Fase 3: Content-Plan Shared Service Layer - Ontwerp

Datum: 16 december 2025

Executive Summary

Dit document beschrijft het ontwerp van een gedeelde service layer voor content-plan functionaliteit, die gebruikt wordt door zowel `/api/client/content-plan/*` als `/api/simplified/content-plan/*` routes.

Doel: Elimineer 60-70% code duplicatie terwijl backwards compatibility behouden blijft.

Architectuur Overzicht

File Structuur

```
lib/services/
└── content-plan-service.ts    (Nieuwe shared service)

app/api/client/content-plan/
├── route.ts                    (Refactored - gebruikt service)
├── refresh/route.ts            (Refactored - gebruikt service)
└── add-ideas/route.ts          (Refactored - gebruikt service)

app/api/simplified/content-plan/
├── route.ts                    (Refactored - gebruikt service)
└── analyze-wordpress/route.ts  (Refactored - gebruikt service)
```

Service Layer Design

Module: `lib/services/content-plan-service.ts`

1. Types & Interfaces

```typescript
// Content Plan Topic (gebruikt door beide API sets)
export interface ContentPlanTopic {
  title: string;
  description: string;
  keywords: string[];
  priority: 'high' | 'medium' | 'low';
  reason?: string;
}

// Article Idea Data (voor database operations)
export interface ArticleIdeaData {
  clientId: string;
  projectId: string | null;
  title: string;
  slug: string;
  focusKeyword: string;
  topic?: string;
  description?: string;
  secondaryKeywords: string[];
  searchIntent: string;
  difficulty: number;
  contentOutline: {
    sections: Array<{
      heading: string;
      subpoints: string[];
    }>;
  };
  contentType: string;
  priority: number | string;
  aiScore: number;
  trending?: boolean;
  competitorGap?: boolean;
  status: string;
  targetKeyword?: string;
  reason?: string;
  keywords?: string[];
  searchVolume?: number;
}

// Generation Options
export interface GenerateContentIdeasOptions {
  keywords?: string[];
  keyword?: string;
  projectContext?: {
    name: string;
    websiteUrl?: string;
    niche?: string;
  };
  language?: string;
  count?: number; // Number of ideas to generate
  temperature?: number;
}

// WordPress Post Type
export interface WordPressPost {
  id: number;
  title: { rendered: string };
  excerpt?: { rendered: string };
  categories?: number[];
  tags?: number[];
}
```

```
// Validation Result
export interface ValidationResult {
  client: any;
  project?: any;
}
```

2. Authentication & Validation

```typescript
/**
 * Validate client authentication
 * @throws Error if not authenticated or client not found
 */
export async function validateClient(
  session: any
): Promise<any> {
  if (!session?.user?.email) {
    throw new Error('UNAUTHORIZED');
  }

  const client = await prisma.client.findUnique({
    where: { email: session.user.email },
  });

  if (!client) {
    throw new Error('CLIENT_NOT_FOUND');
  }

  return client;
}

/**
 * Validate project ownership
 * @throws Error if project not found or not owned by client
 */
export async function validateProject(
  projectId: string,
  clientId: string
): Promise<any> {
  const project = await prisma.project.findUnique({
    where: { id: projectId, clientId: clientId }
  });

  if (!project) {
    throw new Error('PROJECT_NOT_FOUND');
  }

  return project;
}

/**
 * Combined validation for client and project
 * Convenience function for routes that need both
 */
export async function validateClientAndProject(
  session: any,
  projectId: string
): Promise<ValidationResult> {
  const client = await validateClient(session);
  const project = await validateProject(projectId, client.id);

  return { client, project };
}
```

3. AI Content Generation

```
/**
 * Generate content ideas using AI
 * Unified function for all content generation use cases
 */
export async function generateContentIdeas(
  options: GenerateContentIdeasOptions
): Promise<ContentPlanTopic[]> {
  const {
    keywords = [],
    keyword,
    projectContext,
    language = 'NL',
    count = 10,
    temperature = 0.7,
  } = options;

  // Build prompt based on context
  let prompt = '';

  if (keyword) {
    // Single keyword mode (simplified API)
    prompt = buildKeywordPrompt(keyword, projectContext, count);
  } else if (keywords.length > 0) {
    // Multiple keywords mode (client API)
    prompt = buildKeywordsPrompt(keywords, count);
  } else {
    throw new Error('Either keyword or keywords must be provided');
  }

  // Call AI with unified interface
  const aiResponse = await chatCompletion(
    [
      {
        role: 'system',
        content: 'Je bent een SEO expert die gestructureerde JSON content plannen
genereert.'
      },
      { role: 'user', content: prompt }
    ],
    {
      model: 'claude-sonnet-4-20250514',
      temperature,
      max_tokens: 4000,
    }
  );

  // Parse with robust error handling
  const topics = parseAIResponse(aiResponse);

  if (!Array.isArray(topics) || topics.length === 0) {
    throw new Error('NO_TOPICS_GENERATED');
  }

  return topics;
}

/**
 * Build prompt for single keyword
 */
function buildKeywordPrompt(
  keyword: string,
  projectContext?: GenerateContentIdeasOptions['projectContext'],
```

```
  count: number = 10
): string {
  const contextInfo = projectContext
    ? `Context: Dit is voor de website "${projectContext.name}" (${projectCon-
text.websiteUrl || 'geen URL'}) in de niche "${projectContext.niche || 'algemeen'}"`
    : '';

  return `Je bent een expert SEO content strategist. Genereer een uitgebreid content
plan voor het keyword: "${keyword}".

${contextInfo}

Genereer ${count} artikel topics die:
1. Gerelateerd zijn aan het hoofdkeyword
2. Verschillende zoekintents dekken (informationeel, transactioneel, navigational)
3. Long-tail variaties bevatten
4. Content gaps adresseren

Geef je antwoord als een JSON object in dit formaat:
{
  "topics": [
    {
      "title": "Artikel titel",
      "description": "Korte beschrijving van wat het artikel behandelt",
      "keywords": ["keyword1", "keyword2", "keyword3"],
      "priority": "high|medium|low",
      "reason": "Waarom dit topic belangrijk is"
    }
  ]
}

BELANGRIJK: Gebruik alleen "high", "medium", of "low" voor priority.
Geef ALLEEN de JSON terug, geen extra tekst of markdown formatting.`;
}

/**
 * Build prompt for multiple keywords
 */
function buildKeywordsPrompt(
  keywords: string[],
  count: number = 10
): string {
  return `Generate ${count} high-quality content ideas based on these keywords: ${keyw
ords.join(', ')}

For each idea, provide:
- A compelling title
- A brief description (1-2 sentences)
- Focus keyword
- 3-5 secondary keywords
- Search intent (informational/commercial/navigational/transactional)
- Estimated difficulty (0-100)
- Content type (blog-post/how-to/guide/listicle/review/comparison)
- 5-7 H2 outline points

IMPORTANT: Respond with ONLY valid JSON in this exact format:
{
  "topics": [
    {
      "title": "string",
      "description": "string",
      "focusKeyword": "string",
      "secondaryKeywords": ["string"],
```

```
      "searchIntent": "informational|commercial|navigational|transactional",
      "estimatedDifficulty": number,
      "contentType": "blog-post|how-to|guide|listicle|review|comparison",
      "outline": ["string"],
      "priority": number (1-10)
    }
  ]
}

Do not include any markdown formatting, code blocks, or explanations. Return only the
raw JSON object.`;
}

/**
 * Generate content gap analysis for WordPress site
 */
export async function analyzeWordPressContentGaps(
  project: any,
  existingPosts: WordPressPost[]
): Promise<ContentPlanTopic[]> {
  const existingTitles = existingPosts.map(post => post.title.rendered).slice(0, 30);
  const contentSummary = existingTitles.length > 0
    ? `Bestaande artikelen:\n${existingTitles.map((t, i) => `${i + 1}. ${t}`).join('\n
')}`
    : 'Geen bestaande content gevonden of site niet toegankelijk';

  const prompt = `Je bent een expert SEO content strategist. Analyseer de WordPress
website en genereer een content plan.

Website: ${project.name} (${project.websiteUrl})
Niche: ${project.niche || 'Niet gespecificeerd'}

${contentSummary}

Gebaseerd op de bestaande content (of het ontbreken daarvan), genereer 8-12 nieuwe
artikel topics die:
1. Content gaps invullen die nog niet gedekt zijn
2. De niche en doelgroep aanspreken
3. Verschillende zoekintents dekken
4. SEO-vriendelijk zijn met goede zoekvolume potentie
5. Complementair zijn aan bestaande content

BELANGRIJK: Geef je antwoord ALLEEN als een geldig JSON object, zonder extra tekst,
uitleg of markdown formatting.

Exact formaat (volg dit exact):
{
  "topics": [
    {
      "title": "Artikel titel",
      "description": "Korte beschrijving van het artikel",
      "keywords": ["keyword1", "keyword2", "keyword3"],
      "priority": "high",
      "reason": "Waarom dit topic een content gap invult"
    }
  ]
}

Gebruik alleen deze priority waarden: "high", "medium", of "low".
Geef minimaal 8 en maximaal 12 topics.
Antwoord direct met de JSON, geen tekst ervoor of erna.`;

  const aiResponse = await chatCompletion(
```

```
    [
      {
        role: 'system',
        content: 'Je bent een SEO expert die WordPress sites analyseert en gestructur-
eerde JSON content plannen genereert.'
      },
      { role: 'user', content: prompt }
    ],
    {
      model: 'claude-sonnet-4-20250514',
      temperature: 0.7,
      max_tokens: 4000,
    }
  );

  return parseAIResponse(aiResponse);
}
```

4. JSON Parsing (Robust)

```
/**
 * Parse AI response with 4 fallback strategies
 * Based on analyze-wordpress implementation (most robust)
 */
export function parseAIResponse(aiResponse: string): ContentPlanTopic[] {
  let topics: ContentPlanTopic[] = [];

  try {
    // Strategy 1: Try direct JSON parse
    try {
      const parsed = JSON.parse(aiResponse);
      topics = parsed.topics || parsed.ideas || parsed.contentIdeas || [];
      console.log('[content-plan-service] Strategy 1 success: Direct JSON parse');
      return topics;
    } catch (e1) {
      // Strategy 2: Remove markdown code blocks
      try {
        const withoutCodeBlocks = aiResponse
          .replace(/```json\s*/gi, '')
          .replace(/```\s*/g, '')
          .trim();
        const parsed = JSON.parse(withoutCodeBlocks);
        topics = parsed.topics || parsed.ideas || parsed.contentIdeas || [];
        console.log('[content-plan-service] Strategy 2 success: Removed markdown code
blocks');
        return topics;
      } catch (e2) {
        // Strategy 3: Extract JSON from text using regex
        try {
          const jsonMatch = aiResponse.match(/\{[\s\S]*"topics"[\s\S]*\}/);
          if (!jsonMatch) {
            // Try alternative keys
            const altMatch = aiResponse.match(/\{[\s\S]*"(ideas|conten-
tIdeas)"[\s\S]*\}/);
            if (altMatch) {
              const parsed = JSON.parse(altMatch[0]);
              topics = parsed.ideas || parsed.contentIdeas || [];
              console.log('[content-plan-service] Strategy 3 success: Regex JSON
extraction (alt key)');
              return topics;
            }
            throw new Error('No JSON object found in response');
          }
          const parsed = JSON.parse(jsonMatch[0]);
          topics = parsed.topics || parsed.ideas || parsed.contentIdeas || [];
          console.log('[content-plan-service] Strategy 3 success: Regex JSON extrac-
tion');
          return topics;
        } catch (e3) {
          // Strategy 4: Try to find array directly
          try {
            const topicsMatch = aiResponse.match(/"(topics|ideas|contentIdeas)"\s*:
\s*(\[[\s\S]*?\])\s*\}/);
            if (!topicsMatch) {
              throw new Error('No topics array found in response');
            }
            topics = JSON.parse(topicsMatch[2]);
            console.log('[content-plan-service] Strategy 4 success: Direct topics
array extraction');
            return topics;
          } catch (e4) {
            // All strategies failed - log for debugging
```

```
            console.error('[content-plan-service] All parsing strategies failed');
            console.error('[content-plan-service] Raw AI response length:', aiRe-
sponse.length);
            console.error('[content-plan-service] Raw AI response (first 1000
chars):', aiResponse.substring(0, 1000));
            console.error('[content-plan-service] Raw AI response (last 500 chars):',
aiResponse.substring(Math.max(0, aiResponse.length - 500)));

            throw new Error('Failed to parse AI response after trying all strategies')
;
          }
        }
      }
    }
  } catch (parseError: any) {
    console.error('[content-plan-service] Complete parsing failure');
    console.error('[content-plan-service] Parse error:', parseError);
    throw new Error(`AI_PARSE_ERROR: ${parseError.message}`);
  }
}
```

5. Database Operations

```
/**
 * Generate slug from title
 */
export function generateSlug(title: string): string {
  return title.toLowerCase()
    .replace(/[^a-z0-9]+/g, '-')
    .replace(/^-|-$/g, '');
}

/**
 * Save article ideas to database
 * Handles both upsert and insert operations
 */
export async function saveArticleIdeas(
  topics: ContentPlanTopic[] | any[],
  clientId: string,
  projectId: string | null,
  options: {
    targetKeyword?: string;
    useUpsert?: boolean;
  } = {}
): Promise<any[]> {
  const { targetKeyword, useUpsert = true } = options;
  const savedIdeas = [];

  for (let i = 0; i < topics.length; i++) {
    const topic = topics[i];

    try {
      // Normalize topic data (handle both ContentPlanTopic and detailed formats)
      const ideaData = normalizeTopicData(topic, clientId, projectId, targetKeyword, i
);

      let saved;
      if (useUpsert) {
        saved = await prisma.articleIdea.upsert({
          where: {
            clientId_slug: {
              clientId: clientId,
              slug: ideaData.slug,
            }
          },
          update: {
            secondaryKeywords: ideaData.secondaryKeywords,
            contentOutline: ideaData.contentOutline,
            aiScore: ideaData.aiScore,
            trending: ideaData.trending,
            competitorGap: ideaData.competitorGap,
            keywords: ideaData.keywords,
            priority: ideaData.priority,
            description: ideaData.description,
          },
          create: ideaData,
        });
      } else {
        saved = await prisma.articleIdea.create({
          data: ideaData,
        });
      }

      savedIdeas.push(saved);
    } catch (error) {
```

```
      console.error('[content-plan-service] Error saving idea:', error);
      // Continue with other ideas
    }
  }

  return savedIdeas;
}

/**
 * Normalize topic data to ArticleIdeaData format
 */
function normalizeTopicData(
  topic: any,
  clientId: string,
  projectId: string | null,
  targetKeyword?: string,
  index: number = 0
): any {
  const slug = generateSlug(topic.title);

  // Handle different input formats
  const isDetailedFormat = 'focusKeyword' in topic || 'outline' in topic;

  if (isDetailedFormat) {
    // Client API format (from add-ideas)
    return {
      clientId,
      projectId,
      title: topic.title,
      slug,
      focusKeyword: topic.focusKeyword || topic.keywords?.[0] || '',
      topic: topic.description || topic.title,
      secondaryKeywords: topic.secondaryKeywords || [],
      searchIntent: topic.searchIntent || 'informational',
      difficulty: topic.estimatedDifficulty || 50,
      contentOutline: {
        sections: (topic.outline || []).map((h2: string) => ({
          heading: h2,
          subpoints: []
        }))
      },
      contentType: topic.contentType || 'blog-post',
      priority: typeof topic.priority === 'number' ? topic.priority : 5,
      aiScore: typeof topic.priority === 'number' ? topic.priority * 10 : 70,
      status: 'idea',
      trending: topic.trending,
      competitorGap: topic.competitorGap,
    };
  } else {
    // Simplified API format
    return {
      clientId,
      projectId,
      title: topic.title,
      slug,
      description: topic.description,
      keywords: topic.keywords || [],
      priority: topic.priority || 'medium',
      reason: topic.reason,
      targetKeyword: targetKeyword || topic.keywords?.[0],
      aiScore: 1.0 - (index * 0.05),
      searchVolume: 0,
      focusKeyword: topic.keywords?.[0] || '',
```

```
      searchIntent: 'informational',
      difficulty: 50,
      contentType: 'blog-post',
      status: 'idea',
    };
  }
}

/**
 * Get article ideas for a project
 */
export async function getArticleIdeas(
  clientId: string,
  projectId?: string,
  options: {
    limit?: number;
    orderBy?: any;
  } = {}
): Promise<any[]> {
  const { limit, orderBy } = options;

  const where: any = { clientId };
  if (projectId) {
    where.projectId = projectId;
  }

  return prisma.articleIdea.findMany({
    where,
    include: {
      savedContent: {
        select: {
          id: true,
          publishedUrl: true,
          publishedAt: true,
        }
      }
    },
    orderBy: orderBy || [
      { priority: 'desc' },
      { aiScore: 'desc' },
      { createdAt: 'desc' }
    ],
    take: limit,
  });
}
```

6. WordPress Integration

```
/**
 * Fetch WordPress posts from site
 */
export async function fetchWordPressPosts(
  websiteUrl: string,
  options: {
    perPage?: number;
    timeout?: number;
  } = {}
): Promise<WordPressPost[]> {
  const { perPage = 50, timeout = 10000 } = options;

  try {
    const wpApiUrl = `${websiteUrl}/wp-json/wp/v2/posts?per_page=${perPage}&_fields=id
,title,excerpt,categories,tags`;

    const response = await fetch(wpApiUrl, {
      headers: {
        'User-Agent': 'WritGoAI Content Planner/1.0',
      },
      signal: AbortSignal.timeout(timeout),
    });

    if (!response.ok) {
      console.warn(`[content-plan-service] Failed to fetch WordPress posts: ${re-
sponse.status}`);
      return [];
    }

    const posts = await response.json();
    console.log(`[content-plan-service] Fetched ${posts.length} WordPress posts`);
    return posts;

  } catch (error: any) {
    console.warn(`[content-plan-service] Error fetching WordPress content:`, er-
ror.message);
    return [];
  }
}
```

7. Error Handling Helper

```
/**
 * Map service errors to HTTP responses
 */
export function mapServiceError(error: any): {
  status: number;
  error: string;
  message: string;
  details?: string;
} {
  const errorMessage = error.message || 'Unknown error';

  // Map known errors
  const errorMap: Record<string, { status: number; error: string; message: string }>
= {
    'UNAUTHORIZED': {
      status: 401,
      error: 'Unauthorized',
      message: 'Je moet ingelogd zijn'
    },
    'CLIENT_NOT_FOUND': {
      status: 404,
      error: 'Client not found',
      message: 'Gebruiker niet gevonden'
    },
    'PROJECT_NOT_FOUND': {
      status: 404,
      error: 'Project not found',
      message: 'Project niet gevonden'
    },
    'NO_TOPICS_GENERATED': {
      status: 500,
      error: 'No topics generated',
      message: 'Geen topics gegenereerd. Probeer het opnieuw.'
    },
  };

  // Check for specific error types
  for (const [key, response] of Object.entries(errorMap)) {
    if (errorMessage.includes(key)) {
      return { ...response, details: errorMessage };
    }
  }

  // Check for AI parse errors
  if (errorMessage.includes('AI_PARSE_ERROR')) {
    return {
      status: 500,
      error: 'Failed to parse AI response',
      message: 'Kan AI response niet parsen',
      details: errorMessage
    };
  }

  // Default error response
  return {
    status: 500,
    error: 'Internal server error',
    message: 'Er is een fout opgetreden',
    details: errorMessage
  };
}
```

Refactor Strategie

Stap 1: Implementeer Service Layer

1. Creëer `lib/services/content-plan-service.ts`
2. Implementeer alle functies zoals ontworpen
3. Voeg unit tests toe (optioneel)

Stap 2: Refactor Client Routes

Voor elke route:
1. Import shared service functies
2. Vervang duplicate code met service calls
3. Behoud route-specifieke logic (bijv. `refreshDailyInsights` in refresh route)
4. Test endpoint

Volgorde:
1. `route.ts` (GET) - Eenvoudigst
2. `add-ideas/route.ts` (POST) - Medium
3. `refresh/route.ts` (POST) - Bevat unieke logic

Stap 3: Refactor Simplified Routes

Voor elke route:
1. Import shared service functies
2. Vervang duplicate code met service calls
3. Test endpoint

Volgorde:
1. `route.ts` (GET) - Heeft eigen grouping logic
2. `route.ts` (POST) - Standaard refactor
3. `analyze-wordpress/route.ts` (POST) - Heeft WordPress fetching

Stap 4: Testing

1. Test alle 5 routes handmatig
2. Verifieer responses matchen origineel
3. Check error handling
4. Build test: `npm run build`

Stap 5: Cleanup

1. Verwijder commented oude code
2. Update imports consistent (gebruik `@/lib/ai-utils` overal)
3. Add JSDoc comments waar nodig

Backwards Compatibility Checklist

✅ **Response Formats**
- Client routes: Behouden huidige response structure
- Simplified routes: Behouden huidige response structure

✅ **Request Parameters**
- Geen veranderingen in query params of body structure

✅ **Error Responses**
- Behoud status codes
- Behoud error message formats

✅ **API Endpoints**
- Geen endpoint URL veranderingen
- Beide `/api/client/*` en `/api/simplified/*` blijven bestaan

Testing Plan

Unit Tests (Optioneel)

```
// test/services/content-plan-service.test.ts
describe('ContentPlanService', () => {
  describe('generateSlug', () => {
    it('should generate valid slug from title', () => {
      expect(generateSlug('Hello World!')).toBe('hello-world');
    });
  });

  describe('parseAIResponse', () => {
    it('should parse direct JSON', () => {
      const response = '{"topics": [{"title": "Test"}]}';
      const result = parseAIResponse(response);
      expect(result).toHaveLength(1);
    });

    it('should handle markdown code blocks', () => {
      const response = '```json\n{"topics": [{"title": "Test"}]}\n```';
      const result = parseAIResponse(response);
      expect(result).toHaveLength(1);
    });
  });
});
```

Integration Tests

1. **Client GET**: Verifieer data retrieval
2. **Client POST (add-ideas)**: Genereer ideeën met keywords
3. **Client POST (refresh)**: Refresh bestaand plan
4. **Simplified GET**: Verifieer grouping logic
5. **Simplified POST**: Genereer plan met keyword
6. **Simplified POST (analyze)**: WordPress analyse

Build Test

```
cd /home/ubuntu/writgoai_nl/nextjs_space
npm run build
```

Success Criteria

✅ **Code Quality**
- [ ] Geen duplicate authenticatie code
- [ ] Geen duplicate AI generation code

- [ ] Geen duplicate database operations
- [ ] Consistente error handling

✅ **Functionality**
- [ ] Alle 5 routes werken correct
- [ ] Response formats onveranderd
- [ ] Error responses consistent
- [ ] Build succesvol

✅ **Documentation**
- [ ] JSDoc comments toegevoegd
- [ ] FASE3_RAPPORT.md compleet
- [ ] Code comments voor complexe logic

✅ **Git**
- [ ] Clean commit history
- [ ] Descriptive commit messages
- [ ] Pushed naar GitHub

Metrics Targets

- **Code Reductie**: >60% duplicate code geëlimineerd
- **Build Time**: Geen significante increase
- **Response Time**: Geen performance degradation
- **Test Coverage**: 100% manual testing van alle endpoints

Volgende Stap

Proceed naar implementatie van `lib/services/content-plan-service.ts`