Technical Manual

TMGL - Traditional Medicine Global Library

Version: 1.0
Date: 2025
Project: Next.js Frontend Application

Table of Contents

1. Overview
2. Technical Requirements
3. Frameworks and Libraries
4. Project Architecture
5. Project Structure
6. Component Architecture
7. External Integrations
8. API Services
9. Data Models
10. Configuration
11. Build and Deployment
12. Development Guidelines

Overview

The TMGL (Traditional Medicine Global Library) is a Next.js-based frontend application that consumes data from a WordPress REST API. The project is designed as an abstract TMGL-BIREME portal that can be easily populated and managed through the WordPress CMS.

Key Features

- **File-based Routing**: Next.js routing system
- **Server-Side Rendering (SSR)**: Improved SEO and performance
- **Static Site Generation (SSG)**: Pre-rendered pages for better performance
- **API Routes**: Custom API endpoints for external integrations
- **Multi-regional Support**: Dynamic routing for different WHO regions
- **Multi-language Support**: Internationalization capabilities
- **Responsive Design**: Mobile-first approach with Mantine UI

Technical Requirements

Runtime Environment

- **Node.js**: Version 20.11.0 or higher
- **npm**: Version 8.0.0 or higher (or yarn/bun equivalent)

Build Requirements

- **TypeScript**: Version 5.0 or higher
- **Next.js**: Version 14.2.4
- **React**: Version 18.0 or higher

Browser Support

- Chrome (latest)
- Firefox (latest)
- Safari (latest)
- Edge (latest)

Frameworks and Libraries

Core Framework

Next.js 14.2.4

- **Purpose**: React framework for production
- **Features Used**:
  - File-based routing
  - API routes (`/pages/api`)
  - Server-side rendering
  - Static site generation
  - Image optimization
  - Middleware support

Configuration: `next.config.mjs`

```
{
  reactStrictMode: true,
  sassOptions: {
    prependData: `@import "./_mantine.scss";`,
  }
}
```

UI Framework

Mantine 7.11.1

- **Purpose**: React components library
- **Packages Used**:
  - `@mantine/core` : Core components (Buttons, Grid, Container, etc.)
  - `@mantine/carousel` : Carousel/slider components
  - `@mantine/dates` : Date picker components
  - `@mantine/form` : Form management
  - `@mantine/hooks` : Custom React hooks
  - `@mantine/modals` : Modal dialogs

Theme Configuration: `src/styles/mantine-theme.ts`

- Custom color palette (tmgl-blue, tmgl-cyan, tmgl-red, etc.)
- Custom radius values
- Custom font family (Roboto)

React Ecosystem

React 18

- **Features Used**:
  - Functional components
  - Hooks (useState, useEffect, useContext, etc.)
  - Context API for global state

React DOM 18

- Server-side rendering support

HTTP Client

Axios 1.11.0

- **Purpose**: HTTP client for API requests
- **Usage**: All API service classes extend base API classes using Axios

State Management

React Context API

- **Global Context**: `src/contexts/globalContext.tsx`
  - Region name
  - Country name
  - Global configuration
  - Language settings

SWR 2.3.4

- **Purpose**: Data fetching and caching
- **Usage**: Client-side data fetching with automatic revalidation

Styling

SASS/SCSS 1.77.6

- **Purpose**: CSS preprocessor
- **Structure**:
  - `src/styles/custom-global.scss`: Global styles
  - `src/styles/components/*.scss`: Component-specific styles
  - `src/styles/pages/*.scss`: Page-specific styles

PostCSS 8.4.39

- **Purpose**: CSS processing
- **Plugins**:
  - `postcss-preset-mantine`: Mantine-specific PostCSS configuration
  - `postcss-simple-vars`: Variable support

Form Handling

Mantine Form 7.13.1

- **Purpose**: Form state management and validation
- **Usage**: Search forms, filter forms, newsletter subscription

Date Handling

Moment.js 2.30.1

- **Purpose**: Date manipulation and formatting
- **Usage**: Event dates, publication dates

Day.js 1.11.10

- **Purpose**: Lightweight date library (alternative to Moment)

Icons

Tabler Icons 3.16.0

- **Packages**:
  - `@tabler/icons`: Icon library
  - `@tabler/icons-react`: React components for icons

PDF Processing

PDF.js 3.11.174

- **Purpose**: PDF rendering in browser
- **Usage**: PDF viewer component

PDF-lib 1.17.1

- **Purpose**: PDF manipulation
- **Usage**: PDF generation and modification

PDF-poppler 0.2.1

- **Purpose**: PDF to image conversion
- **Usage**: PDF thumbnail generation

PDF2pic 3.2.0

- **Purpose**: PDF to image conversion (alternative)
- **Usage**: Server-side PDF processing

Puppeteer 24.11.2

- **Purpose**: Headless browser automation
- **Usage**: PDF generation, screenshot generation

Image Processing

Sharp 0.34.3

- **Purpose**: High-performance image processing
- **Usage**: Image optimization, resizing, format conversion

Canvas 3.1.2

- **Purpose**: Canvas API for Node.js
- **Usage**: Image manipulation server-side

Media Processing

@coveops/vimeo-thumbnail 1.0.2

- **Purpose**: Vimeo video thumbnail extraction
- **Usage**: Video thumbnail generation

Carousel/Slider

Embla Carousel 7.1.0

- **Purpose**: Carousel component
- **Usage**: Image sliders, content carousels

React Slick 0.30.2

- **Purpose**: Carousel component (alternative)
- **Usage**: Content sliders

Slick Carousel 1.8.1

- **Purpose**: CSS for Slick carousel

React Slideshow Image 4.3.1

- **Purpose**: Image slideshow component

React Background Slider 4.0.0-next.3

- **Purpose**: Background image slider

Data Visualization

@stoddabr/react-tableau-embed-live 0.3.29

- **Purpose**: Tableau visualization embedding
- **Usage**: Evidence maps, data visualizations

Analytics

React Hotjar 6.3.1

- **Purpose**: User behavior analytics
- **Usage**: Heatmaps, session recordings

Utilities

Crypto-js 4.2.0

- **Purpose**: Cryptographic functions
- **Usage**: API key encryption/decryption

HE 1.2.0

- **Purpose**: HTML entity encoding/decoding
- **Usage**: Content sanitization

JS-Cookie 3.0.5

- **Purpose**: Cookie management
- **Usage**: Language preferences, user settings

Spark MD5 3.0.2

- **Purpose**: MD5 hashing
- **Usage**: File hashing, cache keys

XML2JS 0.6.2

- **Purpose**: XML to JavaScript object conversion
- **Usage**: RSS feed parsing

RSS Parser 3.13.0

- **Purpose**: RSS feed parsing
- **Usage**: News feeds, content aggregation

Node-fetch 3.3.2

- **Purpose**: HTTP client for Node.js
- **Usage**: Server-side API requests

Validation

Zod 4.0.10

- **Purpose**: Schema validation
- **Usage**: API response validation, form validation

Type Definitions

All TypeScript type definitions are located in:

- `src/services/types/*.ts`: Service type definitions
- `src/types/@types/*.ts`: Third-party library type definitions

Project Architecture

Architecture Pattern

The project follows a **layered architecture** pattern:

1. **Presentation Layer**: React components (`src/components/`)
2. **Page Layer**: Next.js pages (`src/pages/`)
3. **Service Layer**: API services (`src/services/`)
4. **Data Layer**: Type definitions and DTOs (`src/services/types/`)

Data Flow

```
WordPress REST API
    ↓
BaseUnauthenticatedApi (Axios instance)
    ↓
Service Classes (PostsApi, LisService, etc.)
    ↓
React Components
    ↓
UI Rendering
```

State Management Flow

```
GlobalContext (React Context)
    ↓
GlobalConfigLoader (Initial data fetch)
    ↓
Component Consumption (useContext)
    ↓
Local State (useState) for component-specific data
```

Project Structure

```
src/
├── components/          # Reusable React components
│   ├── breadcrumbs/    # Breadcrumb navigation
│   ├── cards/          # Card components
│   ├── categories/     # Category components
│   ├── embed/          # Embed components (iframes)
│   ├── feed/           # Feed/list components
│   ├── forms/          # Form components
│   ├── gpt/            # GPT widget
│   ├── layout/         # Layout components (header, footer)
│   ├── multitab/       # Multi-tab components
│   ├── pdfview/        # PDF viewer
│   ├── rss/            # RSS feed components
│   ├── sections/       # Page sections
│   ├── share/          # Social sharing
│   ├── slider/         # Slider components
│   ├── video/          # Video components
│   └── videos/         # Video gallery
├── contexts/           # React Context providers
│   ├── globalContext.tsx
│   └── GlobalConfigLoader.tsx
├── data/               # Static data
│   └── countries.ts
├── helpers/            # Utility functions
│   ├── colors.ts
│   ├── crypto.ts
│   ├── regions.ts
│   └── stringhelper.ts
├── middleware.ts       # Next.js middleware
├── pages/              # Next.js pages (routes)
│   ├── api/            # API routes
│   ├── [region]/       # Dynamic regional routes
│   └── ...             # Other pages
├── services/           # API service classes
│   ├── apiRepositories/ # External API services
│   ├── globalConfig/   # Global config service
│   ├── mailchimp/      # MailChimp service
```

```
│   ├── media/          # Media service
│   ├── menus/          # Menu service
│   ├── pages/          # Pages service
│   ├── posts/          # Posts service
│   ├── rss/            # RSS service
│   ├── settings/       # Settings service
│   ├── taxonomies/     # Taxonomies service
│   └── types/          # Type definitions
├── styles/             # Stylesheets
│   ├── components/     # Component styles
│   ├── pages/          # Page styles
│   ├── custom-global.scss
│   └── mantine-theme.ts
└── types/              # TypeScript type definitions
    └── @types/         # Third-party types
```

Component Architecture

Component Creation Pattern

Components follow a consistent pattern:

1. **Functional Components**: All components are functional React components
2. **TypeScript**: Strong typing with interfaces and types
3. **Props Interface**: Defined props interface for each component
4. **Hooks Usage**: useState, useEffect, useContext for state and side effects
5. **Styling**: SCSS modules for component-specific styles

Example Component Structure

```
import { useState, useEffect, useContext } from 'react';
import { GlobalContext } from '@/contexts/globalContext';
import styles from './component.module.scss';

interface ComponentProps {
  title: string;
  data?: any[];
}

export const Component = ({ title, data }: ComponentProps) => {
  const { globalConfig } = useContext(GlobalContext);
  const [state, setState] = useState<string>('');

  useEffect(() => {
    // Side effects
  }, []);

  return (
    <div className={styles.component}>
      {/* Component JSX */}
    </div>
  );
};
```

Component Categories

1. Layout Components

- **Location**: `src/components/layout/`
- **Components**:
  - `HeaderLayout`: Site header with navigation
  - `FooterLayout`: Site footer with links
- **Behavior**:
  - Load menus from WordPress API
  - Handle responsive navigation
  - Manage mega menu state

2. Feed Components

- **Location**: `src/components/feed/`
- **Components**:
  - `NewsFeed`: News listing
  - `EventsFeed`: Events listing
  - `JournalsFeed`: Journals listing
  - `MultimediaFeed`: Multimedia listing
  - `StoriesFeed`: Stories listing
  - `DimensionsFeed`: Dimensions listing
- **Behavior**:
  - Fetch data from WordPress API
  - Handle pagination
  - Apply filters (country, region, thematic area)
  - Support grid/list view toggle

3. Section Components

- **Location**: `src/components/sections/`
- **Components**:
  - `HeroSection`: Hero banner with slider
  - `NewsSection`: News preview section
  - `EventsSection`: Events preview section
  - `StoriesSection`: Stories preview section
  - `DimensionsSection`: Dimensions display
- **Behavior**:
  - Display preview content
  - Link to full listing pages
  - Handle carousel/slider functionality

4. Form Components

- **Location**: `src/components/forms/`
- **Components**:
  - `SearchForm`: Global search form
  - `FiltersForm`: Filter form for listings
- **Behavior**:
  - Handle form submission
  - Manage form state
  - Apply filters to data

5. Utility Components

- **Location**: Various
- **Components**:
  - `BreadCrumbs`: Navigation breadcrumbs
  - `ShareModal`: Social sharing modal
  - `PdfView`: PDF viewer
  - `Video`: Video player
  - `GptWidget`: GPT chat widget

Component Communication

1. **Props**: Parent to child communication
2. **Context**: Global state sharing (GlobalContext)
3. **Callbacks**: Child to parent communication
4. **Events**: DOM events for user interactions

External Integrations

WordPress REST API

Integration Type

- **Protocol**: REST API
- **Authentication**: None (public API)
- **Base URL**: `WP_BASE_URL` environment variable

Endpoints Used

- `/wp-json/wp/v2/posts` : Blog posts
- `/wp-json/wp/v2/pages` : Static pages
- `/wp-json/wp/v2/media` : Media files
- `/wp-json/wp/v2/categories` : Categories
- `/wp-json/wp/v2/tags` : Tags
- `/wp-json/wp/v2/taxonomies` : Custom taxonomies
- `/wp-json/wp/v2/menus` : Navigation menus

Data Flow

```
WordPress CMS
    ↓
WordPress REST API
    ↓
BaseUnauthenticatedApi (Axios)
    ↓
Service Classes (PostsApi, PagesApi, etc.)
    ↓
React Components
```

MailChimp

Integration Type

- **Protocol**: REST API
- **Authentication**: API Key
- **Service**: `src/services/mailchimp/MailChimpService.ts`

Configuration

- `MAILCHIMP_API_KEY`: API key
- `MAILCHIMP_LIST_ID`: List ID
- `MAILCHIMP_DATA_CENTER`: Data center

Usage

- Newsletter subscription
- Email list management

BVSALUD/DIREVE

Integration Type

- **Protocol**: REST API
- **Authentication**: API Key (encrypted)
- **Service**: `src/services/apiRepositories/DireveService.ts`
- **API Route**: `src/pages/api/direve.ts`

Configuration

- `DIREV_API_KEY`: Encrypted API key
- `DIREV_API_URL`: API base URL
- `BVSALUD_URL`: Alternative URL

Usage

- Bibliographic search
- Event data retrieval

LIS (Library Information System)

Integration Type

- **Protocol**: REST API
- **Authentication**: API Key
- **Service**: `src/services/apiRepositories/LisService.ts`
- **API Route**: `src/pages/api/lis.ts`

Configuration

- `LIS_API_URL`: API base URL

Usage

- Database and repository listings
- Bibliographic resources

Evidence Maps API

Integration Type

- **Protocol**: REST API
- **Authentication**: API Key
- **Service**: `src/services/apiRepositories/EvidenceMapsService.ts`
- **API Route**: `src/pages/api/evidencemaps.ts`

Usage

- Evidence maps data
- Research evidence visualization

Journals API

Integration Type

- **Protocol**: REST API
- **Service**: `src/services/apiRepositories/JournalsService.ts`
- **API Route**: `src/pages/api/journals.ts`

Configuration

- `JOURNALS_API_URL`: API base URL

Usage

- Scientific journals listing
- Journal metadata

Multimedia API

Integration Type

- **Protocol**: REST API
- **Service**: `src/services/apiRepositories/MultimediaService.ts`
- **API Route**: `src/pages/api/multimedia.ts`

Configuration

- `MULTIMEDIA_API_URL`: API base URL

Usage

- Video and image content
- Multimedia metadata

RSS Feeds

Integration Type

- **Protocol**: RSS/XML
- **Service**: `src/services/rss/RssService.ts`
- **API Route**: `src/pages/api/rssfeed.ts`

Configuration

- `RSS_FEED_URL`: RSS feed URL

Usage

- News aggregation
- Content syndication

Tableau

Integration Type

- **Library**: `@stoddabr/react-tableau-embed-live`
- **Usage**: Embedded Tableau visualizations
- **Components**: Evidence maps, data dashboards

API Services

Base Service Class

BaseUnauthenticatedApi

- **Location**: `src/services/BaseUnauthenticatedApi.ts`
- **Purpose**: Base class for all WordPress API services
- **Features**:
  - Axios instance creation
  - Language handling (cookie-based)
  - Region support
  - Featured media extraction
  - Image URL formatting

Key Methods:

- `findFeaturedMedia(post, size)`: Extract featured image URL
- Constructor: Initialize Axios instance with base URL

WordPress Services

PostsApi

- **Location**: `src/services/posts/PostsApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - `getCustomPost()`: Get custom post types
  - `listPosts()`: List posts with pagination
  - `getPost()`: Get single post by slug
  - `getPostById()`: Get post by ID
  - `getTaxonomies()`: Get taxonomies
  - `formatTags()`: Format post tags
  - `getFeaturedImageById()`: Get featured image

PagesApi

- **Location**: `src/services/pages/PagesApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - `getPageProperties()`: Get page ACF properties

MenusApi

- **Location**: `src/services/menus/MenusApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - `getMenu()`: Get menu by slug

TaxonomiesApi

- **Location**: `src/services/taxonomies/TaxonomiesApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - `getTaxonomies()`: Get taxonomies
  - `getTaxonomyTerms()`: Get taxonomy terms

MediaApi

- **Location**: `src/services/media/MediaApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - Media file operations

SettingsApi

- **Location**: `src/services/settings/SettingsApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - WordPress settings retrieval

Global Config Service

GlobalConfigApi

- **Location**: `src/services/globalConfig/GlobalConfigApi.ts`
- **Extends**: `BaseUnauthenticatedApi`
- **Methods**:
  - `getGlobalConfig()`: Get global configuration
- **Validation**: Uses Zod schema (`GlobalConfigZodSchema.ts`)

External API Services

LisService

- **Location**: `src/services/apiRepositories/LisService.ts`
- **Methods**:
  - `getResources()`: Get LIS resources
  - `getItem()`: Get single resource

DireveService

- **Location**: `src/services/apiRepositories/DireveService.ts`
- **Methods**:
  - Bibliographic search
  - Event data retrieval

EvidenceMapsService

- **Location**: `src/services/apiRepositories/EvidenceMapsService.ts`
- **Methods**:
  - Evidence maps data retrieval

JournalsService

- **Location**: `src/services/apiRepositories/JournalsService.ts`
- **Methods**:
  - Journals data retrieval

MultimediaService

- **Location**: `src/services/apiRepositories/MultimediaService.ts`
- **Methods**:
  - Multimedia data retrieval

RegulationAndPolicesService

- **Location**: `src/services/apiRepositories/RegulationAndPolices.ts`
- **Methods**:
  - Regulations and policies data

GlobalSummitService

- **Location**: `src/services/apiRepositories/GlobalSummitService.ts`
- **Methods**:
  - Global Summit data

MailChimp Service

MailChimpService

- **Location**: `src/services/mailchimp/MailChimpService.ts`
- **Methods**:
  - `subscribe()`: Subscribe user to newsletter

RSS Service

RssService

- **Location**: `src/services/rss/RssService.ts`
- **Methods**:
  - `FetchRSSFeed()`: Fetch and parse RSS feed

Data Models

Type Definitions

All type definitions are located in `src/services/types/`:

Post Types

- `posts.dto.ts`: Post data structure
- `pages.dto.ts`: Page data structure

Custom Post Types

- `eventsDto.ts`: Events
- `featuredStoriesAcf.ts`: Featured stories
- `journalsDto.ts`: Journals
- `multimediaTypes.ts`: Multimedia
- `legislationsDto.ts`: Legislations
- `regulationsAndPolices.ts`: Regulations

API Response Types

- `evidenceMapsDto.ts`: Evidence maps
- `direveTypes.ts`: DIREVE API
- `repositoryTypes.ts`: Repository API
- `rssFeedTypes.ts`: RSS feeds

Configuration Types

- `globalAcf.ts`: Global configuration
- `homeAcf.dto.ts`: Homepage configuration
- `menus.dto.ts`: Menu structure
- `taxonomies.dto.ts`: Taxonomies

Resource Types

- `resources.ts`: Generic resources
- `defaultResource.ts`: Default resource structure

Data Flow Models

WordPress Post Model

```
interface Post {
  id: number;
  title: { rendered: string };
  content: { rendered: string };
  excerpt: { rendered: string };
  slug: string;
  date: string;
  acf: any; // Advanced Custom Fields
  _embedded: {
    'wp:featuredmedia': Media[];
    'wp:term': Term[][];
  };
}
```

Filter Model

```
interface queryType {
  parameter: string;
  query: string;
}
```

Tag Model

```
interface TagItem {
  name: string;
  type: 'country' | 'region' | 'descriptor';
}
```

Configuration

Environment Variables

All environment variables are configured in `next.config.mjs`:

WordPress Configuration

- `WP_BASE_URL`: WordPress REST API base URL
- `BASE_URL`: Application base URL
- `NEXT_PUBLIC_BASE_URL`: Public base URL
- `NEXT_PUBLIC_API_BASE_URL`: Public API base URL

Content Configuration

- `POSTSPERPAGE`: Default posts per page
- `BASE_SEARCH_URL`: Search base URL

External APIs

- `DIREV_API_KEY` : DIREVE API key (encrypted)
- `DIREV_API_URL` : DIREVE API URL
- `LIS_API_URL` : LIS API URL
- `JOURNALS_API_URL` : Journals API URL
- `MULTIMEDIA_API_URL` : Multimedia API URL
- `BVSALUD_URL` : BVSALUD URL
- `BVSALUD_API_KEY` : BVSALUD API key

MailChimp

- `MAILCHIMP_API_KEY` : MailChimp API key
- `MAILCHIMP_LIST_ID` : MailChimp list ID
- `MAILCHIMP_DATA_CENTER` : MailChimp data center

RSS

- `RSS_FEED_URL` : RSS feed URL

Security

- `SECRET` : Application secret key

Other

- `FIADMIN_URL` : FI Admin URL

TypeScript Configuration

File: `tsconfig.json`

Key Settings:

- Target: ES2017
- Module: ESNext
- JSX: Preserve
- Strict mode: Enabled
- Path aliases:
  - `@/*` → `./src/*`
  - `@styles/*` → `./src/styles/*`
  - `@components/*` → `./src/components/*`
  - `@contexts/*` → `./src/contexts/*`

Next.js Configuration

File: `next.config.mjs`

Key Settings:

- React Strict Mode: Enabled
- SASS options: Mantine SCSS import
- Environment variables: All exposed variables

Mantine Theme Configuration

File: `src/styles/mantine-theme.ts`

Customizations:

- Color palette: TMGL brand colors
- Font family: Roboto
- Border radius: Custom values
- Primary color: tmgl-blue

Build and Deployment

Development

```
npm run dev
```

Starts development server on `http://localhost:3000`

Build

```
npm run build
```

Creates optimized production build in `.next/` directory

Start Production Server

```
npm start
```

Starts production server (requires build first)

Linting

```
npm run lint
```

Runs ESLint for code quality checks

Build Process

1. **Type Checking**: TypeScript compilation
2. **Code Optimization**: Next.js optimization
3. **Static Generation**: Pre-rendering static pages
4. **Bundle Optimization**: Code splitting and tree shaking
5. **Asset Optimization**: Image and font optimization

Deployment Considerations

1. **Environment Variables**: Must be set in deployment environment
2. **Node.js Version**: Must match required version (20.11.0+)
3. **Build Time**: Consider build time for large applications
4. **Static Assets**: Ensure static assets are properly served
5. **API Routes**: Server-side API routes require Node.js runtime

---

Development Guidelines

Code Style

1. **TypeScript**: Use TypeScript for all new code
2. **Functional Components**: Prefer functional components over class components
3. **Hooks**: Use React hooks for state and side effects
4. **Naming Conventions**:
    - Components: PascalCase
    - Files: camelCase for utilities, PascalCase for components
    - Constants: UPPER_SNAKE_CASE

Component Guidelines

1. **Single Responsibility**: Each component should have a single responsibility
2. **Props Interface**: Always define props interface
3. **Error Handling**: Implement proper error handling
4. **Loading States**: Show loading states during data fetching
5. **Accessibility**: Follow WCAG guidelines

API Service Guidelines

1. **Extend Base Class**: All WordPress services should extend `BaseUnauthenticatedApi`
2. **Error Handling**: Implement try-catch blocks
3. **Type Safety**: Use TypeScript interfaces for API responses
4. **Caching**: Consider caching strategies for frequently accessed data

Styling Guidelines

1. **SCSS Modules**: Use SCSS modules for component styles
2. **Mantine Components**: Prefer Mantine components over custom CSS
3. **Responsive Design**: Mobile-first approach
4. **Theme Colors**: Use theme colors from `mantine-theme.ts`

Testing Considerations

1. **Component Testing**: Test component rendering and interactions
2. **API Testing**: Test API service methods
3. **Integration Testing**: Test component integration
4. **E2E Testing**: Consider end-to-end testing for critical flows

---

Additional Resources

Documentation Links

- Next.js Documentation
- Mantine Documentation
- WordPress REST API
- React Documentation
- TypeScript Documentation

Project-Specific Documentation

- Technical Site Map: See `Technical Site Map.pdf`
- Component Documentation: See component files in `src/components/`
- API Documentation: See service files in `src/services/`

Last Updated: 2025
Version: 1.0
Maintained By: TMGL Development Team