WritGo.nl Multi-Project Architecture - Developer Documentation

📐 Architecture Overview

This document describes the technical implementation of the multi-project functionality in WritGo.nl.

Table of Contents

1. Database Schema
2. API Endpoints
3. State Management
4. Components
5. Data Isolation
6. Migration Guide
7. Best Practices

Database Schema

Project Table

```
CREATE TABLE "Project" (
  id TEXT PRIMARY KEY,
  clientId TEXT NOT NULL,
  name TEXT NOT NULL,
  websiteUrl TEXT,
  description TEXT,
  status TEXT DEFAULT 'active',
  settings JSONB DEFAULT '{}'::jsonb,
  createdAt TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT "Project_clientId_fkey"
    FOREIGN KEY (clientId) REFERENCES "Client"(id) ON DELETE CASCADE
);

CREATE INDEX "Project_clientId_idx" ON "Project"("clientId");
CREATE INDEX "Project_status_idx" ON "Project"("status");
```

Content Tables with projectId

All content tables have been updated to include `projectId`:

- **BlogPost**
- **ContentPlan**
- **TopicalAuthorityMap**

- **SocialMediaStrategy**
- **WebsiteAnalysis**
- **AutopilotConfig**

```
-- Example: BlogPost with projectId
ALTER TABLE "BlogPost" ADD COLUMN "projectId" TEXT;
ALTER TABLE "BlogPost" ADD CONSTRAINT "BlogPost_projectId_fkey"
  FOREIGN KEY ("projectId") REFERENCES "Project"(id) ON DELETE CASCADE;
CREATE INDEX "BlogPost_projectId_idx" ON "BlogPost"("projectId");
```

RLS Policies

Row Level Security policies ensure data isolation:

```
-- Clients can only view blog posts from their own projects
CREATE POLICY "Clients can view blog posts from their projects"
  ON "BlogPost" FOR SELECT
  USING (
    EXISTS (
      SELECT 1 FROM "Project"
      WHERE "Project".id = "BlogPost"."projectId"
      AND "Project"."clientId" = auth.uid()::text
    )
  );
```

API Endpoints

Project Management

GET /api/admin/projects

Fetch all projects for the current user's client.

```
// Request
GET /api/admin/projects

// Response
[
  {
    id: "proj_123",
    clientId: "client_456",
    name: "WritGo.nl",
    websiteUrl: "https://writgo.nl",
    description: "Marketing platform",
    status: "active",
    createdAt: "2025-12-12T10:00:00Z",
    updatedAt: "2025-12-12T10:00:00Z"
  }
]
```

POST /api/admin/projects

Create a new project.

```
// Request
POST /api/admin/projects
Content-Type: application/json

{
  "name": "My New Website",
  "websiteUrl": "https://example.com",
  "description": "Optional description"
}

// Response (201 Created)
{
  "id": "proj_789",
  "clientId": "client_456",
  "name": "My New Website",
  "websiteUrl": "https://example.com",
  "description": "Optional description",
  "status": "active",
  "createdAt": "2025-12-12T11:00:00Z",
  "updatedAt": "2025-12-12T11:00:00Z"
}
```

GET /api/admin/projects/[id]

Fetch a specific project.

```
// Request
GET /api/admin/projects/proj_123

// Response
{
  "id": "proj_123",
  "clientId": "client_456",
  "name": "WritGo.nl",
  // ... other fields
}
```

PUT /api/admin/projects/[id]

Update a project.

```
// Request
PUT /api/admin/projects/proj_123
Content-Type: application/json

{
  "name": "Updated Name",
  "status": "inactive"
}

// Response
{
  "id": "proj_123",
  "name": "Updated Name",
  "status": "inactive",
  // ... other fields
}
```

DELETE /api/admin/projects/[id]

Delete a project and all associated content.

```
// Request
DELETE /api/admin/projects/proj_123

// Response
{
  "success": true
}
```

Content APIs with projectId

All content APIs now support `projectId` query parameter:

```
// Example: Blog Posts API
GET /api/admin/blog?projectId=proj_123
POST /api/admin/blog { ..., projectId: "proj_123" }

// Example: Social Media API
GET /api/admin/social?projectId=proj_123
POST /api/admin/social { ..., projectId: "proj_123" }
```

State Management

ProjectContext

Global state management for projects using React Context.

```
// lib/contexts/ProjectContext.tsx

interface ProjectContextType {
  currentProject: Project | null;
  projects: Project[];
  loading: boolean;
  error: string | null;

  // Actions
  switchProject: (projectId: string) => Promise<void>;
  addProject: (data: ProjectData) => Promise<Project | null>;
  updateProject: (id: string, updates: Partial<Project>) => Promise<boolean>;
  deleteProject: (id: string) => Promise<boolean>;
  refreshProjects: () => Promise<void>;
  ensureDefaultProject: () => Promise<void>;
}
```

Usage in Components

```
'use client';

import { useProject } from '@/lib/contexts/ProjectContext';

export default function MyComponent() {
  const { currentProject, switchProject, loading } = useProject();

  if (loading) return <div>Loading...</div>;
  if (!currentProject) return <div>No project selected</div>;

  return (
    <div>
      <h1>Current Project: {currentProject.name}</h1>
      {/* ... */}
    </div>
  );
}
```

Custom Hooks

useProjectFetch

Automatically adds `projectId` to fetch requests:

```
import { useProjectFetch } from '@/lib/hooks/useProjectFetch';

export default function BlogList() {
  const projectFetch = useProjectFetch();

  const fetchPosts = async () => {
    // Automatically adds ?projectId=xyz
    const response = await projectFetch('/api/admin/blog');
    const data = await response.json();
    return data.posts;
  };
}
```

useCurrentProjectId

Get the current project ID:

```
import { useCurrentProjectId } from '@/lib/hooks/useProjectFetch';

export default function MyComponent() {
  const projectId = useCurrentProjectId();

  // Use projectId in API calls
}
```

Project Switch Event

Listen for project switches:

```tsx
import { useProjectSwitch } from '@/lib/contexts/ProjectContext';

useProjectSwitch((project) => {
  console.log('Switched to project:', project.name);
  // Refresh data, clear caches, etc.
});
```

Components

ProjectSwitcher

Dropdown component for switching between projects.

```tsx
// components/project/ProjectSwitcher.tsx
import ProjectSwitcher from '@/components/project/ProjectSwitcher';

<ProjectSwitcher />
```

Features:
- Displays current project with icon
- Lists all available projects
- “Nieuw Project” quick action
- “Projecten Beheren” link
- Auto-close on outside click
- Dispatches `project-changed` event

Location: Sidebar header (desktop), mobile navigation

FirstProjectModal

Onboarding modal for new users.

```tsx
// components/onboarding/FirstProjectModal.tsx
import FirstProjectModal from '@/components/onboarding/FirstProjectModal';

<FirstProjectModal />
```

Features:
- Shows only when `projects.length === 0`
- Cannot be closed until project is created
- Form validation
- Auto-creates first project
- Auto-switches to new project

Location: AdminComplexLayout (always rendered)

Project Management Page

Full CRUD interface for projects.

```tsx
// app/admin/projects/page.tsx
```

Features:
- Grid layout of project cards
- “Nieuw Project” button
- Empty state for no projects
- Project cards with:
- Name, URL, description
- Status badge (Actief/Inactief)
- Actions: Open, Delete
- AddProjectDialog embedded

Data Isolation

Per-Project Filtering

All content queries must include `projectId`:

```
// ❌ BAD: No project filtering
const posts = await prisma.blogPost.findMany({
  where: { status: 'published' }
});

// ✅ GOOD: With project filtering
const posts = await prisma.blogPost.findMany({
  where: {
    status: 'published',
    projectId: currentProject.id
  }
});
```

API Route Pattern

```
// app/api/admin/blog/route.ts
export async function GET(request: Request) {
  const { searchParams } = new URL(request.url);
  const projectId = searchParams.get('projectId');

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

  const posts = await prisma.blogPost.findMany({ where });
  return NextResponse.json({ posts });
}
```

RLS Enforcement

Database-level security ensures isolation:

```
-- Even if API forgets to filter, RLS catches it
CREATE POLICY "project_isolation"
  ON "BlogPost" FOR ALL
  USING (
    EXISTS (
      SELECT 1 FROM "Project"
      WHERE "Project".id = "BlogPost"."projectId"
      AND "Project"."clientId" = auth.uid()::text
    )
  );
```

Migration Guide

Updating Existing Code

1. Add ProjectContext to Layout

```
// app/admin/layout.tsx
import { ProjectProvider } from '@/lib/contexts/ProjectContext';

export default function AdminLayout({ children }) {
  return (
    <ProjectProvider>
      {children}
    </ProjectProvider>
  );
}
```

2. Update API Routes

```
// Before
const posts = await prisma.blogPost.findMany();

// After
const projectId = searchParams.get('projectId');
const where: any = {};
if (projectId) where.projectId = projectId;
const posts = await prisma.blogPost.findMany({ where });
```

3. Update UI Components

```
// Before
const fetchData = async () => {
  const res = await fetch('/api/admin/blog');
  const data = await res.json();
  return data;
};

// After
import { useProjectFetch } from '@/lib/hooks/useProjectFetch';

const projectFetch = useProjectFetch();
const fetchData = async () => {
  const res = await projectFetch('/api/admin/blog');
  const data = await res.json();
  return data;
};
```

4. Handle No Project State

```
import { useProject } from '@/lib/contexts/ProjectContext';

const { currentProject, loading } = useProject();

if (loading) return <Loader />;
if (!currentProject) return <NoProjectWarning />;

// Render content
```

Best Practices

1. Always Check Current Project

```
const { currentProject } = useProject();

if (!currentProject) {
  return <div>Selecteer eerst een project</div>;
}
```

2. Use Custom Hooks

```
// ✅ GOOD: Use utility hooks
const projectFetch = useProjectFetch();
const projectId = useCurrentProjectId();

// ❌ BAD: Manual projectId management
const projectId = currentProject?.id;
```

3. Handle Project Switches

```
useProjectSwitch(() => {
  // Refresh data when project changes
  fetchData();
});
```

4. Validate projectId in APIs

```
export async function POST(request: Request) {
  const { projectId } = await request.json();

  if (!projectId) {
    return NextResponse.json(
      { error: 'projectId is required' },
      { status: 400 }
    );
  }

  // Continue...
}
```

5. Use Cascade Deletes

```
-- Foreign keys with CASCADE
ALTER TABLE "BlogPost" ADD CONSTRAINT "BlogPost_projectId_fkey"
  FOREIGN KEY ("projectId") REFERENCES "Project"(id)
  ON DELETE CASCADE;
```

6. Index projectId Columns

```
-- Performance optimization
CREATE INDEX "BlogPost_projectId_idx"
  ON "BlogPost"("projectId");
```

7. Document API Changes

```
/**
 * GET /api/admin/blog
 *
 * Query Parameters:
 * - projectId (string): Filter by project ID
 * - status (string): Filter by post status
 *
 * Returns: Array of blog posts for the specified project
 */
```

Error Handling

No Project Selected

```
if (!currentProject) {
  throw new Error('No project selected');
}
```

Invalid Project ID

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

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

Permission Check

```
if (project.clientId !== client.id) {
  return NextResponse.json(
    { error: 'Forbidden' },
    { status: 403 }
  );
}
```

Testing

Unit Tests

```
describe('ProjectContext', () => {
  it('should switch projects', async () => {
    const { result } = renderHook(() => useProject());
    await act(async () => {
      await result.current.switchProject('proj_123');
    });
    expect(result.current.currentProject?.id).toBe('proj_123');
  });
});
```

Integration Tests

```
describe('Blog API with Projects', () => {
  it('should filter posts by projectId', async () => {
    const response = await fetch('/api/admin/blog?projectId=proj_123');
    const data = await response.json();

    expect(data.posts.every(p => p.projectId === 'proj_123')).toBe(true);
  });
});
```

Performance Considerations

1. Cache Project List

```
// ProjectContext caches projects in state
// No need to refetch on every render
```

2. Lazy Load Project Data

```
// Only fetch content when project is selected
useEffect(() => {
  if (currentProject) {
    fetchProjectData(currentProject.id);
  }
}, [currentProject]);
```

3. Database Indexes

```
-- All projectId columns are indexed
CREATE INDEX idx ON "BlogPost"("projectId");
```

4. Debounce Project Switches

```
// Built into ProjectContext
// Prevents rapid switching issues
```

Deployment Checklist

- [ ] Run database migration: `20251212_multi_project_support.sql`
- [ ] Verify all indexes are created
- [ ] Test RLS policies in production
- [ ] Verify ProjectProvider wraps AdminLayout
- [ ] Test onboarding flow for new users
- [ ] Test project switching across all pages
- [ ] Verify data isolation between projects

- [ ] Check cascade deletes work correctly
- [ ] Test API endpoints with/without projectId
- [ ] Verify mobile navigation includes ProjectSwitcher

Troubleshooting

Projects Not Loading

```
// Check browser console for errors
// Verify API endpoint returns 200
// Check session/authentication

console.log('Current project:', currentProject);
console.log('All projects:', projects);
console.log('Loading state:', loading);
```

Data Not Filtering by Project

```
// Verify projectId is in API call
console.log('Fetching with projectId:', currentProject?.id);

// Check API response
const response = await fetch(`/api/admin/blog?projectId=${currentProject.id}`);
console.log('Response data:', await response.json());
```

Project Switch Not Working

```
// Listen for project-changed event
window.addEventListener('project-changed', (e) => {
  console.log('Project changed to:', e.detail.projectId);
});
```

Version History

- **v2.0** (December 2025) - Initial multi-project implementation
- Database schema updated
- API endpoints created
- UI components implemented
- Documentation written

WritGo.nl - Developer Documentation
Last Updated: December 12, 2025