Fase 3: Content-Plan Routes Analyse

Datum: 16 december 2025

Overzicht

Analyse van duplicate functionaliteit tussen `/api/client/content-plan/*` en `/api/simplified/content-plan/*` routes.

Routes Inventaris

Client Content-Plan Routes

1. **`/api/client/content-plan/route.ts`** (GET)
   - Functie: Haalt bestaande content plan en article ideas op voor een specifiek project
   - Parameters: `projectId`
   - Response: Project details + article ideas met priority/aiScore sorting

2. **`/api/client/content-plan/refresh/route.ts`** (POST)
   - Functie: Refresht bestaand plan met nieuwe AI-gegenereerde ideeën
   - Parameters: `projectId`, `language`
   - Dependencies: `@/lib/intelligent-content-planner`
   - Gebruikt: `refreshDailyInsights()` functie

3. **`/api/client/content-plan/add-ideas/route.ts`** (POST)
   - Functie: Voegt nieuwe content ideeën toe op basis van keywords
   - Parameters: `keywords[]`, `projectId`, `language`
   - AI Model: Claude Sonnet via `@/lib/aiml-api`
   - Output: 5-10 content ideeën met outline

Simplified Content-Plan Routes

1. **`/api/simplified/content-plan/route.ts`** (GET + POST)
   - **GET**: Haalt alle content plans op, gegroepeerd per project/keyword
   - **POST**: Genereert nieuw content plan op basis van keyword
   - Parameters: `projectId` (optional), `keyword`
   - AI Model: Claude Sonnet via `@/lib/ai-utils`
   - Output: 8-12 artikel topics

2. **`/api/simplified/content-plan/analyze-wordpress/route.ts`** (POST)
   - Functie: Analyseert WordPress site en genereert content gap analyse
   - Parameters: `projectId`
   - Extra: Haalt bestaande WordPress posts op via WP REST API
   - AI Model: Claude Sonnet via `@/lib/ai-utils`
   - Output: 8-12 content gap topics

Overlap & Duplicate Functionaliteit

🔴 Kritieke Duplicatie

1. Client Authenticatie Patroon

Duplicate code in ALLE 5 routes:

```
const session = await getServerSession(authOptions);
if (!session?.user?.email) {
  return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
}

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

if (!client) {
  return NextResponse.json({ error: 'Client not found' }, { status: 404 });
}
```

Impact: ~20 regels code duplicatie per route = ~100 regels totaal

2. Project Validatie Patroon

Duplicate code in 4 van 5 routes:

```
const project = await prisma.project.findUnique({
  where: { id: projectId, clientId: client.id }
});

if (!project) {
  return NextResponse.json({ error: 'Project not found' }, { status: 404 });
}
```

Impact: ~10 regels code duplicatie per route = ~40 regels totaal

3. AI Content Generatie

Vergelijkbare functionaliteit in 3 routes:
- `client/content-plan/add-ideas`
- `simplified/content-plan` (POST)
- `simplified/content-plan/analyze-wordpress`

Overlap:
- Alle gebruiken `chatCompletion` met Claude Sonnet
- Vergelijkbare prompts voor content idee generatie
- Identieke output structuur (title, description, keywords, priority)
- Zelfde temperature (0.7-0.8) en token limits

Verschil:
- `client/add-ideas` gebruikt `@/lib/aiml-api`
- `simplified` routes gebruiken `@/lib/ai-utils`

Impact: ~50 regels prompt logic duplicatie per route = ~150 regels

4. JSON Parsing van AI Responses

Duplicate patroon in alle AI-gebruikende routes:

```
let responseContent = aiResponse.trim()
  .replace(/```json\n?/g, '')
  .replace(/```\n?/g, '')
  .trim();

const result = JSON.parse(responseContent);
```

Note: `analyze-wordpress` heeft meest robuuste implementatie met 4 fallback strategieën

Impact: ~15 regels per route = ~60 regels totaal

5. Database ArticleIdea Upsert/Insert

Vergelijkbare logic in 4 routes:

```
await prisma.articleIdea.upsert({
  where: {
    clientId_slug: {
      clientId: client.id,
      slug: slug,
    }
  },
  update: { /* fields */ },
  create: { /* full data */ }
});
```

Impact: ~30-50 regels per route = ~150 regels totaal

6. Slug Generatie

Identiek in meerdere routes:

```
const slug = title.toLowerCase()
  .replace(/[^a-z0-9]+/g, '-')
  .replace(/^-|-$/g, '');
```

Impact: ~3 regels per route = ~12 regels totaal

🟡 Medium Duplicatie

7. Error Handling Patroon

Alle routes hebben vergelijkbaar try-catch patroon met:
- Console logging
- Structured error responses
- Status code 500
- Dutch error messages (simplified routes)

8. Request Validation

Vergelijkbare input validatie voor:
- `projectId` required checks
- `keyword` validation
- Type checking

🟢 Unieke Functionaliteit (Geen Duplicatie)

1. WordPress Posts Fetching (`analyze-wordpress`)

```
const wpApiUrl = `${project.websiteUrl}/wp-json/wp/v2/posts`;
const response = await fetch(wpApiUrl, { /* config */ });
```

Unieke feature voor WordPress content gap analyse.

2. Intelligent Content Planner Integration (`refresh`)

```
const newIdeas = await refreshDailyInsights(
  existingPlan,
  niche,
  targetAudience
);
```

Gebruikt specifieke library voor plan refresh.

Consolidatie Opportuniteiten

High Priority (Meeste Impact)

1. **Shared Authentication Service**
   - Consolideer client authenticatie en validatie
   - Potentiele besparing: ~100 regels

2. **Shared AI Content Generator**
   - Unified content idee generatie functie
   - Configureerbare prompts per use case
   - Potentiele besparing: ~150 regels

3. **Shared Database Service**
   - ArticleIdea CRUD operations
   - Slug generatie utility
   - Potentiele besparing: ~150 regels

4. **Robust JSON Parser**
   - Gebruik analyze-wordpress's 4-strategy parser voor alle routes
   - Potentiele besparing: ~60 regels + verhoogde reliability

Medium Priority

1. **Shared Project Validator**
   - Project ownership validation
   - Potentiele besparing: ~40 regels

2. **Shared WordPress Client**
   - WordPress API integration
   - Herbruikbaar voor toekomstige features

Low Priority

1. **Shared Error Handler**
   - Consistent error formatting
   - Structured logging

Architectuur Overwegingen

API Import Consolidatie

Probleem: Twee verschillende imports voor chatCompletion:
- `@/lib/aiml-api` (client routes)
- `@/lib/ai-utils` (simplified routes)

Oplossing: Gebruik consistent `@/lib/ai-utils` (nieuwere versie volgens summaries)

Backwards Compatibility

- **Kritisch**: Beide endpoint sets (`/api/client/*` en `/api/simplified/*`) moeten blijven werken
- Response formats mogen niet breken
- Bestaande frontend code is afhankelijk van huidige structuur

Service Layer Design

Voorgestelde structuur:

```
lib/services/content-plan-service.ts
├── Authentication & Validation
│   ├── validateClient()
│   ├── validateProject()
│   └── validateClientProject()
├── Content Generation
│   ├── generateContentIdeas()
│   ├── generateContentPlan()
│   └── analyzeContentGaps()
├── Database Operations
│   ├── saveArticleIdeas()
│   ├── getArticleIdeas()
│   └── upsertArticleIdea()
├── Utilities
│   ├── parseAIResponse()
│   ├── generateSlug()
│   └── fetchWordPressPosts()
└── Types & Interfaces
    ├── ContentPlanTopic
    ├── ArticleIdeaData
    └── GenerateOptions
```

Metrics

Code Duplicatie

- **Totale duplicate regels**: ~562 regels
- **Aantal routes**: 5
- **Gemiddelde duplicatie per route**: ~112 regels

Consolidatie Potentie

- **Geschatte reductie**: 60-70% van duplicate code
- **Nieuwe shared service**: ~400 regels
- **Totale besparing**: ~200 regels netto
- **Maintainability verbetering**: Significant (DRY principe)

Risico Assessment

- **Breaking changes risico**: 🟡 Medium (met goede tests: 🟢 Low)
- **Testing effort**: 🟡 Medium
- **Migration complexity**: 🟢 Low (incremental refactor mogelijk)

Volgende Stappen

1. ✅ Analyse compleet
2. ⏭️ Ontwerp shared service layer architectuur
3. ⏭️ Implementeer service layer
4. ⏭️ Refactor client routes
5. ⏭️ Refactor simplified routes
6. ⏭️ Test en valideer
7. ⏭️ Documenteer en deploy

Conclusie

Er is **significante code duplicatie** tussen client en simplified content-plan routes. Door een shared service layer te implementeren kunnen we:
- ✅ ~60-70% code duplicatie elimineren
- ✅ Consistency tussen endpoints verbeteren
- ✅ Maintenance burden verminderen
- ✅ Robuustere error handling implementeren (analyze-wordpress parser voor allen)
- ✅ Toekomstige features makkelijker toevoegen

Aanbeveling: Proceed met consolidatie, behoud beide endpoint sets voor backwards compatibility.