TMG CMS News API - Developer Documentation

Overview

The TMG CMS News API provides content management functionality for news articles on the front-end website. This API allows administrators to create, update, delete, and manage news articles that will be displayed to users.

Base URL: `https://api.tmg.com` (replace with your actual API URL)
Version: v1
Authentication: *To be implemented*

Table of Contents

1. Quick Start
2. Data Models
3. API Endpoints
4. Error Handling
5. Status Codes
6. Examples
7. Integration Guide

Quick Start

Base API URL Structure

```
GET    /api/news-articles          # Get all articles (paginated)
GET    /api/news-articles/{id}     # Get specific article
POST   /api/news-articles          # Create new article
PUT    /api/news-articles/{id}     # Update article
DELETE /api/news-articles/{id}     # Delete article
POST   /api/news-articles/{id}/publish    # Publish article
POST   /api/news-articles/{id}/unpublish  # Unpublish article
```

Content-Type

All requests should use:

```
Content-Type: application/json
```

Data Models

NewsArticle Response Model

```json
{
  "id": 123,
  "title": "Breaking News: Major Development",
  "content": "Full article content here...",
  "metaDescription": "SEO meta description",
  "author": "John Doe",
  "summary": "Brief summary of the article",
  "tags": ["technology", "breaking", "business"],
  "category": "Technology",
  "featuredImageUrl": "https://example.com/image.jpg",
  "isPublished": true,
  "publishedAt": "2024-01-15T10:30:00Z",
  "createdAt": "2024-01-15T09:15:00Z",
  "updatedAt": "2024-01-15T10:30:00Z"
}
```

Create Article Request Model

```json
{
  "title": "Article Title (Required, max 200 chars)",
  "content": "Full article content (Required, max 10,000 chars)",
  "metaDescription": "SEO description (Optional, max 160 chars)",
  "author": "Author name (Required, max 100 chars)",
  "summary": "Brief summary (Optional, max 500 chars)",
  "category": "Category name (Optional, max 50 chars)",
  "featuredImageUrl": "Image URL (Optional, max 500 chars)",
  "tags": ["tag1", "tag2", "tag3"]
}
```

Update Article Request Model

json

```json
{
  "title": "Updated title (Required)",
  "content": "Updated content (Required)",
  "metaDescription": "Updated meta description (Optional)",
  "author": "Author name (Required)",
  "summary": "Updated summary (Optional)",
  "category": "Updated category (Optional)",
  "featuredImageUrl": "Updated image URL (Optional)",
  "tags": ["updated", "tags"]
}
```

Paginated Response Model

json

```json
{
  "data": [
    {
      "id": 123,
      "title": "Article Title",
      // ... full article object
    }
  ],
  "totalCount": 156,
  "pageNumber": 1,
  "pageSize": 10,
  "totalPages": 16,
  "hasPreviousPage": false,
  "hasNextPage": true
}
```

Error Response Model

json

```
{
  "title": "Validation Failed",
  "errors": [
    "Title cannot be empty",
    "Content is required",
    "Author must be specified"
  ]
}
```

API Endpoints

1. Get All Articles (Paginated)

Endpoint: `GET /api/news-articles`

Query Parameters:

- `pageNumber` (optional, default: 1) - Page number to retrieve
- `pageSize` (optional, default: 10, max: 100) - Number of articles per page
- `isPublished` (optional) - Filter by published status (true/false)
- `searchTerm` (optional) - Search in title, content, and meta description

Example Request:

```
GET /api/news-articles?pageNumber=1&pageSize=20&isPublished=true&searchTerm=technology
```

Example Response:

```
json
```

```
{
  "data": [
    {
      "id": 123,
      "title": "Latest Tech News",
      "content": "Full content here...",
      "author": "John Doe",
      "isPublished": true,
      "createdAt": "2024-01-15T10:30:00Z"
    }
  ],
  "totalCount": 45,
  "pageNumber": 1,
  "pageSize": 20,
  "totalPages": 3,
  "hasPreviousPage": false,
  "hasNextPage": true
}
```

2. Get Single Article

Endpoint: `GET /api/news-articles/{id}`

Path Parameters:

- `id` (required) - Article ID

Example Request:

```
GET /api/news-articles/123
```

Example Response:

```
json
```

```
{
  "id": 123,
  "title": "Specific Article Title",
  "content": "Full article content...",
  "metaDescription": "SEO description",
  "author": "John Doe",
  "summary": "Article summary",
  "tags": ["tech", "news"],
  "category": "Technology",
  "featuredImageUrl": "https://example.com/image.jpg",
  "isPublished": true,
  "publishedAt": "2024-01-15T10:30:00Z",
  "createdAt": "2024-01-15T09:15:00Z",
  "updatedAt": "2024-01-15T10:30:00Z"
}
```

3. Create New Article

Endpoint: `POST /api/news-articles`

Request Body:

```json
{
  "title": "New Article Title",
  "content": "Full article content here. This should be comprehensive and well-written content that provides value t
  "metaDescription": "SEO-friendly description of the article",
  "author": "Jane Smith",
  "summary": "Brief summary that will be displayed on article lists",
  "category": "Technology",
  "featuredImageUrl": "https://example.com/featured-image.jpg",
  "tags": ["technology", "innovation", "business"]
}
```

Success Response (201 Created):

```json
{
  "id": 124
}
```

4. Update Article

Endpoint: `PUT /api/news-articles/{id}`

Path Parameters:

- `id` (required) - Article ID to update

Request Body:

```json
{
  "title": "Updated Article Title",
  "content": "Updated article content",
  "metaDescription": "Updated meta description",
  "author": "Jane Smith",
  "summary": "Updated summary",
  "category": "Business",
  "featuredImageUrl": "https://example.com/new-image.jpg",
  "tags": ["business", "update", "news"]
}
```

Success Response (204 No Content): *Empty body*

5. Delete Article

Endpoint: `DELETE /api/news-articles/{id}`

Path Parameters:

- `id` (required) - Article ID to delete

Success Response (204 No Content): *Empty body*

6. Publish Article

Endpoint: `POST /api/news-articles/{id}/publish`

Path Parameters:

- `id` (required) - Article ID to publish

Success Response (200 OK):

```json
```

```
{
  "message": "NewsArticle published successfully"
}
```

7. Unpublish Article

Endpoint: `POST /api/news-articles/{id}/unpublish`

Path Parameters:

- `id` (required) - Article ID to unpublish

Success Response (200 OK):

```json
{
  "message": "NewsArticle unpublished successfully"
}
```

Error Handling

Validation Errors (400 Bad Request)

```json
{
  "title": "Validation Failed",
  "errors": [
    "Title cannot exceed 200 characters",
    "Content is required and cannot be empty",
    "Author name must be between 1 and 100 characters"
  ]
}
```

Not Found Errors (404 Not Found)

```json
```

```
{
  "title": "NewsArticle not found",
  "errors": [
    "NewsArticle with ID 999 was not found"
  ]
}
```

Server Errors (500 Internal Server Error)

json

```json
{
  "title": "Internal Server Error",
  "errors": [
    "An unexpected error occurred while processing your request"
  ]
}
```

Status Codes

Code	Description	When It Occurs
`200`	OK	Successful GET requests, Publish/Unpublish operations
`201`	Created	Article successfully created
`204`	No Content	Article successfully updated or deleted
`400`	Bad Request	Validation errors, malformed request
`404`	Not Found	Article with specified ID doesn't exist
`500`	Internal Server Error	Unexpected server-side error

Examples

Front-End Integration Examples

JavaScript/TypeScript (Fetch API)

Get Articles for News Page:

javascript

```javascript
async function getPublishedArticles(page = 1, pageSize = 10) {
  try {
    const response = await fetch(`/api/news-articles?pageNumber=${page}&pageSize=${pageSize}&isPublished=tr

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Error fetching articles:', error);
    throw error;
  }
}

// Usage
getPublishedArticles(1, 20)
  .then(result => {
    console.log(`Found ${result.totalCount} articles`);
    result.data.forEach(article => {
      console.log(`- ${article.title} by ${article.author}`);
    });
  })
  .catch(error => console.error('Failed to load articles:', error));
```

Get Single Article for Article Page:

javascript

```
async function getArticleById(articleId) {
  try {
    const response = await fetch(`/api/news-articles/${articleId}`);

    if (!response.ok) {
      if (response.status === 404) {
        throw new Error('Article not found');
      }
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Error fetching article:', error);
    throw error;
  }
}

// Usage
getArticleById(123)
  .then(article => {
    document.title = article.title;
    document.getElementById('article-title').textContent = article.title;
    document.getElementById('article-content').innerHTML = article.content;
    document.getElementById('article-author').textContent = `By ${article.author}`;
  })
  .catch(error => {
    document.getElementById('article-content').innerHTML = '<p>Article not found</p>';
  });
```

Create Article (Admin Panel):

```javascript
```

```
async function createArticle(articleData) {
  try {
    const response = await fetch('/api/news-articles', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(articleData)
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.errors?.join(', ') || 'Failed to create article');
    }

    const result = await response.json();
    return result.id;
  } catch (error) {
    console.error('Error creating article:', error);
    throw error;
  }
}

// Usage
const newArticle = {
  title: 'My New Article',
  content: 'This is the full content of my article...',
  author: 'John Doe',
  summary: 'A brief summary',
  category: 'Technology',
  tags: ['tech', 'news']
};

createArticle(newArticle)
  .then(articleId => {
    console.log(`Article created with ID: ${articleId}`);
    // Redirect to article or show success message
  })
  .catch(error => {
    alert(`Error: ${error.message}`);
  });
```

React Example

```
jsx
```

```
import React, { useState, useEffect } from 'react';

function NewsArticlesList() {
  const [articles, setArticles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(0);

  useEffect(() => {
    loadArticles(currentPage);
  }, [currentPage]);

  const loadArticles = async (page) => {
    try {
      setLoading(true);
      const response = await fetch(`/api/news-articles?pageNumber=${page}&pageSize=10&isPublished=true`);

      if (!response.ok) {
        throw new Error('Failed to load articles');
      }

      const data = await response.json();
      setArticles(data.data);
      setTotalPages(data.totalPages);
      setError(null);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  if (loading) return <div>Loading articles...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div>
      <h1>Latest News</h1>

      {articles.map(article => (
        <article key={article.id} className="news-article">
          <h2>{article.title}</h2>
          <p className="article-meta">
            By {article.author} • {new Date(article.publishedAt).toLocaleDateString()}
          </p>
          <p>{article.summary}</p>
          <div className="tags">
            {article.tags.map(tag => (
              <span key={tag} className="tag">#{tag}</span>
            ))}
          </div>
        </article>
      ))}

      <div className="pagination">
        <button
          onClick={() => setCurrentPage(p => Math.max(1, p - 1))}
          disabled={currentPage === 1}
        >
          Previous
        </button>

        <span>Page {currentPage} of {totalPages}</span>

        <button
          onClick={() => setCurrentPage(p => Math.min(totalPages, p + 1))}
          disabled={currentPage === totalPages}
        >
          Next
        </button>
      </div>
    </div>
  );
}

export default NewsArticlesList;
```

Integration Guide

For Public Website (Front-End)

1. **News Listing Page**
   - Use `GET /api/news-articles?isPublished=true` to show only published articles
   - Implement pagination for better performance

- Add search functionality using `searchTerm` parameter

2. **Individual Article Page**
   - Use `GET /api/news-articles/{id}` to get full article content
   - Handle 404 errors gracefully (show "Article not found" page)
   - Use `isPublished` field to prevent showing unpublished articles

3. **SEO Considerations**
   - Use `metaDescription` for meta tags
   - Use `title` for page titles and H1 tags
   - Use `tags` for keyword meta tags
   - Use `publishedAt` for article:published_time meta tag

For Admin Panel (CMS)

1. **Article Management**
   - Use all CRUD endpoints (Create, Read, Update, Delete)
   - Implement draft/published workflow using publish/unpublish endpoints
   - Show both published and unpublished articles in admin lists

2. **Form Validation**
   - Implement client-side validation matching API requirements
   - Handle API validation errors and display them to users
   - Use proper form controls for all fields

3. **User Experience**
   - Show loading states during API calls
   - Implement auto-save for long-form content
   - Provide immediate feedback for publish/unpublish actions

Performance Recommendations

1. **Pagination**
   - Always use pagination for article lists (don't load all articles at once)
   - Consider implementing infinite scroll for better UX

2. **Caching**
   - Cache article lists on the front-end for better performance
   - Cache individual articles to reduce API calls

3. **Error Handling**
    - Implement retry logic for network errors
    - Show user-friendly error messages
    - Log errors for debugging

Security Considerations

1. **Input Sanitization**
    - Always sanitize HTML content before displaying
    - Validate all inputs on both client and server side
2. **Authentication** (When Implemented)
    - Secure all CMS endpoints with proper authentication
    - Use HTTPS for all API communications

Testing

Test Data Examples

Sample Article for Testing:

```json
{
  "title": "Test Article for Development",
  "content": "This is a comprehensive test article with enough content to verify that the API is working correctly. It c
  "metaDescription": "A test article created for development and testing purposes to ensure API functionality",
  "author": "Test Author",
  "summary": "This is a test article summary that provides a brief overview of the content",
  "category": "Technology",
  "featuredImageUrl": "https://via.placeholder.com/800x400?text=Test+Image",
  "tags": ["test", "development", "api", "technology"]
}
```

Common Test Scenarios

1. **Create and retrieve an article**
2. **Update article content and verify changes**
3. **Test pagination with different page sizes**

4. **Search functionality with various terms**
5. **Publish/unpublish workflow**
6. **Error handling (invalid IDs, malformed requests)**

Support

For technical questions or issues:

- Check server logs for detailed error information
- Verify request format matches the documentation
- Ensure all required fields are provided
- Confirm data types and length limits

API Version: 1.0
Last Updated: January 2024