E 3
6. Fetch CSS wrapper template
7. Assemble final HTML (CSS + content + image)
8. Save HTML file to SharePoint
9. Update item status to "In Production"

Complete n8n Workflow:

```
Webhook (POST /api/content/produce)
  ↓
SharePoint: Get Item by ID
  ↓
SharePoint: Get Brand Details (via lookup)
  ↓
Code: Prepare All Context Variables
  ↓
┌─────────────────────────────────────┐
│ STEP 1: Generate HTML Content       │
└─────────────────────────────────────┘
  ↓
OpenAI: GPT-4 Investigative Writer
  ↓
Code: Validate HTML
  ├─ Check for banned tags
  ├─ Ensure proper structure
  └─ Strip disallowed elements
  ↓
Code: Save State (html_generated)
  ↓
┌─────────────────────────────────────┐
│ STEP 2: Generate Cover Image        │
└─────────────────────────────────────┘
  ↓
OpenAI: GPT-4 Image Prompt Generator
  ↓
OpenAI: DALL-E 3 Image Generation
  ├─ Timeout: 90 seconds
  ├─ Success → Get image URL
  └─ Fail → Use placeholder image
  ↓
Code: Save State (image_generated)
  ↓
┌─────────────────────────────────────┐
│ STEP 3: Assemble Final HTML         │
└─────────────────────────────────────┘
  ↓
HTTP Request: Fetch CSS Wrapper
  ↓
Code: String Assembly
  ├─ Replace {{CONTENT}}
  ├─ Replace {{COVER_IMAGE_URL}}
  └─ Replace {{TITLE}}
  ↓
Code: Save State (assembly_complete)
  ↓
┌─────────────────────────────────────┐
│ STEP 4: Save to SharePoint          │
└─────────────────────────────────────┘
  ↓
SharePoint: Create File
  - Folder: /Master Assets/
  - Filename: {{title}}.html
  - Content: {{final_html}}
  ↓
SharePoint: Update Item
  - Master_Asset_Link = file_url
  - Status = "In Production"
  ↓
Respond: {success, asset_url, generation_time}
```

Detailed Step Implementations:

Step 1: HTML Generation

```
// OpenAI Node: Investigative Writer
{
  "model": "gpt-4",
  "messages": [
    {
      "role": "system",
      "content": `You are an investigative reporter for {{brand_name}}.
Persona: {{brand_persona}}
Tone: {{brand_tone}}

MANDATORY:
- Investigate → Draft → Audit
- Two-source rule for factual claims
- Append disclaimer block: {{brand_disclaimer_block}}

HTML requirements:
- Output HTML BODY only (h1, h2, p, ul, blockquote)
- BANNED TAGS: {{brand_banned_html_tags}}
- NO: <table>, float, iframe, background-color

At the end include:
AUDIT TRAIL: List major claims and 2 sources per claim.`
    },
    {
      "role": "user",
      "content": `Topic: {{title}}
Background Research: {{search_summary}}
Authority Brief: {{authority_brief}}

Generate a comprehensive investigative article (2000-2500 words).`
    }
  ],
  "temperature": 0.7,
  "max_tokens": 4000
}
```

Step 1b: HTML Validation

```
// Code Node: Validate HTML
const html = $json.choices[0].message.content;
const bannedTags = $('SharePoint').item.json.Banned_HTML_Tags.split(',').map(t => t.trim());

// Remove banned tags
let cleanHtml = html;
bannedTags.forEach(tag => {
  const regex = new RegExp(`<${tag}[^>]*>.*?<\/${tag}>`, 'gis');
  cleanHtml = cleanHtml.replace(regex, '');

  // Also remove self-closing versions
  const selfClosing = new RegExp(`<${tag}[^>]*\/>`, 'gi');
  cleanHtml = cleanHtml.replace(selfClosing, '');
});

// Validate structure (must have h1, h2, p)
const hasH1 = /<h1[^>]*>/.test(cleanHtml);
const hasH2 = /<h2[^>]*>/.test(cleanHtml);
const hasP = /<p[^>]*>/.test(cleanHtml);

if (!hasH1 || !hasH2 || !hasP) {
  throw new Error('HTML missing required structure (h1, h2, p)');
}

return {
  html_body: cleanHtml,
  validation_passed: true,
  word_count: cleanHtml.split(/\s+/).length
};
```

Step 2: Image Generation

```
// OpenAI Node: Image Prompt Generator
{
  "model": "gpt-4",
  "messages": [
    {
      "role": "user",
      "content": `Generate a DALL-E 3 prompt for a cover image.
Article title: "{{title}}"
Article topic: "{{search_summary}}"

Requirements:
- Professional, modern style
- Relevant to content topic
- No text in image
- Landscape orientation
- Clean, simple composition

Output ONLY the DALL-E prompt, nothing else.`
    }
  ],
  "temperature": 0.8,
  "max_tokens": 200
}

// Then DALL-E Node
{
  "model": "dall-e-3",
  "prompt": "{{$json.dalle_prompt}}",
  "size": "1792x1024",
  "quality": "hd",
  "n": 1
}
```

Step 3: Assembly

```
// Code Node: Assemble Final HTML
const cssWrapper = $('HTTP Request').item.json;
const htmlBody = $('Validate HTML').item.json.html_body;
const coverImageUrl = $('DALL-E').item.json.data[0].url;
const title = $('SharePoint Get Item').item.json.Title;

let finalHtml = cssWrapper;

// Perform replacements
finalHtml = finalHtml.replace(/\{\{CONTENT\}\}/g, htmlBody);
finalHtml = finalHtml.replace(/\{\{COVER_IMAGE_URL\}\}/g, coverImageUrl);
finalHtml = finalHtml.replace(/\{\{TITLE\}\}/g, title);

// Add metadata
const metadata = `
<!-- Generated by AI Content Copilot -->
<!-- Date: ${new Date().toISOString()} -->
<!-- Brand: ${$('SharePoint Brand').item.json.Brand} -->
`;

finalHtml = finalHtml.replace('</head>', `${metadata}</head>`);

return {
  final_html: finalHtml,
  file_size_kb: Math.round(finalHtml.length / 1024)
};
```

Step 4: Save & Update

```
// SharePoint: Create File Node
{
  "site_url": "{{$env.SHAREPOINT_SITE}}",
  "folder_path": "/Master Assets/",
  "file_name": "{{$json.title}}.html",
  "file_content": "{{$json.final_html}}",
  "content_type": "text/html"
}

// SharePoint: Update Item Node
{
  "item_id": "{{$json.sharepoint_id}}",
  "fields": {
    "Master_Asset_Link": "{{$('Create File').item.json.file_url}}",
    "Status": "In Production"
  }
}
```

State Persistence (Critical!):

```
// After each major step, save state for resume capability
const executionId = $execution.id;
const state = {
  execution_id: executionId,
  item_id: $json.sharepoint_id,
  step: 'html_generated', // or 'image_generated', 'assembly_complete'
  data: {
    html_body: $json.html_body,
    timestamp: new Date().toISOString()
  }
};

// Save to n8n's execution data or external DB
await $http.request({
  method: 'POST',
  url: 'http://localhost:5678/webhook/save-execution-state',
  body: state
});
```

Error Handling:

- GPT timeout → Retry once
- DALL-E timeout → Use placeholder image
- CSS wrapper not found → Use basic HTML template
- SharePoint file creation fails → Retry, then error
- Any step fails → Update item status to "Error" with notes

Performance Targets:

- Total time: < 3 minutes
- HTML generation: < 60 seconds
- Image generation: < 60 seconds
- Assembly + save: < 30 seconds

Testing Scenarios:

- ☐ Normal flow (all steps succeed)
- ☐ GPT returns invalid HTML
- ☐ DALL-E times out
- ☐ CSS wrapper 404
- ☐ SharePoint file creation fails
- ☐ Resume from failure (state recovery)

Flow 11: Publish Content (4 hours)

Endpoint: `POST /api/content/publish`

Functionality:

- Copy HTML file to public folder
- OR publish to external CMS (WordPress API)
- Update status to "Published"
- Record published URL

Flow 12: Update Status (4 hours)

Endpoint: `PUT /api/content/status/{id}`

Functionality:

- Generic status updater
- Validate status transitions
- Add timestamp and notes

Phase 2: Frontend Development (Weeks 5-6) - 64 hours

Week 5: Core Pages (32 hours)

5.1 Authentication & Layout (8 hours)

Login Page:

```
// src/pages/Login.jsx
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { loginUser } from '../services/auth';

export default function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const { token, user } = await loginUser(email, password);
      localStorage.setItem('token', token);
      navigate('/dashboard');
    } catch (err) {
      setError('Invalid credentials');
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-50">
      <div className="max-w-md w-full space-y-8 p-8 bg-white rounded-lg shadow">
        <h2 className="text-3xl font-bold text-center">AI Content Copilot</h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            placeholder="Email"
            className="w-full px-4 py-2 border rounded"
          />
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            placeholder="Password"
            className="w-full px-4 py-2 border rounded"
          />
          {error && <p className="text-red-500">{error}</p>}
          <button type="submit" className="w-full bg-blue-600 text-white py-2 rounded">
            Login
          </button>
        </form>
      </div>
    </div>
  );
}
```

Main Layout:

```
// src/components/layout/MainLayout.jsx
export default function MainLayout({ children }) {
  return (
    <div className="min-h-screen bg-gray-50">
      <Navbar />
      <div className="flex">
        <Sidebar />
        <main className="flex-1 p-8">
          {children}
        </main>
      </div>
    </div>
  );
}
```

5.2 Analyze Keyword Page (10 hours)

Full Implementation:

```
// src/pages/AnalyzeKeyword.jsx
import { useState } from 'react';
import { useBrands } from '../hooks/useBrands';
import { analyzeKeyword, saveIdea } from '../services/content';
import PAVEScoreCard from '../components/PAVEScoreCard';

export default function AnalyzeKeyword() {
  const { brands, loading: brandsLoading } = useBrands();
  const [brandId, setBrandId] = useState('');
  const [keyword, setKeyword] = useState('');
  const [analyzing, setAnalyzing] = useState(false);
  const [scores, setScores] = useState(null);

  const handleAnalyze = async () => {
    setAnalyzing(true);
    try {
      const result = await analyzeKeyword({ brand_id: brandId, keyword });
      setScores(result.pave_scores);
    } catch (error) {
      alert('Analysis failed');
    } finally {
      setAnalyzing(false);
    }
  };

  const handleSave = async (finalScores) => {
    try {
      await saveIdea({
        brand_id: brandId,
        keyword,
        pave_scores: finalScores,
        ...scores
      });
      alert('Idea saved to backlog!');
    } catch (error) {
      alert('Save failed');
    }
  };

  return (
    <div className="max-w-4xl mx-auto">
      <h1 className="text-3xl font-bold mb-8">Analyze Keyword</h1>

      {/* Brand Selection */}
```

```
      <div className="bg-white p-6 rounded-lg shadow mb-6">
        <label className="block mb-2 font-semibold">Select Brand</label>
        <select
          value={brandId}
          onChange={(e) => setBrandId(e.target.value)}
          className="w-full p-2 border rounded"
        >
          <option value="">-- Select Brand --</option>
          {brands.map(b => (
            <option key={b.id} value={b.id}>{b.name}</option>
          ))}
        </select>
      </div>

      {/* Keyword Input */}
      <div className="bg-white p-6 rounded-lg shadow mb-6">
        <label className="block mb-2 font-semibold">Keyword</label>
        <input
          type="text"
          value={keyword}
          onChange={(e) => setKeyword(e.target.value)}
          placeholder="e.g., best vitamin D supplements"
          className="w-full p-2 border rounded"
        />
        <button
          onClick={handleAnalyze}
          disabled={!brandId || !keyword || analyzing}
          className="mt-4 bg-blue-600 text-white px-6 py-2 rounded disabled:bg-gray-400"
        >
          {analyzing ? 'Analyzing...' : 'Analyze with AI'}
        </button>
      </div>

      {/* PAVE Scores */}
      {scores && (
        <PAVEScoreCard
          scores={scores}
          onSave={handleSave}
        />
      )}
    </div>
  );
}
```

PAVE Score Card Component:

```
// src/components/PAVEScoreCard.jsx
import { useState } from 'react';

export default function PAVEScoreCard({ scores, onSave }) {
  const [editedScores, setEditedScores] = useState({
    P: scores.profitability.score,
    A: scores.authority.score,
    V: 0, // User must enter
    E: scores.effort.score
  });

  const totalScore = Object.values(editedScores).reduce((a, b) => a + b, 0);

  return (
    <div className="bg-white p-6 rounded-lg shadow">
      <h2 className="text-2xl font-bold mb-4">PAVE Scores</h2>

      {/* Profitability */}
      <div className="mb-4 p-4 border rounded">
        <div className="flex justify-between items-center mb-2">
```

```
    <div className="flex justify-between items-center mb-2">
      <span className="font-semibold">Profitability (P)</span>
      <input
        type="number"
        min="1"
        max="5"
        value={editedScores.P}
        onChange={(e) => setEditedScores({...editedScores, P: +e.target.value})}
        className="w-16 p-1 border rounded"
      />
    </div>
    <p className="text-sm text-gray-600">{scores.profitability.reasoning}</p>
  </div>

  {/* Authority */}
  <div className="mb-4 p-4 border rounded">
    <div className="flex justify-between items-center mb-2">
      <span className="font-semibold">Authority (A)</span>
      <input
        type="number"
        min="1"
        max="5"
        value={editedScores.A}
        onChange={(e) => setEditedScores({...editedScores, A: +e.target.value})}
        className="w-16 p-1 border rounded"
      />
    </div>
    <p className="text-sm text-gray-600">{scores.authority.reasoning}</p>
  </div>

  {/* Volume (Manual Entry) */}
  <div className="mb-4 p-4 border rounded bg-yellow-50">
    <div className="flex justify-between items-center mb-2">
      <span className="font-semibold text-orange-600">Volume (V) - Manual Required</span>
      <input
        type="number"
        min="1"
        max="5"
        value={editedScores.V}
        onChange={(e) => setEditedScores({...editedScores, V: +e.target.value})}
        className="w-16 p-1 border rounded"
        placeholder="1-5"
      />
    </div>
    <p className="text-sm text-gray-600">Check search volume manually and enter score</p>
  </div>

  {/* Effort */}
  <div className="mb-4 p-4 border rounded">
    <div className="flex justify-between items-center mb-2">
      <span className="font-semibold">Effort (E)</span>
      <input
        type="number"
        min="1"
        max="5"
        value={editedScores.E}
        onChange={(e) => setEditedScores({...editedScores, E: +e.target.value})}
        className="w-16 p-1 border rounded"
      />
    </div>
    <p className="text-sm text-gray-600">{scores.effort.reasoning}</p>
  </div>

  {/* Total & Save */}
  <div className="mt-6 p-4 bg-blue-50 rounded">
    <div className="flex justify-between items-center mb-4">
```

```
          <span className="text-xl font-bold">Total Score</span>
          <span className="text-3xl font-bold text-blue-600">{totalScore} / 20</span>
        </div>
        <div className="mb-4">
          <span className={`px-4 py-2 rounded ${totalScore >= 18 ? 'bg-green-500' : 'bg-yellow-500'} text-white font-semibold`}>
            Status: {totalScore >= 18 ? 'Approved for Briefing' : 'PAVE Scored'}
          </span>
        </div>
        <button
          onClick={() => onSave(editedScores)}
          disabled={editedScores.V === 0}
          className="w-full bg-blue-600 text-white py-2 rounded disabled:bg-gray-400"
        >
          Save to Backlog
        </button>
      </div>
    </div>
  );
}
```

5.3 Content Backlog Page (8 hours)

Table View with Filters:

```
// src/pages/ContentBacklog.jsx
import { useState } from 'react';
import { useContentBacklog } from '../hooks/useContent';

export default function ContentBacklog() {
  const [statusFilter, setStatusFilter] = useState('Approved for Briefing');
  const { items, loading } = useContentBacklog({ status: statusFilter });

  return (
    <div>
      <h1 className="text-3xl font-bold mb-8">Content Backlog</h1>

      {/* Filters */}
      <div className="bg-white p-4 rounded-lg shadow mb-6">
        <label className="mr-4">Filter by Status:</label>
        <select
          value={statusFilter}
          onChange={(e) => setStatusFilter(e.target.value)}
          className="p-2 border rounded"
        >
          <option value="Approved for Briefing">Approved for Briefing</option>
          <option value="PAVE Scored">PAVE Scored</option>
          <option value="Briefed">Briefed</option>
          <option value="In Production">In Production</option>
          <option value="Published">Published</option>
        </select>
      </div>

      {/* Table */}
      <div className="bg-white rounded-lg shadow overflow-hidden">
        <table className="w-full">
          <thead className="bg-gray-50">
            <tr>
              <th className="px-6 py-3 text-left">Title</th>
              <th className="px-6 py-3 text-left">Brand</th>
              <th className="px-6 py-3 text-center">P</th>
              <th className="px-6 py-3 text-center">A</th>
              <th className="px-6 py-3 text-center">V</th>
              <th className="px-6 py-3 text-center">E</th>
              <th className="px-6 py-3 text-center">Total</th>
```

```
              <th className="px-6 py-3 text-left">Status</th>
              <th className="px-6 py-3 text-left">Actions</th>
            </tr>
          </thead>
          <tbody>
            {items.map(item => (
              <tr key={item.id} className="border-t hover:bg-gray-50">
                <td className="px-6 py-4">{item.title}</td>
                <td className="px-6 py-4">{item.brand}</td>
                <td className="px-6 py-4 text-center">{item.pave_scores.P}</td>
                <td className="px-6 py-4 text-center">{item.pave_scores.A}</td>
                <td className="px-6 py-4 text-center">{item.pave_scores.V}</td>
                <td className="px-6 py-4 text-center">{item.pave_scores.E}</td>
                <td className="px-6 py-4 text-center font-bold">{item.total_score}</td>
                <td className="px-6 py-4">
                  <StatusBadge status={item.status} />
                </td>
                <td className="px-6 py-4">
                  <button className="text-blue-600 hover:underline">View</button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
}
```

5.4 Create Brief Page (6 hours)

Brief generation interface with format recommendation.

Week 6: Production Pages & Polish (32 hours)

6.1 Produce Content Page (12 hours)

Multi-step progress indicator for content generation.

6.2 Published Gallery (6 hours)

Grid view of published content with preview.

6.3 UI Polish & Responsiveness (8 hours)

- Mobile optimization
- Loading states
- Error handling
- Toasts/notifications

6.4 Testing & Bug Fixes (6 hours)

Phase 3: Integration & Testing (Week 7) - 32 hours

7.1 API Integration Testing (12 hours)

- Connect all frontend pages to n8n endpoints
- Test error scenarios
- Validate response formats

7.2 End-to-End Testing (12 hours)

- Journey 1: Analyze → Save
- Journey 2: Backlog → Brief → Save
- Journey 3: Produce → Publish

7.3 Performance Testing (8 hours)

- Load testing
- Response time optimization
- Database query optimization

Phase 4: Deployment & Handover (Week 8) - 24 hours

8.1 Production Deployment (12 hours)

- Deploy n8n to production
- Deploy frontend to Vercel
- Configure environment variables
- SSL/HTTPS setup

8.2 Documentation & Training (8 hours)

- Record video walkthroughs
- Create user guide
- Admin training session

8.3 Post-Launch Support (4 hours)

- Bug fixes
- Performance tuning

Total Effort Summary

Phase	Hours
Phase 0: Setup	16
Phase 1: n8n Workflows	80
Phase 2: Frontend	64
Phase 3: Testing	32
Phase 4: Deployment	24
TOTAL	**216 hours**

Timeline: 8 weeks @ 25-30 hours/week

Cost Breakdown

Development (One-Time)

- 216 hours @ $[your rate] = $[total]

Monthly Infrastructure

- n8n Cloud (Pro): $50/month
- Frontend Hosting (Vercel): $20/month
- Azure OpenAI API: ~$200/month
- Bing Search API: ~$50/month
- SharePoint: Included in M365
- **Total:** ~$320/month

Savings vs Copilot Studio

- Old architecture: ~$800/month
- New architecture: ~$320/month
- **Monthly savings:** $480
- **Annual savings:** $5,760

✅ Success Criteria

Performance

- ☐ Keyword analysis: < 30 seconds
- ☐ Content production: < 3 minutes
- ☐ Page load times: < 2 seconds
- ☐ 99% uptime during business hours

Functionality

- ☐ All 3 user journeys working
- ☐ PAVE auto-approval working correctly
- ☐ HTML generation includes disclaimer
- ☐ No banned tags in output
- ☐ Images generated successfully
- ☐ Files saved to SharePoint

Quality

- ☐ Mobile-responsive design
- ☐ Comprehensive error handling
- ☐ State recovery working
- ☐ Clear user feedback

Ready to start? Let's build this! 🚀