Pana Sports v1.0 - Architecture Documentation

> **Version:** 1.0
> **Last Updated:** December 6, 2025
> **Status:** Production Ready

Table of Contents

1. Overview
2. Technology Stack
3. Project Structure
4. Architecture Layers
    - Database Layer (Supabase)
    - Schema Layer (Zod Validation)
    - API Layer (Next.js Route Handlers)
    - Data Fetching Layer (TanStack Query Hooks)
    - UI Component Layer
5. Route Architecture
    - Public Routes
    - CMS Routes
    - API Routes
6. Authentication & Authorization
7. Component Templates
8. Styling Architecture
9. Best Practices & Patterns

Overview

Pana Sports is a comprehensive Ethiopian football hub built with Next.js 16, featuring a public-facing sports portal and an admin CMS for content management. The application follows a modular, template-based architecture that ensures consistency and maintainability across all features.

Key Features

- **Multi-language support** (English/Amharic)
- **Real-time match tracking** with live scores
- **League management** (Premier League, Women's League, Ethiopian Cup, etc.)
- **Team & player profiles**
- **News and content management**
- **Standings and statistics**
- **Admin CMS dashboard**

Technology Stack

Category	Technology	Version
Framework	Next.js	16.0.3
Runtime	React	19.2.0
Language	TypeScript	5.x
Database	Supabase (PostgreSQL)	-
State Management	TanStack Query	5.90.9
Form Handling	React Hook Form	7.66.1
Validation	Zod	4.1.13
Styling	Tailwind CSS	4.x
UI Components	Radix UI	Various
Rich Text Editor	TipTap	3.11.x
Animations	Framer Motion	12.23.24
Icons	Lucide React	0.553.0
Notifications	Sonner	2.0.7

Project Structure

```
pana-sports/
├── app/                          # Next.js App Router
│   ├── api/                      # API Route Handlers
│   │   ├── public/               # Public API endpoints
│   │   │   ├── leagues/          # League data endpoints
│   │   │   ├── matches/          # Match data endpoints
│   │   │   ├── teams/            # Team data endpoints
│   │   │   ├── players/          # Player data endpoints
│   │   │   ├── standings/        # Standings endpoints
│   │   │   └── top-scorers/      # Top scorers endpoints
│   │   ├── teams/                # CMS Team CRUD endpoints
│   │   ├── players/              # CMS Player CRUD endpoints
│   │   ├── matches/              # CMS Match CRUD endpoints
│   │   ├── leagues/              # CMS League CRUD endpoints
│   │   ├── standings/            # CMS Standings CRUD endpoints
│   │   ├── top-scorers/          # CMS Top Scorers CRUD endpoints
│   │   ├── news/                 # CMS News CRUD endpoints
│   │   ├── authors/              # CMS Authors CRUD endpoints
│   │   ├── upload/               # File upload endpoint
│   │   └── ...                   # Other API routes
│   ├── cms/                      # Admin CMS Routes
│   │   ├── layout.tsx            # CMS Layout with Sidebar
│   │   ├── dashboard/            # Dashboard page
│   │   ├── teams/                # Team management
│   │   │   ├── page.tsx          # Team list
```

```
│   │   │   ├── create/           # Create team
│   │   │   └── [id]/edit/       # Edit team
│   │   ├── players/             # Player management
│   │   ├── matches/             # Match management
│   │   ├── leagues/             # League management
│   │   ├── standings/           # Standings management
│   │   ├── top-scorers/         # Top scorers management
│   │   ├── news/                # News management
│   │   └── ...                  # Other CMS routes
│   ├── premier-league/          # Premier League public page
│   ├── womens-league/           # Women's League public page
│   ├── ethiopian-cup/           # Ethiopian Cup public page
│   ├── higher-league/           # Higher League public page
│   ├── live/                    # Live matches page
│   ├── matches/                 # All matches page
│   ├── teams/                   # Public teams pages
│   ├── players/                 # Public player pages
│   ├── news/                    # News pages
│   ├── layout.tsx               # Root layout
│   ├── page.tsx                 # Homepage
│   └── global.css               # Global styles
├── components/                  # React Components
│   ├── cms/                     # CMS-specific components
│   │   ├── layout/              # Sidebar, Header
│   │   ├── teams/               # TeamTable, TeamForm
│   │   ├── players/             # PlayerTable, PlayerForm
│   │   ├── matches/             # MatchTable, MatchForm
│   │   ├── leagues/             # LeagueTable, LeagueForm
│   │   ├── standings/           # StandingsTable, StandingsForm
│   │   ├── top-scorers/         # TopScorersTable, TopScorersForm
│   │   ├── news/                # NewsTable, NewsForm
│   │   └── ...                  # Other CMS components
│   ├── shared/                  # Shared components
│   │   ├── tabs/                # Reusable tab components
│   │   │   ├── MatchesTab.tsx
│   │   │   ├── OverviewTab.tsx
│   │   │   ├── TableTab.tsx
│   │   │   └── TeamsTab.tsx
│   │   ├── Skeletons/           # Loading skeleton components
│   │   ├── navbar.tsx           # Public navigation
│   │   ├── Footer.tsx           # Public footer
│   │   ├── AppShell.tsx         # App wrapper
│   │   └── ...                  # Other shared components
│   ├── premier-league/          # Premier League components
│   ├── womens-league/           # Women's League components
│   ├── ethiopian-cup/           # Ethiopian Cup components
│   ├── news/                    # News components
│   ├── players/                 # Player components
│   ├── teams/                   # Team components
│   ├── providers/               # React context providers
│   │   └── query-client-provider.tsx
│   └── ui/                      # Base UI components (Radix)
│       ├── button.tsx
│       ├── card.tsx
```

```
│       ├── dialog.tsx
│       ├── form.tsx
│       ├── input.tsx
│       ├── select.tsx
│       ├── table.tsx
│       ├── tabs.tsx
│       └── ...                    # Other UI primitives
├── lib/                           # Utilities and configurations
│   ├── hooks/                     # Custom React hooks
│   │   ├── cms/                   # CMS data hooks (CRUD)
│   │   │   ├── useTeams.ts
│   │   │   ├── usePlayers.ts
│   │   │   ├── useMatches.ts
│   │   │   ├── useLeagues.ts
│   │   │   ├── useStandings.ts
│   │   │   ├── useTopScorers.ts
│   │   │   ├── useNews.ts
│   │   │   └── ...
│   │   └── public/                # Public data hooks (read-only)
│   │       ├── useLeagues.ts
│   │       ├── useMatches.ts
│   │       ├── useTeamDetail.ts
│   │       ├── usePlayerDetail.ts
│   │       ├── useStandings.ts
│   │       └── useTopScorers.ts
│   ├── schemas/                   # Zod validation schemas
│   │   ├── team.ts
│   │   ├── player.ts
│   │   ├── match.ts
│   │   ├── league.ts
│   │   ├── standing.ts
│   │   ├── topScorer.ts
│   │   ├── news.ts
│   │   └── ...
│   ├── supabase/                  # Supabase client configurations
│   │   ├── client.ts              # Browser client
│   │   ├── server.ts              # Server client
│   │   └── middleware.ts          # Auth middleware
│   ├── auth.ts                    # Authentication utilities
│   ├── auth.server.ts             # Server-side auth
│   └── utils.ts                   # General utilities (cn, etc.)
└── public/                        # Static assets
    ├── logo.png
    ├── logo2.png
    └── ...
```

Architecture Layers

The application follows a **5-layer architecture** that ensures separation of concerns and maintainability:

```
┌─────────────────────────────────────────────────────────────┐
│                    UI COMPONENT LAYER                       │
│         (React Components, Pages, Layouts)                  │
├─────────────────────────────────────────────────────────────┤
│                  DATA FETCHING LAYER                        │
│           (TanStack Query Hooks - useTeams, etc.)           │
├─────────────────────────────────────────────────────────────┤
│                     API LAYER                               │
│         (Next.js Route Handlers - /api/*)                   │
├─────────────────────────────────────────────────────────────┤
│                   SCHEMA LAYER                              │
│            (Zod Validation Schemas)                         │
├─────────────────────────────────────────────────────────────┤
│                  DATABASE LAYER                             │
│               (Supabase PostgreSQL)                         │
└─────────────────────────────────────────────────────────────┘
```

Database Layer (Supabase)

The database layer uses **Supabase** with two client configurations:

Server Client (`lib/supabase/server.ts`)

```typescript
import { createServerClient } from "@supabase/ssr";
import { cookies } from "next/headers";

export async function createClient() {
  const cookieStore = await cookies();
  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!, // Service role for admin ops
    {
      cookies: {
        getAll() { return cookieStore.getAll(); },
        setAll(cookiesToSet) {
          cookiesToSet.forEach(({ name, value }) =>
            cookieStore.set(name, value)
          );
        },
      },
    }
  );
}
```

Browser Client (`lib/supabase/client.ts`)

```
import { createBrowserClient } from '@supabase/ssr'

export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  )
}
```

Schema Layer (Zod Validation)

Each entity has a dedicated schema file in `lib/schemas/` with three schema types:

Schema Template (`lib/schemas/[entity].ts`)

```
import { z } from "zod";

// 1. Database Entity Schema (full database record)
export const entitySchema = z.object({
  id: z.string().uuid(),
  slug: z.string(),
  name_en: z.string(),
  name_am: z.string(),
  // ... other fields
  created_at: z.string().datetime(),
  updated_at: z.string().datetime(),
  created_by: z.string().uuid().nullable(),
  is_active: z.boolean().default(true),
});

// 2. Create Input Schema (for POST requests)
export const createInputSchema = z.object({
  slug: z.string().min(1, "Slug is required"),
  name_en: z.string().min(1, "English name is required"),
  name_am: z.string().min(1, "Amharic name is required"),
  // ... required and optional fields with validation
});

// 3. Update Input Schema (partial of create)
export const updateInputSchema = createInputSchema.partial();

// 4. Schema with Relations (for joined queries)
export const entityWithRelationsSchema = entitySchema.extend({
  relation: z.object({ /* ... */ }).optional(),
});

// Type exports
export type Entity = z.infer<typeof entitySchema>;
```

```
export type CreateEntity = z.infer<typeof createInputSchema>;
export type UpdateEntity = z.infer<typeof updateInputSchema>;
```

Example: Team Schema

```
// lib/schemas/team.ts
export const createTeamInputSchema = z.object({
  slug: z.string().min(1).max(50),
  name_en: z.string().min(1).max(100),
  name_am: z.string().min(1).max(100),
  short_name_en: z.string().optional(),
  short_name_am: z.string().optional(),
  league_id: z.string().uuid("League is required"),
  logo_url: z.string().url("Invalid URL").optional(),
  description_en: z.string().optional(),
  description_am: z.string().optional(),
  stadium_en: z.string().optional(),
  stadium_am: z.string().optional(),
  founded: z.number().optional(),
  is_active: z.boolean().optional(),
});
```

API Layer (Next.js Route Handlers)

The API layer follows two distinct patterns:

CMS API Routes (Full CRUD)

Located in `app/api/[entity]/` with authenticated mutations:

Collection Route (`route.ts`):

```
// app/api/teams/route.ts
import { createClient } from "@/lib/supabase/server";
import { createTeamInputSchema } from "@/lib/schemas/team";
import { requireAdmin } from "@/lib/auth";
import { NextResponse } from "next/server";

// GET - List all (no auth required)
export async function GET() {
  const supabase = await createClient();
  const { data, error } = await supabase
    .from("teams")
    .select(`*, league:leagues(id, name_en, name_am, slug, category)`)
    .order("created_at", { ascending: false });

  if (error) return NextResponse.json({ error: error.message }, { status: 500 });
  return NextResponse.json(data);
```

```
}

// POST - Create (admin auth required)
export async function POST(request: Request) {
  const user = await requireAdmin(); // Throws if not admin
  const body = await request.json();
  const validatedData = createTeamInputSchema.parse(body);

  const supabase = await createClient();
  const { data, error } = await supabase
    .from("teams")
    .insert({ ...validatedData, created_by: user.id })
    .select()
    .single();

  if (error) return NextResponse.json({ error: error.message }, { status: 500 });
  return NextResponse.json(data, { status: 201 });
}
```

Individual Route (`[id]/route.ts`):

```
// app/api/teams/[id]/route.ts

// GET - Single item
export async function GET(request: Request, { params }: { params: Promise<{ id:
string }> }) {
  const { id } = await params;
  const supabase = await createClient();
  const { data, error } = await supabase
    .from("teams")
    .select("*")
    .eq("id", id)
    .single();
  // ...
}

// PATCH - Update (admin auth required)
export async function PATCH(request: Request, { params }: { params: Promise<{ id:
string }> }) {
  const { id } = await params;
  await requireAdmin();
  const body = await request.json();
  const validatedData = updateTeamInputSchema.parse(body);
  // ...
}

// DELETE - Remove (admin auth required)
export async function DELETE(request: Request, { params }: { params: Promise<{ id:
string }> }) {
  const { id } = await params;
  await requireAdmin();
  // Check for dependencies before deletion
```

```
  // ...
}
```

Public API Routes (Read-Only)

Located in `app/api/public/[entity]/` for public consumption:

```
// app/api/public/leagues/[id]/route.ts
export async function GET(request: Request, { params }: { params: Promise<{ id:
string }> }) {
  const { id } = await params;
  const supabase = await createClient();

  const { data: league, error } = await supabase
    .from("leagues")
    .select(`
      *,
      teams:teams(id, name_en, name_am, slug, logo_url),
      standings:standings(id, team_id, season, played, won, draw, lost, ...),
      matches:matches(id, home_team_id, away_team_id, date, status, ...)
    `)
    .eq("id", id)
    .single();

  if (error) return NextResponse.json({ error: error.message }, { status: 500 });
  return NextResponse.json(league);
}
```

Data Fetching Layer (TanStack Query Hooks)

Custom hooks wrap API calls with TanStack Query for caching and state management.

CMS Hooks Pattern (`lib/hooks/cms/use[Entity].ts`)

```
// lib/hooks/cms/useTeams.ts
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query'
import { Team, CreateTeam, UpdateTeam } from '@/lib/schemas/team'

// API Helpers
async function fetchTeams() {
  const res = await fetch('/api/teams')
  if (!res.ok) throw new Error('Failed to fetch teams')
  return res.json() as Promise<Team[]>
}

async function createTeam(data: CreateTeam) {
  const res = await fetch('/api/teams', {
```

```
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data),
  })
  if (!res.ok) throw new Error('Failed to create team')
  return res.json() as Promise<Team>
}

// Query Hooks
export function useTeams() {
  return useQuery({
    queryKey: ['teams'],
    queryFn: fetchTeams,
  })
}

export function useTeam(id: string) {
  return useQuery({
    queryKey: ['teams', id],
    queryFn: () => fetchTeam(id),
    enabled: !!id,
  })
}

// Mutation Hooks
export function useCreateTeam() {
  const queryClient = useQueryClient()
  return useMutation({
    mutationFn: createTeam,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['teams'] })
    },
  })
}

export function useUpdateTeam() {
  const queryClient = useQueryClient()
  return useMutation({
    mutationFn: ({ id, updates }: { id: string; updates: UpdateTeam }) =>
      updateTeam(id, updates),
    onSuccess: (_, { id }) => {
      queryClient.invalidateQueries({ queryKey: ['teams'] })
      queryClient.invalidateQueries({ queryKey: ['teams', id] })
    },
  })
}

export function useDeleteTeam() {
  const queryClient = useQueryClient()
  return useMutation({
    mutationFn: deleteTeam,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['teams'] })
    },
```

```
  })
}
```

Public Hooks Pattern (`lib/hooks/public/use[Entity].ts`)

```
// lib/hooks/public/useLeagues.ts
export function useLeague(id: string) {
  return useQuery({
    queryKey: ['league', id],
    queryFn: () => fetch(`/api/public/leagues/${id}`).then(r => r.json()),
    enabled: !!id,
  })
}

export function useLeagueMatches(id: string, params?: { matchday?: string }) {
  return useQuery({
    queryKey: ['league-matches', id, params],
    queryFn: () => fetchLeagueMatches(id, params),
    enabled: !!id,
  })
}

export function useLeagueStandings(id: string) {
  return useQuery({
    queryKey: ['league-standings', id],
    queryFn: () => fetchLeagueStandings(id),
    enabled: !!id,
  })
}

export function useLeagueTeams(id: string) {
  return useQuery({
    queryKey: ['league-teams', id],
    queryFn: () => fetchLeagueTeams(id),
    enabled: !!id,
  })
}
```

Query Client Provider

```
// components/providers/query-client-provider.tsx
"use client";

import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { ReactQueryDevtools } from "@tanstack/react-query-devtools";
import { useState } from "react";

export default function QueryClientProvider({ children }: { children:
```

```
  React.ReactNode }) {
    const [queryClient] = useState(() => new QueryClient({
      defaultOptions: {
        queries: {
          staleTime: 60 * 1000, // 1 minute
          refetchOnWindowFocus: false,
        },
      },
    }));

    return (
      <RQQueryClientProvider client={queryClient}>
        {children}
        <ReactQueryDevtools initialIsOpen={false} />
      </RQQueryClientProvider>
    );
  }
```

UI Component Layer

Component Categories

Category	Location	Purpose
UI Primitives	`components/ui/`	Base Radix UI components
CMS Components	`components/cms/`	Admin dashboard components
Shared Components	`components/shared/`	Reusable across public/CMS
Feature Components	`components/[feature]/`	Feature-specific components

Route Architecture

Public Routes

Route	Description	Component
`/`	Homepage	`app/page.tsx`
`/premier-league`	Premier League page	`PremierLeaguePage`
`/womens-league`	Women's League page	`WomensLeaguePage`
`/ethiopian-cup`	Ethiopian Cup page	`EthiopianCupPage`
`/higher-league`	Higher League page	`HigherLeaguePage`
`/live`	Live matches	`LiveMatchesPage`
`/matches`	All matches	`MatchesPage`
`/teams/[id]`	Team detail	`TeamDetailPage`

Route	Description	Component
`/players/[id]`	Player detail	`PlayerDetailPage`
`/news`	News listing	`NewsPage`
`/news/[slug]`	News detail	`NewsDetail`

CMS Routes

Route	Description	Component
`/cms/dashboard`	Admin dashboard	Dashboard page
`/cms/teams`	Team list	`TeamTable`
`/cms/teams/create`	Create team	`TeamForm`
`/cms/teams/[id]/edit`	Edit team	`TeamForm`
`/cms/players`	Player list	`PlayerTable`
`/cms/matches`	Match list	`MatchTable`
`/cms/leagues`	League list	`LeagueTable`
`/cms/standings`	Standings	`StandingsTable`
`/cms/top-scorers`	Top scorers	`TopScorersTable`
`/cms/news`	News list	`NewsTable`

Authentication & Authorization

Auth Utilities (`lib/auth.ts`)

```
import { createClient } from "./supabase/server";
import { redirect } from "next/navigation";

export async function getSession() {
  const supabase = await createClient();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return null;
  const { data: { session } } = await supabase.auth.getSession();
  return session;
}

export async function getCurrentUser() {
  const session = await getSession();
  return session?.user || null;
}

export async function isAdmin() {
  const user = await getCurrentUser();
```

```
  return user?.user_metadata?.role === "admin";
}

export async function requireAdmin() {
  const supabase = await createClient();
  const { data: { user }, error } = await supabase.auth.getUser();

  if (error || !user) redirect("/login");
  if (user.user_metadata?.role !== "admin") redirect("/unauthorized");

  return user;
}
```

Component Templates

CMS Table Component Template

```
// components/cms/[entity]/[Entity]Table.tsx
"use client";

import { useState } from "react";
import { useEntities, useDeleteEntity } from "@/lib/hooks/cms/useEntities";
import { Card, CardHeader, CardTitle, CardContent } from "@/components/ui/card";
import { Table, TableHeader, TableRow, TableHead, TableBody, TableCell } from
"@/components/ui/table";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { AlertDialog, ... } from "@/components/ui/alert-dialog";
import { toast } from "sonner";
import Link from "next/link";

export default function EntityTable() {
  const [searchTerm, setSearchTerm] = useState("");
  const { data: entities, isLoading, error } = useEntities();
  const deleteEntityMutation = useDeleteEntity();

  const filteredEntities = entities?.filter(e =>
    e.name_en.toLowerCase().includes(searchTerm.toLowerCase())
  ) || [];

  const handleDelete = async (id: string, name: string) => {
    toast.promise(deleteEntityMutation.mutateAsync(id), {
      loading: `Deleting "${name}"...`,
      success: `"${name}" deleted successfully`,
      error: (err) => err.message || "Failed to delete",
    });
  };

  if (isLoading) return <LoadingSpinner />;
  if (error) return <ErrorState error={error} />;
```

```
  return (
    <div className="space-y-6">
      {/* Stats Cards */}
      <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
        <StatCard title="Total" value={entities?.length || 0} />
        {/* ... more stats */}
      </div>

      {/* Main Table Card */}
      <Card>
        <CardHeader>
          <div className="flex justify-between items-center">
            <CardTitle>Entities</CardTitle>
            <div className="flex gap-3">
              <Input placeholder="Search..." value={searchTerm} onChange={...} />
              <Link href="/cms/entities/create">
                <Button>Add Entity</Button>
              </Link>
            </div>
          </div>
        </CardHeader>
        <CardContent>
          {/* Desktop Table */}
          <div className="hidden lg:block">
            <Table>...</Table>
          </div>
          {/* Mobile Card View */}
          <div className="lg:hidden">...</div>
        </CardContent>
      </Card>
    </div>
  );
}
```

CMS Form Component Template

```
// components/cms/[entity]/[Entity]Form.tsx
"use client";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { toast } from "sonner";
import { Form, FormField, FormItem, FormLabel, FormControl, FormMessage } from
"@/components/ui/form";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { createEntityInputSchema, Entity, CreateEntity } from
"@/lib/schemas/entity";
import { useCreateEntity, useUpdateEntity } from "@/lib/hooks/cms/useEntities";
```

```tsx
interface EntityFormProps {
  entity?: Entity;
  onSuccess?: () => void;
  onCancel?: () => void;
}

export default function EntityForm({ entity, onSuccess, onCancel }:
EntityFormProps) {
  const isEditing = !!entity;

  const form = useForm<CreateEntity>({
    resolver: zodResolver(createEntityInputSchema),
    defaultValues: {
      name_en: entity?.name_en || "",
      name_am: entity?.name_am || "",
      // ... other fields
    },
  });

  const createMutation = useCreateEntity();
  const updateMutation = useUpdateEntity();

  const onSubmit = async (data: CreateEntity) => {
    const promise = isEditing
      ? updateMutation.mutateAsync({ id: entity.id, updates: data })
      : createMutation.mutateAsync(data);

    toast.promise(promise, {
      loading: isEditing ? "Updating..." : "Creating...",
      success: (result) => `"${result.name_en}" ${isEditing ? 'updated' :
'created'}`,
      error: (err) => err.message,
    });

    await promise;
    onSuccess?.();
  };

  return (
    <Card>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)}>
          <CardContent>
            <Tabs defaultValue="english">
              <TabsList>
                <TabsTrigger value="english">English</TabsTrigger>
                <TabsTrigger value="amharic">Amharic</TabsTrigger>
              </TabsList>

              <TabsContent value="english">
                <FormField
                  control={form.control}
                  name="name_en"
```

```
                  render={({ field }) => (
                    <FormItem>
                      <FormLabel>Name (English)</FormLabel>
                      <FormControl>
                        <Input {...field} />
                      </FormControl>
                      <FormMessage />
                    </FormItem>
                  )}
                />
                {/* ... more fields */}
              </TabsContent>

              <TabsContent value="amharic">
                {/* Amharic fields */}
              </TabsContent>
            </Tabs>
          </CardContent>

          <CardFooter>
            <Button type="button" variant="outline" onClick=
{onCancel}>Cancel</Button>
            <Button type="submit" disabled={form.formState.isSubmitting}>
              {isEditing ? "Update" : "Create"}
            </Button>
          </CardFooter>
        </form>
      </Form>
    </Card>
  );
}
```

Shared Tab Component Template

```
// components/shared/tabs/[Feature]Tab.tsx
"use client";

import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { useFeatureData } from "@/lib/hooks/public/useFeatureData";

interface FeatureTabProps {
  leagueId: string;
}

export default function FeatureTab({ leagueId }: FeatureTabProps) {
  const { data, isLoading, error } = useFeatureData(leagueId);

  if (isLoading) return <FeatureSkeleton />;
  if (error) return <ErrorState error={error} />;

  return (
```

```
      <div className="space-y-6">
        {/* Tab content */}
      </div>
    );
  }
```

Styling Architecture

Theme System

The app uses a **dark theme by default** with a **light theme for CMS**:

```
/* Dark theme (default) */
:root {
  --background: oklch(0.09 0.01 285.823);
  --foreground: oklch(0.985 0 0);
  --primary: oklch(0.76 0.12 65); /* Sunlit Clay - Brand Color */
  /* ... */
}

/* CMS Light Theme */
.cms-light-theme {
  --background: oklch(0.985 0 0);
  --foreground: oklch(0.09 0.01 285.823);
  --primary: oklch(0.68 0.13 65);
  /* ... */
}
```

Custom Utility Classes

```
/* Pana-specific utilities */
.btn-pana      /* Branded button style */
.card-pana     /* Branded card style */
.bg-pana-gradient     /* Brand gradient background */
.text-pana-gradient   /* Brand text gradient */
.glass-zinc    /* Glassmorphic effect */
.hide-scrollbar /* Hidden scrollbar */
```

Best Practices & Patterns

1. Bilingual Content

All content entities support English (`_en`) and Amharic (`_am`) fields.

2. Slug Generation

Auto-generate slugs from English names for URL-friendly identifiers.

3. Mobile-First Design

All components implement responsive designs with separate mobile/desktop views.

4. Error Handling

Consistent error handling with toast notifications using Sonner.

5. Loading States

Skeleton components for all data-fetching components.

6. Cache Invalidation

TanStack Query invalidation on mutations for consistent UI state.

7. Type Safety

End-to-end TypeScript with Zod validation for runtime safety.

Version History

Version	Date	Changes
1.0	Dec 2025	Initial release
1.1	TBD	Upcoming features

This documentation is maintained alongside the codebase. For updates, contributions, or issues, please refer to the project repository.