Phase 16: Site Settings CMS - Implementation Summary

📋 Overview

Phase 16 introduces a comprehensive **Site Settings CMS** that allows administrators to manage site-wide configurations without touching code. This system provides a centralized interface for managing logos, contact information, social media links, SEO metadata, appearance settings, and footer content.

✅ Completed Tasks

1. Database Schema

- **New Model**: `SiteSetting`
- Flexible key-value structure for storing settings
- Organized by categories (general, contact, social, seo, appearance, footer)
- Support for multiple input types (text, textarea, url, email, image, json, boolean, color)
- Bilingual labels and help text for admin UI
- Order field for display sequence

2. Helper Library (`lib/site-settings.ts`)

- Comprehensive helper functions with caching:
- `getAllSettings()` - Fetch all settings
- `getSettingsByCategory(category)` - Category-specific settings
- `getSetting(key)` - Individual setting value
- `getSettingWithDefault(key, default)` - With fallback
- `getBooleanSetting(key)` - Typed boolean getter
- `getJsonSetting(key)` - Parse JSON settings
- `getSettingsMap()` - Key-value map for easy access
- Specialized getters:
- `getSiteInfo()` - Site name, logo, description
- `getContactInfo()` - Contact details
- `getSocialLinks()` - Social media URLs
- `getFooterContent()` - Footer text and copyright
- `getSeoMetadata()` - SEO tags
- `getAppearanceSettings()` - Theme colors and fonts

3. API Routes

`/api/site-settings` (GET, POST, PUT)

- **GET**: Retrieve all settings or filter by category
- **POST**: Create new setting (Admin only)
- **PUT**: Bulk update multiple settings
- Role-based access control (SYSADMIN, EIC, MANAGING_EDITOR)
- Audit logging for all changes

`/api/site-settings/[key]` (GET, PATCH, DELETE)

- **GET**: Retrieve single setting by key
- **PATCH**: Update specific setting (Admin only)
- **DELETE**: Remove setting (SYSADMIN only)
- Change tracking with before/after values

4. Admin UI (`/dashboard/admin/cms/settings`)

Features:
- **Tabbed Interface** with 6 categories:
1. **General** (Chung) - Site name, logo, description, keywords
2. **Contact** (Liên hệ) - Email, phone, address, office hours
3. **Social** (Mạng xã hội) - Facebook, Twitter, LinkedIn, YouTube, Instagram, Zalo
4. **SEO** - Meta titles, descriptions, keywords, OG image
5. **Appearance** (Giao diện) - Primary/secondary/accent colors, fonts, header style
6. **Footer** - Footer text, copyright, logo

UI Components:
- Dynamic form fields based on setting type
- Color picker for color settings
- Visual change tracking (unsaved changes indicator)
- Bulk save with confirmation
- Reset functionality
- Contextual help text for each setting
- Bilingual labels (Vietnamese/English)

5. Sidebar Integration

- Added “Cài đặt Website” link in CMS section
- Accessible to SYSADMIN, MANAGING_EDITOR, and EIC roles
- Proper icon (Settings) for easy identification

6. Audit Logging

- Added new audit event types:
- `SETTINGS_CHANGED` - Site settings modifications
- `BANNER_CHANGED`, `MENU_CHANGED`, `HOMEPAGE_CHANGED`, `PAGE_CHANGED` - CMS events
- Tracks who, what, when for all setting changes
- IP address logging for security compliance

7. Seed Data (`seed_site_settings.ts`)

36 Default Settings across 6 categories:

General (7 settings)

- Site name (VN/EN)
- Site description (VN/EN)
- Logo, Favicon
- SEO keywords

Contact (6 settings)

- Email, Phone, Fax
- Address (VN/EN)

- Office hours

Social Media (6 settings)

- Facebook, Twitter, LinkedIn
- YouTube, Instagram, Zalo

SEO (7 settings)

- Meta title (VN/EN)
- Meta description (VN/EN)
- Keywords, Author, OG image

Appearance (5 settings)

- Primary/Secondary/Accent colors
- Font family
- Header style

Footer (5 settings)

- Footer text (VN/EN)
- Copyright (VN/EN)
- Footer logo

📊 Data Model

```
model SiteSetting {
  id          String    @id @default(uuid())
  category    String    // Category grouping
  key         String    @unique // Unique identifier
  value       String?   @db.Text // Setting value
  label       String    // Admin UI label
  labelEn     String?   // English label
  type        String    @default("text") // Input type
  placeholder String?   // Input placeholder
  helpText    String?   @db.Text // Help text
  order       Int       @default(0) // Display order

  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([category])
  @@index([key])
  @@index([order])
}
```

🎨 Key Features

1. Flexible Input Types

- **text**: Simple text input
- **textarea**: Multi-line text
- **url**: URL with validation
- **email**: Email with validation
- **image**: Image URL/path
- **color**: Color picker + hex input

- **json**: Structured data storage
- **boolean**: Toggle switch

2. Multilingual Support

- Vietnamese and English labels
- Bilingual content fields
- Language-aware help text

3. Change Management

- Real-time change detection
- Visual indicators for unsaved changes
- Bulk save optimization
- Reset to original values

4. Security & Compliance

- Role-based access control
- Audit trail for all changes
- IP address logging
- Admin-only modifications

5. Developer Experience

- Cached helper functions for performance
- Type-safe getters (string, boolean, JSON)
- Default fallback values
- Easy integration pattern

💻 Usage Examples

1. Accessing Settings in Server Components

```
import { getSiteInfo, getContactInfo, getSocialLinks } from '@/lib/site-settings';

export default async function MyPage() {
  const siteInfo = await getSiteInfo();
  const contact = await getContactInfo();
  const social = await getSocialLinks();

  return (
    <div>
      <h1>{siteInfo.siteName}</h1>
      <p>{siteInfo.siteDescription}</p>
      <p>Email: {contact.contactEmail}</p>
      <a href={social.facebook}>Facebook</a>
    </div>
  );
}
```

2. Getting Individual Settings

```
import { getSetting, getBooleanSetting } from '@/lib/site-settings';

const siteName = await getSetting('site_name');
const maintenanceMode = await getBooleanSetting('maintenance_mode', false);
```

3. Admin API Usage

```
// Update settings
const response = await fetch('/api/site-settings', {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    settings: [
      { key: 'site_name', value: 'New Name' },
      { key: 'contact_email', value: 'new@email.com' }
    ]
  })
});
```

📂 Files Created/Modified

Created:

1. `prisma/schema.prisma` - Added SiteSetting model
2. `lib/site-settings.ts` - Helper library (300+ lines)
3. `app/api/site-settings/route.ts` - Main API endpoint
4. `app/api/site-settings/[key]/route.ts` - Individual setting API
5. `app/dashboard/admin/cms/settings/page.tsx` - Admin UI (400+ lines)
6. `seed_site_settings.ts` - Default data seeding script
7. `lib/audit-logger.ts` - Added CMS event types

Modified:

1. `components/dashboard/sidebar.tsx` - Added settings link

🔄 Database Migration

Migration completed successfully:

```
# Push schema changes
yarn prisma db push

# Seed default settings
yarn tsx seed_site_settings.ts
```

Result: 36 site settings created across 6 categories

🎯 Admin Workflow

Accessing Settings:

1. Log in as SYSADMIN, EIC, or MANAGING_EDITOR
2. Navigate to Dashboard → Nội dung (CMS) → Cài đặt Website
3. Select desired category tab

Updating Settings:

1. Modify values in the form
2. See “Có thay đổi chưa lưu” indicator
3. Click “Lưu thay đổi” to save
4. Changes tracked in audit log

Managing New Settings:

- POST to `/api/site-settings` with:
- category, key, label, type, value
- Automatically appears in admin UI

🔐 Security & Permissions

Access Control:

- **View Settings**: SYSADMIN, EIC, MANAGING_EDITOR
- **Create/Update Settings**: SYSADMIN, EIC, MANAGING_EDITOR
- **Delete Settings**: SYSADMIN only

Audit Trail:

- All changes logged with:
- User ID (actorId)
- Timestamp
- Before/after values
- IP address

🚀 Performance Optimization

1. **Caching**: All helper functions use React `cache()` for automatic deduplication
2. **Bulk Updates**: Single API call for multiple settings
3. **Indexed Queries**: Database indexes on category, key, order
4. **Selective Loading**: Filter by category to reduce payload

📱 Responsive Design

- Mobile-friendly tabbed interface
- Adaptive grid layout
- Touch-optimized controls
- Responsive color picker

🔧 Future Enhancements

1. **Setting Validation**: Custom validation rules per setting type
2. **Setting Groups**: Hierarchical organization
3. **Import/Export**: Backup/restore settings configuration
4. **Version History**: Track setting changes over time
5. **Environment-specific Settings**: Dev/staging/prod variations
6. **Setting Templates**: Predefined setting collections
7. **Real-time Preview**: See changes before saving
8. **Setting Search**: Quick find across all categories

🧪 Testing

Manual Testing Checklist:

- [x] Access admin settings page
- [x] View all categories
- [x] Update text settings
- [x] Update color settings
- [x] Bulk save multiple settings
- [x] Reset unsaved changes
- [x] Verify audit logging
- [x] Check role-based access

TypeScript Compilation:

```
yarn tsc --noEmit
# ✅ No errors
```

Database Operations:

```
yarn tsx seed_site_settings.ts
# ✅ 36 settings created successfully
```

📚 Documentation

API Documentation:

- **GET** `/api/site-settings` - List all settings
- **GET** `/api/site-settings?category=general` - Filter by category
- **GET** `/api/site-settings/site_name` - Get specific setting
- **POST** `/api/site-settings` - Create setting (Admin)
- **PUT** `/api/site-settings` - Bulk update (Admin)
- **PATCH** `/api/site-settings/[key]` - Update setting (Admin)
- **DELETE** `/api/site-settings/[key]` - Delete setting (SYSADMIN)

Response Format:

```
{
  "success": true,
  "data": {
    "settings": [...]
  },
  "message": "Site settings updated successfully"
}
```

🎉 Benefits

1. **No-Code Management**: Editors can update site-wide settings without developer intervention
2. **Centralized Configuration**: All settings in one place
3. **Type Safety**: Helper functions with TypeScript support
4. **Audit Compliance**: Full change tracking for security
5. **Performance**: Cached queries for fast access
6. **Extensibility**: Easy to add new settings
7. **Multilingual**: Vietnamese and English support
8. **User-Friendly**: Intuitive categorized interface

📋 Conclusion

Phase 16 successfully implements a comprehensive Site Settings CMS that empowers administrators to manage all site-wide configurations through an intuitive web interface. The system is production-ready, secure, performant, and extensible for future enhancements.

Status: ✅ **COMPLETE**

Next Steps:
- Integrate settings into Header component (optional)
- Integrate settings into Footer component (optional)
- Add setting validation rules
- Implement setting change history

Implementation Date: November 13, 2025
Developer: DeepAgent
Phase: 16/16 - Site Settings CMS