KTRH Hospital Website - Strapi Backend Documentation

I. Overview

This document provides the setup and configuration guide for the Strapi backend of the KTRH hospital website.
The website frontend is built with HTML, CSS, and JavaScript, and uses Strapi as a headless CMS.

II. Prerequisites

- Node.js and npm installed
- Strapi project created (if not, follow: https://strapi.io/documentation/developer-docs/latest/getting-started/quick-start.html)
- Basic knowledge of Strapi admin panel

III. Strapi Project Initial Setup

1. Create a new Strapi project (if not already done):

```bash
npx create-strapi-app ktrh-backend --quickstart
```

2. Once the project is created, log in to the admin panel (usually at http://localhost:1337/admin) and create an admin account.

IV. Content Types Setup

We have to create several content types (collection types and single types) as per the documentation.

A. Projects Page

1. **Collection Type: Project**
   - Create a new collection type named "Project".
   - Add the following fields (with types and settings as below):

Field Name	**Type**	**Required**	**Description**
title	Text (short)	Yes	Project title

Field Name	Type	Required	Description
slug	UID	Yes (attached to title)	URL slug
shortDescription	Text (short)	Yes	Short description for cards
fullDescription	Rich Text	Yes	Full project description
category	Enumeration	Yes	Options: ongoing, completed, upcoming, research, community
status	Text (short)	Yes	e.g., "75% Complete"
progress	Number	Yes	0-100
startDate	Date	Yes	Start date
endDate	Date	Yes	End date
budget	Text (short)	Yes	e.g., "KES 850,000,000"
fundingSource	Text (short)	Yes	Funding sources
contractor	Text (short)	No	Contractor name
projectManager	Text (short)	Yes	Project manager name
contactEmail	Email	Yes	Contact email
contactPhone	Text (short)	Yes	Contact phone
coverImage	Media (single image)	Yes	Cover image
imageGallery	Media (multiple images)	No	Gallery images

Field Name	Type	Required	Description
impactPoints	JSON	No	Array of strings
milestones	JSON	No	Array of objects with date and description
partners	JSON	No	Array of strings
beneficiaries	Text (short)	No	Beneficiaries description
isFeatured	Boolean	Yes	Default: false
displayOrder	Number	Yes	For ordering

- Save the content type.

2. **Single Type: ProjectPage**

- Create a single type named "ProjectPage" for the project page content.
- Add fields:

Field Name	Type	Description
pageTitle	Text (short)	Default: "Our Projects"
heroTitle	Text (short)	Default: "Transforming Healthcare Through Innovation"
heroDescription	Text (short)	Hero description
heroImage	Media (single image)	Hero image
introductionText	Rich Text	Introduction text
statisticsTitle	Text (short)	Default: "Our Impact in Numbers"
fundingInformation	Rich Text	Funding information

Field Name	Type	Description
contactTitle	Text (short)	Default: "Project Inquiries"
contactEmail	Email	Contact email for inquiries
contactPhone	Text (short)	Contact phone for inquiries

3. **Collection Type: ProjectStats**
   - Create a collection type named "ProjectStats" for statistics.
   - Add fields (all numbers except lastUpdated):

Field Name	Type	Description
activeProjects	Number	Active projects count
completedProjects	Number	Completed projects count
totalInvestment	Text (short)	Total investment as string (e.g., "KES 2.5B")
partnerOrganizations	Number	Number of partner organizations
jobsCreated	Number	Number of jobs created
patientsServed	Number	Number of patients served
lastUpdated	Text (short)	Last updated date (e.g., "2024-01-01")

B. Services Page

1. **Collection Type: Service**
   - Create a collection type named "Service".
   - Fields:

Field Name	Type	Required	Description
name	Text (short)	Yes	Service name

Field Name	Type	Required	Description
slug	UID	Yes (attached to name)	URL slug
category	Enumeration	Yes	Options: specialized, diagnostic, surgical, support
icon_class	Text (short)	Yes	FontAwesome class (e.g., "fas fa-heartbeat")
description	Rich Text	Yes	Full description
image	Media (single image)	Yes	Service image
short_description	Text (short)	No	Short summary for cards
operating_hours	Text (short)	No	e.g., "Mon-Fri: 8AM-6PM"
emergency_available	Boolean	No	If 24/7 service available
is_emergency_service	Boolean	No	If it's an emergency service
features	JSON	No	Array of features
equipment	JSON	No	Array of equipment
procedures	JSON	No	Array of procedures
specialists	JSON	No	Array of specialist objects (each with name, specialty, qualification, experience)
contact	JSON	No	Contact info object (phone, email, location, hours)

C. Departments Page

1. **Collection Type: Department**
   - Create a collection type named "Department".
   - Fields:

Field Name	Type	Required	Description
name	Text (short)	Yes	Department name
slug	UID	Yes (attached to name)	URL slug
category	Enumeration	Yes	Options: clinical, surgical, diagnostic, support
icon_class	Text (short)	Yes	FontAwesome class (e.g., "fas fa-heartbeat")
description	Rich Text	Yes	Full description
image	Media (single image)	Yes	Department image
head_of_department	Text (short)	No	Head of department name
head_qualification	Text (short)	No	Head's qualification
head_experience	Text (short)	No	Head's experience (e.g., "15+ years")
extension	Text (short)	No	Extension number
email	Email	No	Department email
location	Text (short)	No	Department location
operating_hours	Text (short)	No	Operating hours
services	JSON	No	Array of services

Field Name	Type	Required	Description
equipment	JSON	No	Array of equipment
staff	JSON	No	Array of staff objects (each with name, role, qualification, experience)
stats	JSON	No	Object with statistics (patients, procedures, successRate)
contact	JSON	No	Contact info object (phone, email, location, hours)

D. Doctors Page

1. **Collection Type: Doctor**
   - Create a collection type named "Doctor".
   - Fields:

Field Name	Type	Required	Description
name	Text (short)	Yes	Doctor's full name
slug	UID	Yes (attached to name)	URL slug
title	Text (short)	Yes	Professional title (e.g., "Senior Consultant Cardiologist")
department	Enumeration	Yes	Options: cardiology, neurology, pediatrics, orthopedics, surgery, emergency
image	Media (single image)	Yes	Doctor's profile picture

Field Name	Type	Required	Description
bio	Rich Text	Yes	Detailed biography
experience	Text (short)	Yes	Years of experience (e.g., "15+ Years")
qualifications	JSON	Yes	Array of qualifications
expertise	JSON	Yes	Array of expertise tags
consultation_hours	Text (short)	Yes	Consultation schedule (e.g., "Mon, Wed, Fri: 9AM-4PM")
achievements	Rich Text	No	List of achievements
availability	JSON	No	Array of availability objects
contact	JSON	No	Contact info object
stats	JSON	No	Statistics object
is_featured	Boolean	No	Mark as Doctor of the Month
order	Number	No	Display order

E. Gallery Page

1. **Collection Type: Gallery Item**
   - Create a collection type named "Gallery Item" (API ID: gallery-item).
   - Fields:

Field Name	Type	Required	Description
title	Text (short)	Yes	Title of the gallery item
description	Text (long)	Yes	Detailed description

Field Name	Type	Required	Description
category	Enumeration	Yes	Options: events, facilities, team, achievements, community, videos
mediaType	Enumeration	Yes	Options: image, video
date	Date	Yes	Date when the media was captured/created
publishedAt	DateTime	No	For scheduling/draft mode
image	Media (single image)	No (conditional)	Main image (required if mediaType is image)
thumbnail	Media (single image)	No	Thumbnail image (optional)
videoUrl	Text (short)	No (conditional)	Video URL (required if mediaType is video)
location	Text (short)	No	Location where media was captured
isFeatured	Boolean	No	For featured videos section (default: false)
tags	JSON	No	Array of tags for search/filtering
views	Number	No	View count (default: 0)

- Note: For videoUrl, use a regex pattern to validate YouTube or Vimeo URLs.

F. About Page

1. **Single Type: AboutPage**
   - Create a single type named "AboutPage".
   - The documentation does not specify the fields for the About page. However, from the example in the file, it seems the About page might use the same structure as the Gallery? But note the example given in the About section is the same as the Gallery example.

Since the About page is not detailed, we might need to clarify with the original developer. However, for now, we can assume that the About page is a static page with content that can be managed via a single type.
Suggested fields for AboutPage:

Field Name	**Type**	**Description**
heroTitle	Text (short)	Hero title
heroDescription	Text (short)	Hero description
heroImage	Media (single image)	Hero image
mission	Rich Text	Mission statement
vision	Rich Text	Vision statement
history	Rich Text	History of the hospital
values	JSON	Array of value objects (each with title and description)
leadership	JSON	Array of leadership team members (each with name, title, bio, image)
milestones	JSON	Array of milestone objects (each with year and description)

Alternatively, the About page might have multiple sections. We can adjust as needed.

V. Populating Data

1. After creating the content types, go to the Content Manager in Strapi.
2. For each content type, click "Create New Entry" and fill in the data as per the examples provided in the original documentation.
3. Make sure to publish the entries (set the publishedAt date or use the publish button).

VI. API Permissions

1. Go to Settings → Users & Permissions Plugin → Roles.

2. Edit the "Public" role.
3. For each content type (Project, Service, Department, Doctor, Gallery Item, AboutPage, ProjectPage, ProjectStats), enable:
   - find (for listing)
   - findOne (for single entry)
4. Click Save.

VII. Frontend Integration Notes

- The frontend (HTML, CSS, JS) will fetch data from the Strapi API endpoints.
- The base URL for the API is typically: http://localhost:1337/api (for local development).
- Endpoints:
  - Projects: GET /api/projects
  - Single Project: GET /api/projects/:id or /api/projects?slug=:slug
  - ProjectPage: GET /api/project-page
  - ProjectStats: GET /api/project-stats
  - Services: GET /api/services
  - Departments: GET /api/departments
  - Doctors: GET /api/doctors
  - Gallery Items: GET /api/gallery-items
  - AboutPage: GET /api/about-page
- Use the Strapi REST API query parameters to filter, sort, and populate fields as needed.

VIII. Additional Notes

- The Strapi server must be running for the frontend to work (in development, run `npm run develop` in the Strapi project directory).
- For production, build and deploy the Strapi project following the Strapi deployment guides.

This documentation should be sufficient for a technician to set up the Strapi backend and populate data. However, for frontend-specific issues, refer to the frontend code and comments.

Let me know if you need any further details.

KTRH Hospital Website - Technical Documentation

Technology Stack: HTML, CSS, JavaScript (Frontend) + Strapi (Backend/Headless CMS)

---

1. Overview & Architecture

Project Structure

```text
ktrh-website/
├── frontend/                 # HTML, CSS, JavaScript files
│   ├── index.html           # Homepage
│   ├── projects.html        # Projects page
│   ├── services.html        # Services page
│   ├── departments.html     # Departments page
│   ├── doctors.html         # Doctors page
│   ├── gallery.html         # Gallery page
│   ├── about.html           # About page
│   ├── css/
│   ├── js/
│   └── assets/
└── strapi-backend/          # Strapi CMS directory
    ├── src/
    ├── config/
    └── ...
```

Data Flow

```text
Strapi API (localhost:1337) → Frontend (JavaScript Fetch) → HTML Display
```

---

2. Strapi Backend Setup & Configuration

Initial Setup

```bash
# 1. Navigate to Strapi directory
cd strapi-backend

# 2. Install dependencies (if not done)
npm install

# 3. Start Strapi development server
npm run develop
# Access admin panel at: http://localhost:1337/admin
```

Default Admin Credentials

- **URL:** `http://localhost:1337/admin`
- **Username/Email:** [Set during initial setup]
- **Password:** [Set during initial setup]

---

3. Content Types Configuration

Follow this sequence to recreate all content types if starting fresh:

A. Projects Content Types

1. Collection Type: `project`

Purpose: Manage individual hospital projects

Fields to create:

```text
1. title (Text - Short Text) [Required]
2. slug (UID) [Required - Attach to: title]
3. shortDescription (Text - Long Text)
4. fullDescription (Rich Text)
```

```
5. category (Enumeration)
   Options: ongoing, completed, upcoming, research, community
6. status (Text - Short Text)
7. progress (Number - Integer, 0-100)
8. startDate (Date)
9. endDate (Date)
10. budget (Text - Short Text)
11. fundingSource (Text - Long Text)
12. contractor (Text - Short Text) [Optional]
13. projectManager (Text - Short Text)
14. contactEmail (Email)
15. contactPhone (Text - Short Text)
16. coverImage (Media - Single image)
17. imageGallery (Media - Multiple images)
18. impactPoints (JSON)
19. milestones (JSON)
20. partners (JSON)
21. beneficiaries (Text - Long Text)
22. isFeatured (Boolean)
23. displayOrder (Number - Integer)
```

2. Single Type: `project-page`

Purpose: Homepage content for projects section

Fields:

```text
1. pageTitle (Text - Short Text)
2. heroTitle (Text - Short Text)
3. heroDescription (Text - Long Text)
4. heroImage (Media - Single image)
5. introductionText (Rich Text)
6. statisticsTitle (Text - Short Text)
7. fundingInformation (Rich Text)
8. contactTitle (Text - Short Text)
9. contactEmail (Email)
10. contactPhone (Text - Short Text)
```

3. Collection Type: `project-stats`

Purpose: Display statistics on projects page

Fields:

```text
1. activeProjects (Number - Integer)
2. completedProjects (Number - Integer)
3. totalInvestment (Text - Short Text)
4. partnerOrganizations (Number - Integer)
5. jobsCreated (Number - Integer)
6. patientsServed (Number - Integer)
7. lastUpdated (Text - Short Text)
```

B. Services Content Type

Collection Type: `service`

Purpose: Manage hospital services

Fields:

```text
1. name (Text - Short Text) [Required]
2. slug (UID) [Required - Attach to: name]
3. category (Enumeration)
   Options: specialized, diagnostic, surgical, support
4. icon_class (Text - Short Text)
5. description (Rich Text)
6. image (Media - Single image)
7. short_description (Text - Long Text)
8. operating_hours (Text - Short Text)
9. emergency_available (Boolean)
10. is_emergency_service (Boolean)
11. features (JSON)
```

```
12. equipment (JSON)
13. procedures (JSON)
14. specialists (JSON)
15. contact (JSON)
```

C. Departments Content Type

Collection Type: `department`

Purpose: Manage hospital departments

Fields:

```text
1. name (Text - Short Text) [Required]
2. slug (UID) [Required - Attach to: name]
3. category (Enumeration)
   Options: clinical, surgical, diagnostic, support
4. icon_class (Text - Short Text)
5. description (Rich Text)
6. image (Media - Single image)
7. head_of_department (Text - Short Text)
8. head_qualification (Text - Short Text)
9. head_experience (Text - Short Text)
10. extension (Text - Short Text)
11. email (Email)
12. location (Text - Short Text)
13. operating_hours (Text - Short Text)
14. services (JSON)
15. equipment (JSON)
16. staff (JSON)
17. stats (JSON)
18. contact (JSON)
```

D. Doctors Content Type

Collection Type: `doctor`

Purpose: Manage doctor profiles

Fields:

```text
1. name (Text - Short Text) [Required]
2. slug (UID) [Required - Attach to: name]
3. title (Text - Short Text) [Required]
4. department (Enumeration)
   Options: cardiology, neurology, pediatrics, orthopedics, surgery, emergency
5. image (Media - Single image)
6. bio (Rich Text)
7. experience (Text - Short Text)
8. qualifications (JSON)
9. expertise (JSON)
10. consultation_hours (Text - Short Text)
11. achievements (Rich Text) [Optional]
12. availability (JSON) [Optional]
13. contact (JSON) [Optional]
14. stats (JSON) [Optional]
15. is_featured (Boolean)
16. order (Number - Integer)
```

E. Gallery Content Type

Collection Type: `gallery-item`

Purpose: Manage gallery images/videos

Fields:

text

```
1. title (Text - Short Text) [Required]
2. description (Text - Long Text) [Required]
3. category (Enumeration) [Required]
   Options: events, facilities, team, achievements, community, videos
4. mediaType (Enumeration) [Required]
   Options: image, video
5. date (Date) [Required]
6. publishedAt (DateTime)
7. image (Media - Single image) [Conditional: if mediaType = image]
8. thumbnail (Media - Single image) [Optional]
9. videoUrl (Text - Short Text) [Conditional: if mediaType = video]
   Regex pattern: ^https?:\/\/(?:www\.)?(?:youtube\.com\/watch\?v=|youtu\.be\/|vimeo\.com\/)[\w-]+$
10. location (Text - Short Text) [Optional]
11. isFeatured (Boolean)
12. tags (JSON) [Optional]
13. views (Number - Integer)
```

F. About Page Content Type

Single Type: `about-page`

Purpose: Manage about page content

Fields:

```text
1. title (Text - Short Text)
2. heroImage (Media - Single image)
3. missionStatement (Rich Text)
4. visionStatement (Rich Text)
5. history (Rich Text)
6. values (JSON)
7. leadership (JSON)
8. milestones (JSON)
9. contactInfo (JSON)
```

---

4. API Permissions Setup

Important: This must be done after creating content types

Steps:

1. Go to **Settings** → **Users & Permissions Plugin** → **Roles**
2. Edit **Public** role
3. For EACH content type, enable:
   - `find` (for listing items)
   - `findOne` (for viewing single item)
4. Click **Save**

Content Types to enable permissions for:

- project
- project-page
- project-stats
- service
- department
- doctor
- gallery-item
- about-page

---

5. Frontend Integration Guide

A. API Endpoints Structure

```javascript
// Base URL (Development)
const API_BASE = 'http://localhost:1337/api';

// Endpoints:
const ENDPOINTS = {
  projects: `${API_BASE}/projects`,
  projectPage: `${API_BASE}/project-page`,
  projectStats: `${API_BASE}/project-stats`,
  services: `${API_BASE}/services`,
  departments: `${API_BASE}/departments`,
  doctors: `${API_BASE}/doctors`,
  gallery: `${API_BASE}/gallery-items`,
  about: `${API_BASE}/about-page`
};

// Query parameters for filtering:
// ?filters[category][$eq]=ongoing
// ?populate=* (to include media)
// ?sort=displayOrder:asc
```

B. Sample Fetch Request

```javascript
// Fetch all projects
async function fetchProjects() {
  try {
    const response = await fetch('http://localhost:1337/api/projects?populate=*');
    const data = await response.json();
    return data.data;
  } catch (error) {
    console.error('Error fetching projects:', error);
    return [];
  }
}

// Fetch single project by slug
```

```javascript
async function fetchProjectBySlug(slug) {
  try {
    const response = await fetch(
      `http://localhost:1337/api/projects?filters[slug][$eq]=${slug}&populate=*`
    );
    const data = await response.json();
    return data.data[0];
  } catch (error) {
    console.error('Error fetching project:', error);
    return null;
  }
}
```

C. Image URL Construction

```javascript
// Strapi returns media in this format:
const imageUrl = `http://localhost:1337${imageData.attributes.url}`;

// For thumbnails or different formats:
const formats = imageData.attributes.formats;
const thumbnailUrl = `http://localhost:1337${formats.thumbnail.url}`;
const smallUrl = `http://localhost:1337${formats.small.url}`;
```

---

6. Troubleshooting Guide

Common Issues & Solutions:

1. "Cannot GET /api/..." error

```bash
# Check if Strapi is running
# Solution: Ensure Strapi server is started
npm run develop
```

```
# Check if content type permissions are set
# Solution: Set Public role permissions as in Section 4
```

2. Images not displaying

javascript

```javascript
// Check if populate=* is in query
// Incorrect: /api/projects
// Correct: /api/projects?populate=*

// Check image URL construction
console.log(imageData); // Check structure
```

3. CORS Errors

javascript

```javascript
// In Strapi config/middlewares.js:
module.exports = [
  'strapi::errors',
  'strapi::security',
  'strapi::cors',
  'strapi::poweredBy',
  'strapi::logger',
  'strapi::query',
  'strapi::body',
  'strapi::session',
  'strapi::favicon',
  'strapi::public',
];

// In config/cors.js (create if doesn't exist):
module.exports = {
  origin: ['http://localhost:3000', 'http://127.0.0.1:5500'], // Your frontend URL
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'HEAD', 'OPTIONS'],
  headers: ['Content-Type', 'Authorization', 'Origin', 'Accept'],
  keepHeaderOnError: true,
};
```

4. Data not appearing on frontend

javascript

```
// Check:
// 1. Is the entry published in Strapi?
// 2. Are you using correct API endpoint?
// 3. Check browser console for errors
// 4. Verify data structure with console.log()
```

---

7. Maintenance Tasks

Daily Tasks:

1. **Check Strapi server status**
2. **Verify all content types are accessible**
3. **Test API endpoints** (use Postman or browser)
4. **Check for unpublished content**

Weekly Tasks:

1. **Backup Strapi database**

```bash
# Export data
strapi export --no-encrypt
```

2. **Update displayOrder fields** for proper sorting
3. **Review and update featured content** (isFeatured flags)

Monthly Tasks:

1. **Update project progress percentages**
2. **Review and archive completed projects**

3. **Update statistics** (project-stats collection)
4. **Backup entire Strapi project folder**

---

8. Deployment Notes

Production Changes:

1. **Update API base URL** in frontend JavaScript:

```javascript
// Change from:
const API_BASE = 'http://localhost:1337/api';

// To production URL:
const API_BASE = 'https://your-domain.com/api';
```

2. **Update CORS configuration** in Strapi for production domain
3. **Set environment variables** for production:

```bash
NODE_ENV=production
DATABASE_URL=your_production_db_url
```

Build Commands:

```bash
# Frontend (if using build tools)
npm run build

# Strapi production build
NODE_ENV=production npm run build
```

---

9. Emergency Contact & Recovery

If I'm unavailable:

1. **Access to Strapi Admin:**
   - URL: [Your Strapi Admin URL]
   - Contact [Backup Admin] for credentials
2. **Server Access:**
   - SSH: `ssh user@server-ip`
   - Strapi Directory: `/var/www/strapi/`
   - Logs: `/var/www/strapi/logs/`
3. **Database Backup Location:**
   - Path: `/var/www/strapi/backups/`
   - Auto-backup: Daily at 2 AM

Quick Recovery Steps:

1. Restart Strapi: `pm2 restart strapi`
2. Check logs: `pm2 logs strapi`
3. Restore from backup: `strapi import backup-file.tar.gz`

---

10. Sample Data for Testing

Quick Test Entry (Doctor):

json

```
{
  "name": "Dr. Test User",
  "department": "cardiology",
  "title": "Test Cardiologist",
  "experience": "5+ Years",
  "consultation_hours": "Mon-Fri: 9AM-5PM",
  "qualifications": ["MD, Test University"],
  "expertise": ["General Cardiology", "Testing"]
}
```

Quick Test Entry (Project):

```json
{
  "title": "Test Project",
  "slug": "test-project",
  "shortDescription": "Test project description",
  "category": "ongoing",
  "status": "Testing",
  "progress": 50,
  "startDate": "2024-01-01",
  "endDate": "2024-12-31",
  "isFeatured": false
}
```

Last Updated: [Date]

Maintained By: [Your Name]

Next Scheduled Review: [Date]