Pavilion360 V2.0 - Architecture Documentation

> **Version:** 2.0.0
> **Last Updated:** December 18, 2024
> **Status:** Planning Phase

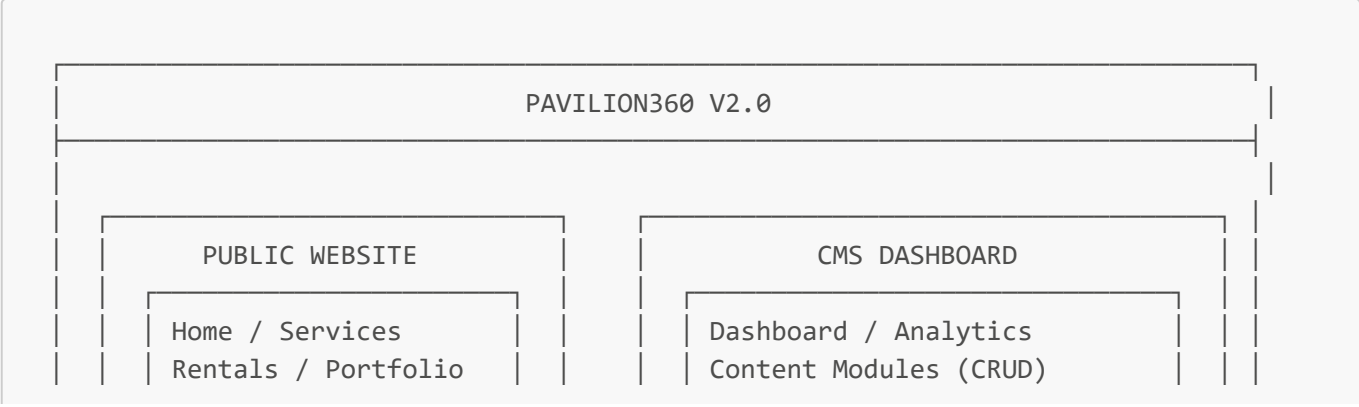
Table of Contents

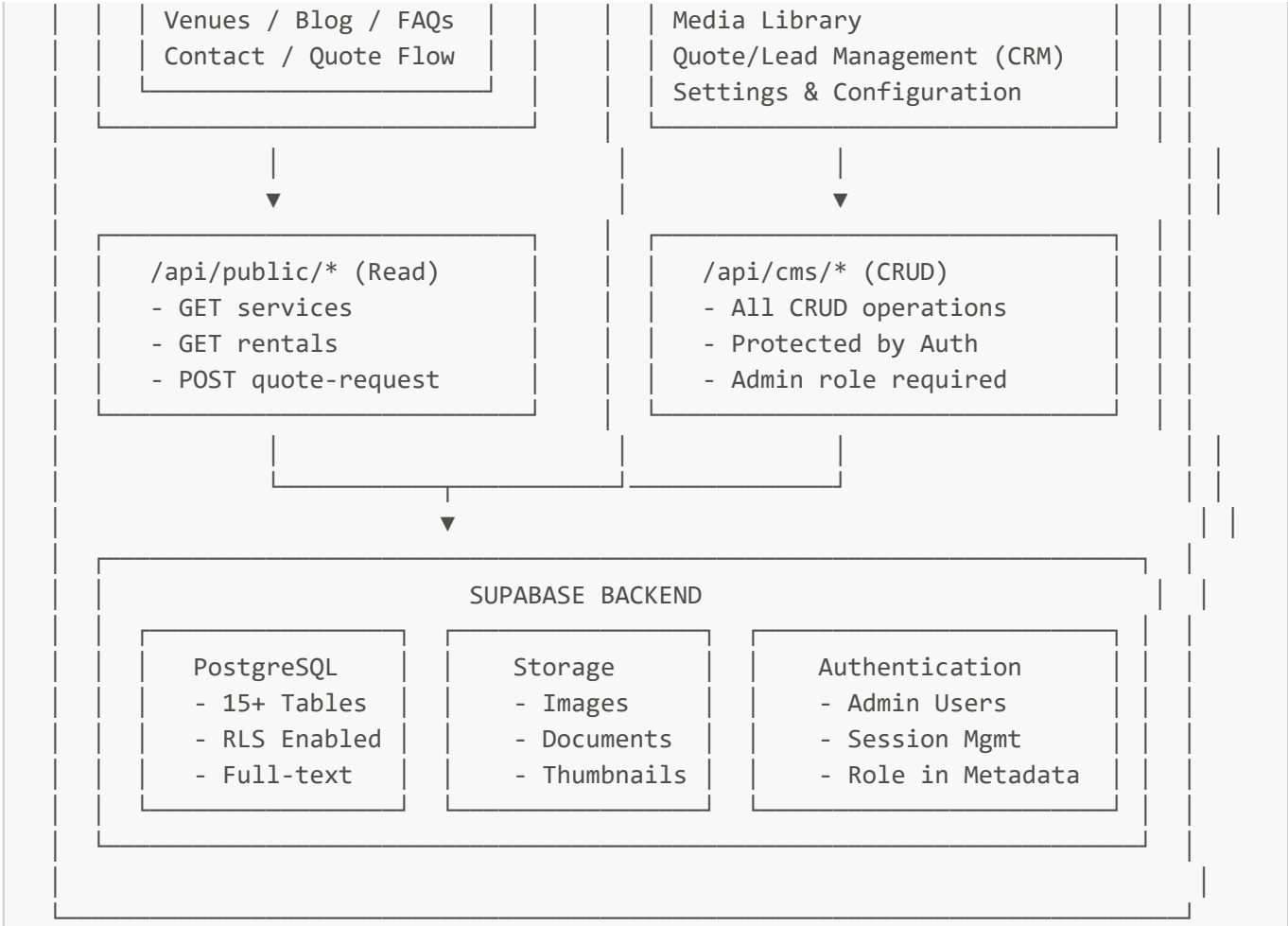
Overview

Pavilion360 V2.0 transforms the existing static marketing website into a fully data-driven platform with an integrated Content Management System (CMS). The architecture emphasizes:

- **Data-Driven Content:** All content managed via Supabase PostgreSQL
- **Premium CMS Interface:** Mobile-first, animated, templated design
- **Modular Architecture:** Clear separation between public site and CMS
- **Type Safety:** End-to-end TypeScript with Zod validation
- **Performance:** Optimized queries with TanStack Query caching
- **Lightweight CRM:** Quote request management integrated with contact flow

High-Level Architecture Diagram





Technology Stack

Core Framework

Technology	Version	Purpose
Next.js	16.x	React framework with App Router
React	19.x	UI library
TypeScript	5.x	Type safety

Backend & Database

Technology	Version	Purpose
Supabase	Latest	Backend-as-a-Service
PostgreSQL	15.x	Primary database (via Supabase)
Supabase Auth	Latest	Authentication
Supabase Storage	Latest	File/image storage

Data & State Management

Technology	Version	Purpose
TanStack Query	5.x	Server state management & caching
React Hook Form	7.x	Form state management
Zod	3.x	Schema validation

UI & Styling

Technology	Version	Purpose
Tailwind CSS	4.x	Utility-first CSS
Radix UI	Latest	Accessible primitives
Framer Motion	12.x	Animations
Lucide React	Latest	Icons
Sonner	1.x	Toast notifications
TipTap	Latest	Rich text editor

Development Tools

Technology	Purpose
ESLint	Code linting
Prettier	Code formatting
pnpm	Package manager

System Architecture

Directory Structure

```
pavilion360/
├── app/
│   ├── (public)/                    # Public routes (grouped)
│   │   ├── page.tsx                 # Home
│   │   ├── about/
│   │   ├── services/
│   │   │   ├── page.tsx
│   │   │   └── [slug]/page.tsx
│   │   ├── rentals/
│   │   │   ├── page.tsx
│   │   │   └── [slug]/page.tsx
│   │   ├── portfolio/
│   │   │   ├── page.tsx
│   │   │   └── [slug]/page.tsx
│   │   ├── venues/
```

```
│   │   ├── blog/
│   │   │   ├── page.tsx
│   │   │   └── [slug]/page.tsx
│   │   ├── faqs/
│   │   ├── resources/
│   │   └── contact/
│   │
│   ├── (cms)/                      # CMS routes (protected)
│   │   ├── layout.tsx              # CMS shell layout
│   │   ├── cms/
│   │   │   ├── page.tsx            # Dashboard
│   │   │   ├── services/
│   │   │   │   ├── page.tsx        # List
│   │   │   │   ├── new/page.tsx    # Create
│   │   │   │   └── [id]/
│   │   │   │       ├── page.tsx    # Detail/View
│   │   │   │       └── edit/page.tsx # Edit
│   │   │   ├── rentals/
│   │   │   ├── portfolio/
│   │   │   ├── venues/
│   │   │   ├── testimonials/
│   │   │   ├── faqs/
│   │   │   ├── blog/
│   │   │   ├── team/
│   │   │   ├── quotes/             # CRM - Quote requests
│   │   │   ├── inquiries/          # CRM - Contact form submissions
│   │   │   ├── media/              # Media library
│   │   │   └── settings/
│   │   │       ├── page.tsx
│   │   │       └── site/page.tsx   # Site-wide settings
│   │   └── login/page.tsx          # CMS Login
│   │
│   ├── api/
│   │   ├── public/                 # Public API (read-mostly)
│   │   │   ├── services/
│   │   │   │   ├── route.ts        # GET all
│   │   │   │   └── [slug]/route.ts  # GET by slug
│   │   │   ├── rentals/
│   │   │   ├── portfolio/
│   │   │   ├── venues/
│   │   │   ├── testimonials/
│   │   │   ├── faqs/
│   │   │   ├── blog/
│   │   │   ├── quote-request/      # POST quote requests
│   │   │   │   └── route.ts
│   │   │   ├── contact/            # POST contact form
│   │   │   │   └── route.ts
│   │   │   └── analytics/          # POST view tracking
│   │   │       └── track/route.ts
│   │   │
│   │   └── cms/                    # Protected CMS API (full CRUD)
│   │       ├── services/
│   │       │   ├── route.ts        # GET all, POST create
│   │       │   └── [id]/route.ts   # GET, PUT, DELETE
```

```
│   │       ├── rentals/
│   │       ├── portfolio/
│   │       ├── venues/
│   │       ├── testimonials/
│   │       ├── faqs/
│   │       ├── blog/
│   │       ├── team/
│   │       ├── quotes/
│   │       ├── inquiries/
│   │       ├── media/
│   │       │   ├── route.ts          # Upload, list
│   │       │   └── [id]/route.ts     # Delete
│   │       ├── settings/
│   │       └── analytics/
│   │           └── route.ts          # GET dashboard stats
│   │
│   ├── proxy.ts                      # Auth middleware (Next.js 16 convention)
│   ├── layout.tsx                    # Root layout
│   └── globals.css
│
├── components/
│   ├── public/                       # Public site components
│   │   ├── services/
│   │   │   ├── service-card.tsx
│   │   │   ├── service-detail.tsx
│   │   │   └── service-grid.tsx
│   │   ├── rentals/
│   │   │   ├── rental-card.tsx
│   │   │   ├── rental-filters.tsx
│   │   │   ├── rental-grid.tsx
│   │   │   └── quote-basket.tsx
│   │   ├── portfolio/
│   │   ├── venues/
│   │   ├── blog/
│   │   ├── faqs/
│   │   ├── testimonials/
│   │   └── shared/
│   │       ├── hero-section.tsx
│   │       ├── cta-button.tsx
│   │       ├── instagram-feed.tsx
│   │       └── page-skeleton.tsx
│   │
│   ├── cms/                          # CMS components
│   │   ├── layout/
│   │   │   ├── cms-shell.tsx         # Main layout wrapper
│   │   │   ├── cms-header.tsx        # Top navigation
│   │   │   ├── cms-sidebar.tsx       # Side navigation
│   │   │   └── cms-breadcrumb.tsx
│   │   ├── dashboard/
│   │   │   ├── stats-card.tsx
│   │   │   ├── recent-activity.tsx
│   │   │   └── quick-actions.tsx
│   │   ├── data-table/
│   │   │   ├── data-table.tsx        # Reusable table component
```

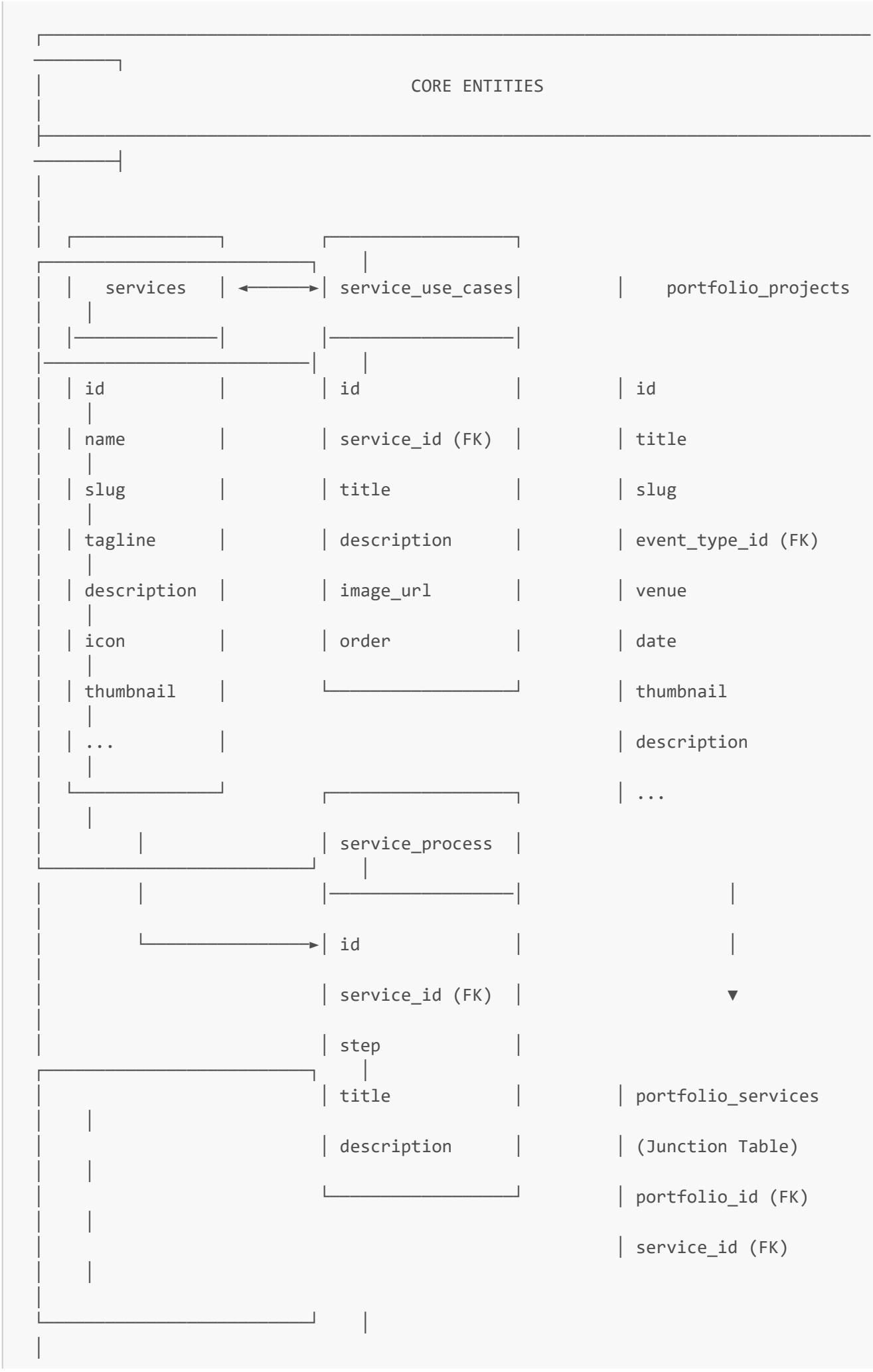
```
│   │   │   ├── data-table-toolbar.tsx
│   │   │   ├── data-table-pagination.tsx
│   │   │   ├── data-table-column-header.tsx
│   │   │   └── data-table-row-actions.tsx
│   │   ├── forms/
│   │   │   ├── form-field.tsx       # Reusable form field wrapper
│   │   │   ├── image-upload.tsx     # Image upload component
│   │   │   ├── multi-image-upload.tsx
│   │   │   ├── rich-text-editor.tsx # TipTap integration
│   │   │   ├── slug-input.tsx       # Auto-slug generation
│   │   │   ├── tag-input.tsx        # Tag management
│   │   │   └── searchable-select.tsx
│   │   ├── shared/
│   │   │   ├── empty-state.tsx
│   │   │   ├── loading-skeleton.tsx
│   │   │   ├── confirm-dialog.tsx
│   │   │   ├── status-badge.tsx
│   │   │   └── page-header.tsx
│   │   └── modules/                 # Module-specific components
│   │       ├── services/
│   │       ├── rentals/
│   │       ├── portfolio/
│   │       ├── blog/
│   │       └── quotes/
│   │
│   ├── layout/                      # Site-wide layouts
│   │   ├── site-header.tsx
│   │   └── site-footer.tsx
│   │
│   └── ui/                          # Radix UI primitives
│       ├── accordion.tsx
│       ├── button.tsx
│       ├── card.tsx
│       ├── dialog.tsx
│       ├── dropdown-menu.tsx
│       ├── input.tsx
│       ├── label.tsx
│       ├── select.tsx
│       ├── sheet.tsx
│       ├── skeleton.tsx
│       ├── table.tsx
│       ├── tabs.tsx
│       ├── textarea.tsx
│       ├── toast.tsx
│       └── tooltip.tsx
│
├── hooks/
│   ├── public/                      # Public data hooks
│   │   ├── use-services.ts
│   │   ├── use-service.ts
│   │   ├── use-rentals.ts
│   │   ├── use-rental.ts
│   │   ├── use-portfolio.ts
│   │   ├── use-portfolio-project.ts
```

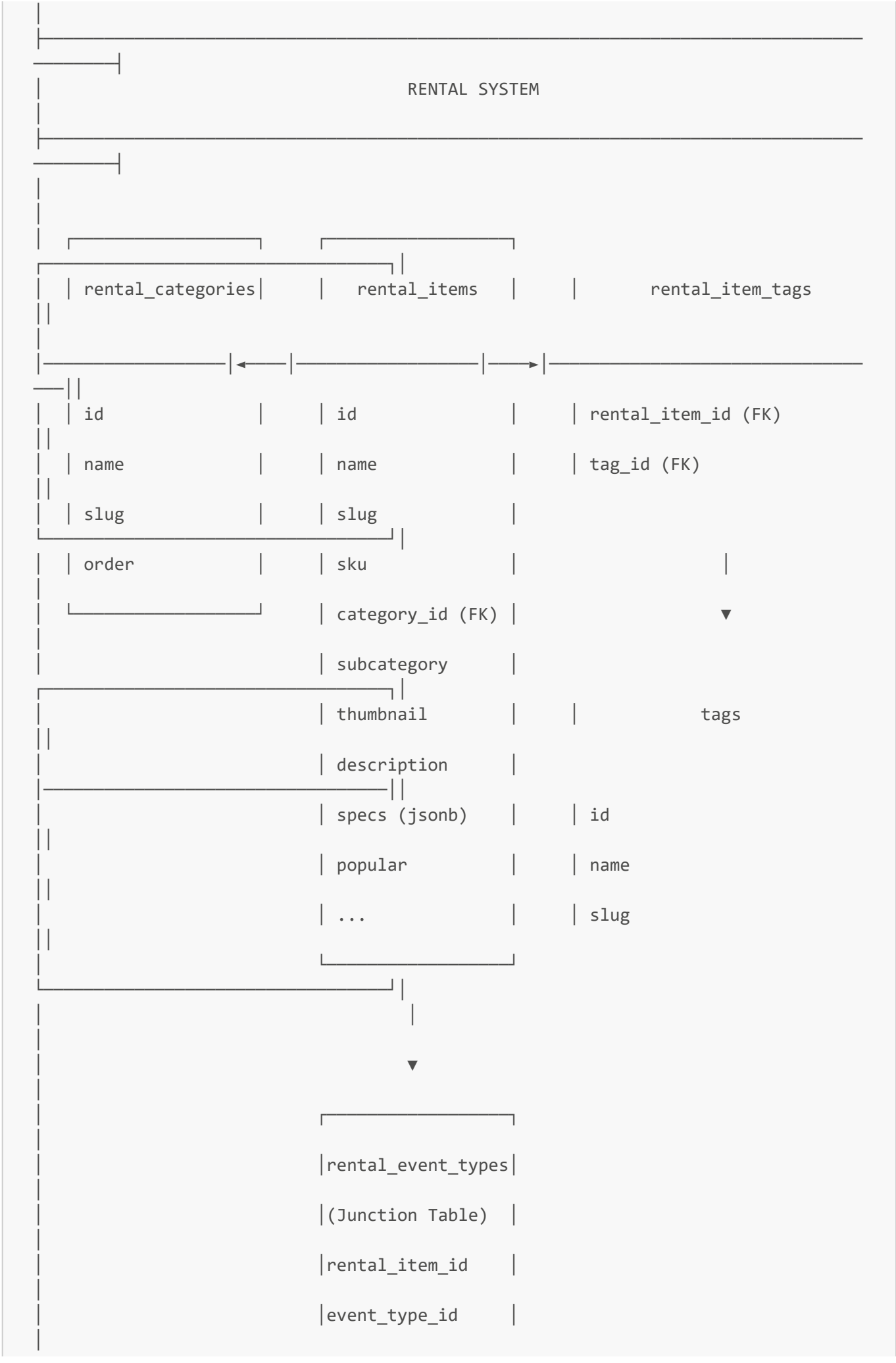
```
│   │   ├── use-venues.ts
│   │   ├── use-testimonials.ts
│   │   ├── use-faqs.ts
│   │   ├── use-blog-posts.ts
│   │   ├── use-blog-post.ts
│   │   └── use-quote-basket.ts
│   │
│   └── cms/                          # CMS data hooks
│       ├── use-services.ts
│       ├── use-service-mutations.ts
│       ├── use-rentals.ts
│       ├── use-rental-mutations.ts
│       ├── use-portfolio.ts
│       ├── use-portfolio-mutations.ts
│       ├── use-venues.ts
│       ├── use-venue-mutations.ts
│       ├── use-testimonials.ts
│       ├── use-testimonial-mutations.ts
│       ├── use-faqs.ts
│       ├── use-faq-mutations.ts
│       ├── use-blog-posts.ts
│       ├── use-blog-mutations.ts
│       ├── use-team.ts
│       ├── use-team-mutations.ts
│       ├── use-quotes.ts
│       ├── use-quote-mutations.ts
│       ├── use-inquiries.ts
│       ├── use-media.ts
│       ├── use-media-mutations.ts
│       ├── use-analytics.ts
│       └── use-settings.ts
│
├── lib/
│   ├── supabase/
│   │   ├── client.ts                 # Browser Supabase client
│   │   ├── server.ts                 # Server Supabase client
│   │   ├── admin.ts                  # Admin client (for scripts)
│   │   └── types.ts                  # Generated database types
│   │
│   ├── schemas/                      # Zod validation schemas
│   │   ├── service.schema.ts
│   │   ├── rental.schema.ts
│   │   ├── portfolio.schema.ts
│   │   ├── venue.schema.ts
│   │   ├── testimonial.schema.ts
│   │   ├── faq.schema.ts
│   │   ├── blog.schema.ts
│   │   ├── team.schema.ts
│   │   ├── quote.schema.ts
│   │   ├── inquiry.schema.ts
│   │   ├── media.schema.ts
│   │   └── settings.schema.ts
│   │
│   ├── types/                        # TypeScript type definitions
```

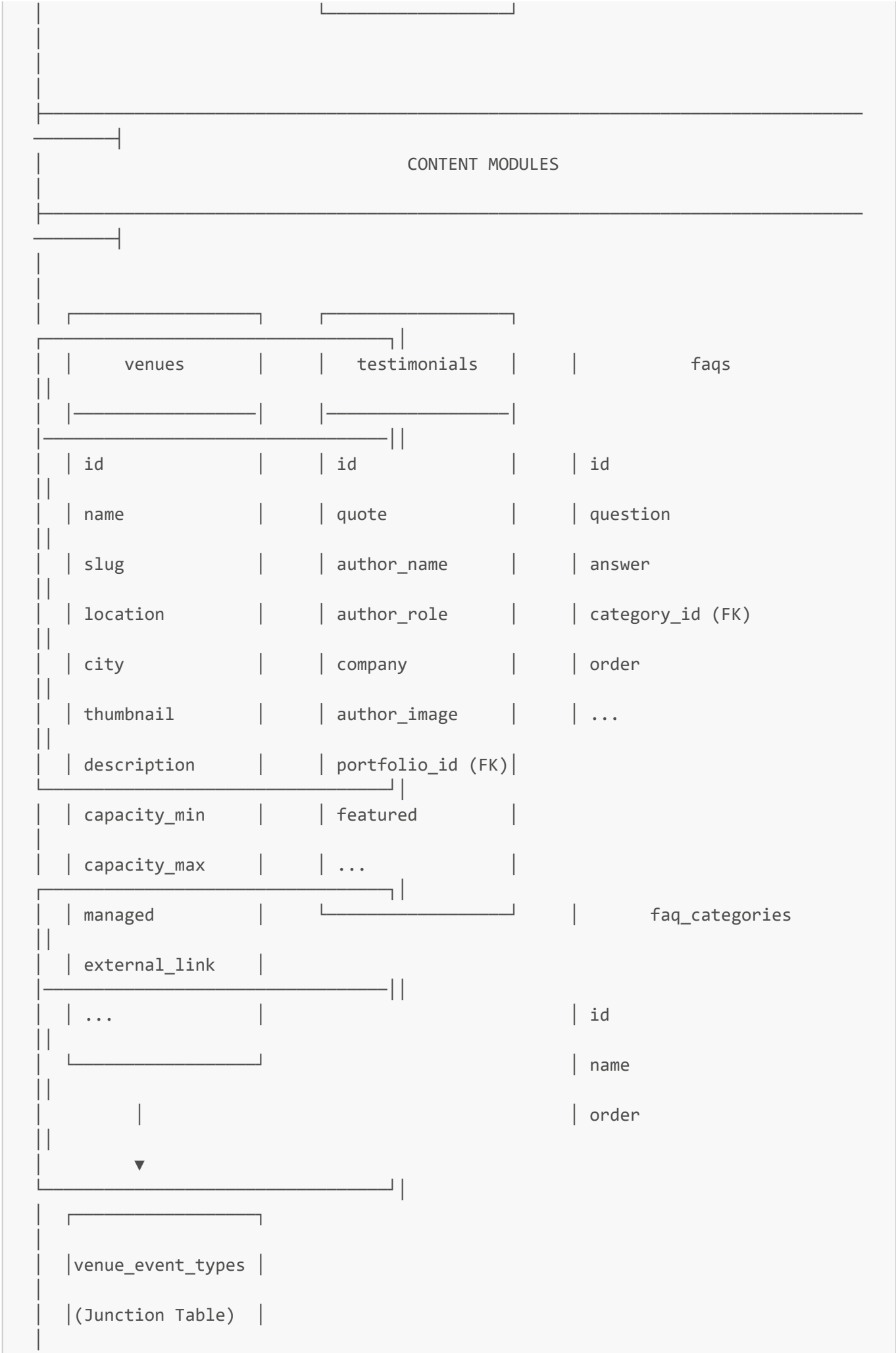
```
│   │   ├── database.types.ts        # Auto-generated from Supabase
│   │   ├── service.types.ts
│   │   ├── rental.types.ts
│   │   ├── portfolio.types.ts
│   │   ├── venue.types.ts
│   │   ├── testimonial.types.ts
│   │   ├── faq.types.ts
│   │   ├── blog.types.ts
│   │   ├── team.types.ts
│   │   ├── quote.types.ts
│   │   ├── inquiry.types.ts
│   │   └── api.types.ts             # API response types
│   │
│   ├── constants/
│   │   ├── navigation.ts            # CMS navigation config
│   │   ├── rental-categories.ts
│   │   ├── event-types.ts
│   │   └── query-keys.ts            # TanStack Query keys
│   │
│   ├── utils/
│   │   ├── cn.ts                    # Class name utility
│   │   ├── format.ts                # Formatting utilities
│   │   ├── slug.ts                  # Slug generation
│   │   ├── seo.ts                   # SEO utilities
│   │   └── storage.ts               # Supabase storage utilities
│   │
│   └── context/
│       ├── quote-basket-context.tsx
│       └── auth-context.tsx
│
├── scripts/
│   ├── migrate-images.ts            # Image migration to Supabase
│   ├── seed-data.ts                 # Initial data seeding
│   └── generate-types.ts            # Generate Supabase types
│
├── docs/
│   └── v2/
│       ├── ARCHITECTURE_V2.md       # This document
│       ├── WORKFLOW_V2.md           # Implementation workflow
│       ├── SCHEMA_V1.sql            # Database schema
│       └── CMS_UI_TEMPLATE.md       # UI/UX guidelines
│
└── public/
    └── ... (static assets)
```

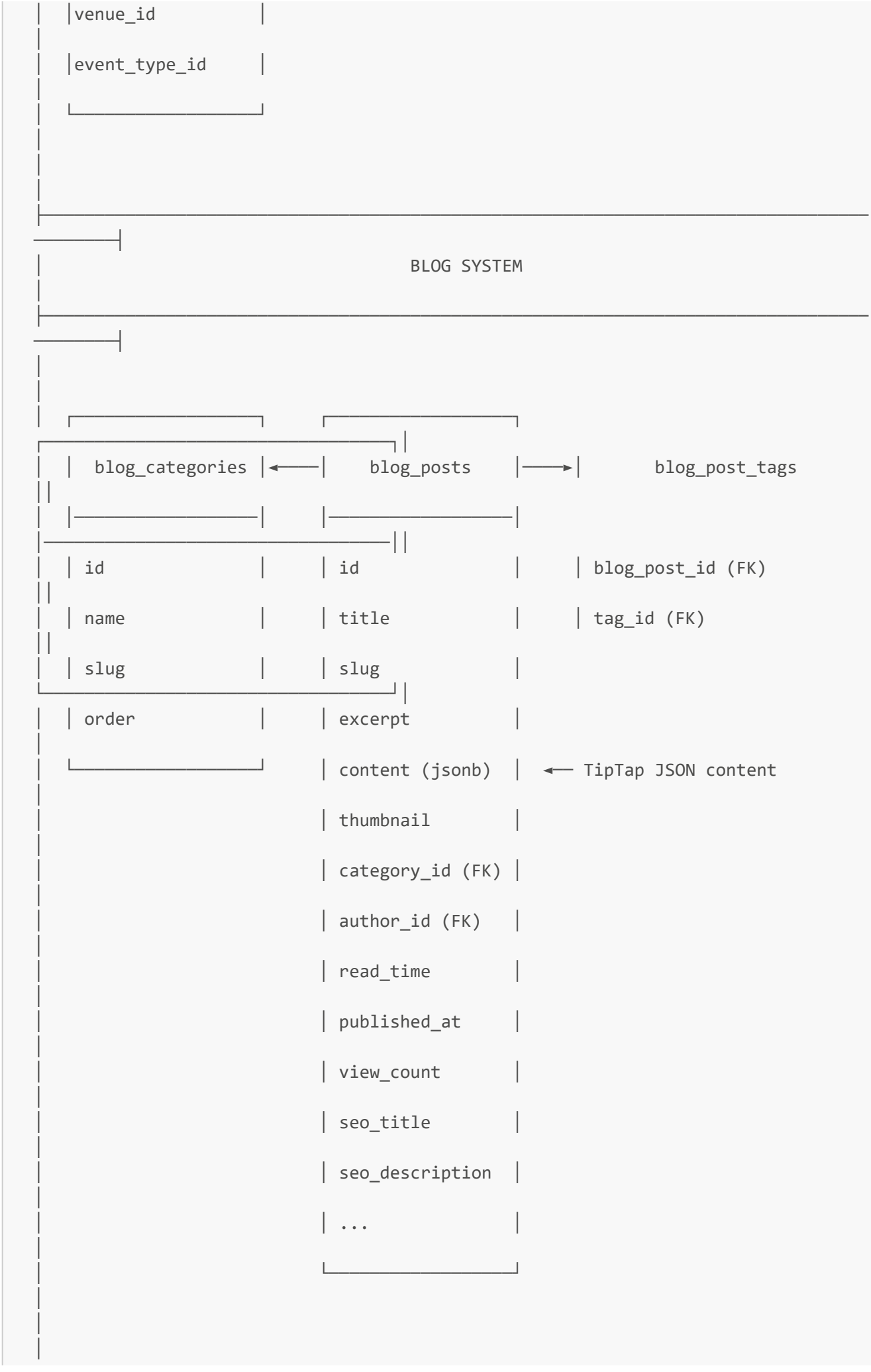
Data Architecture

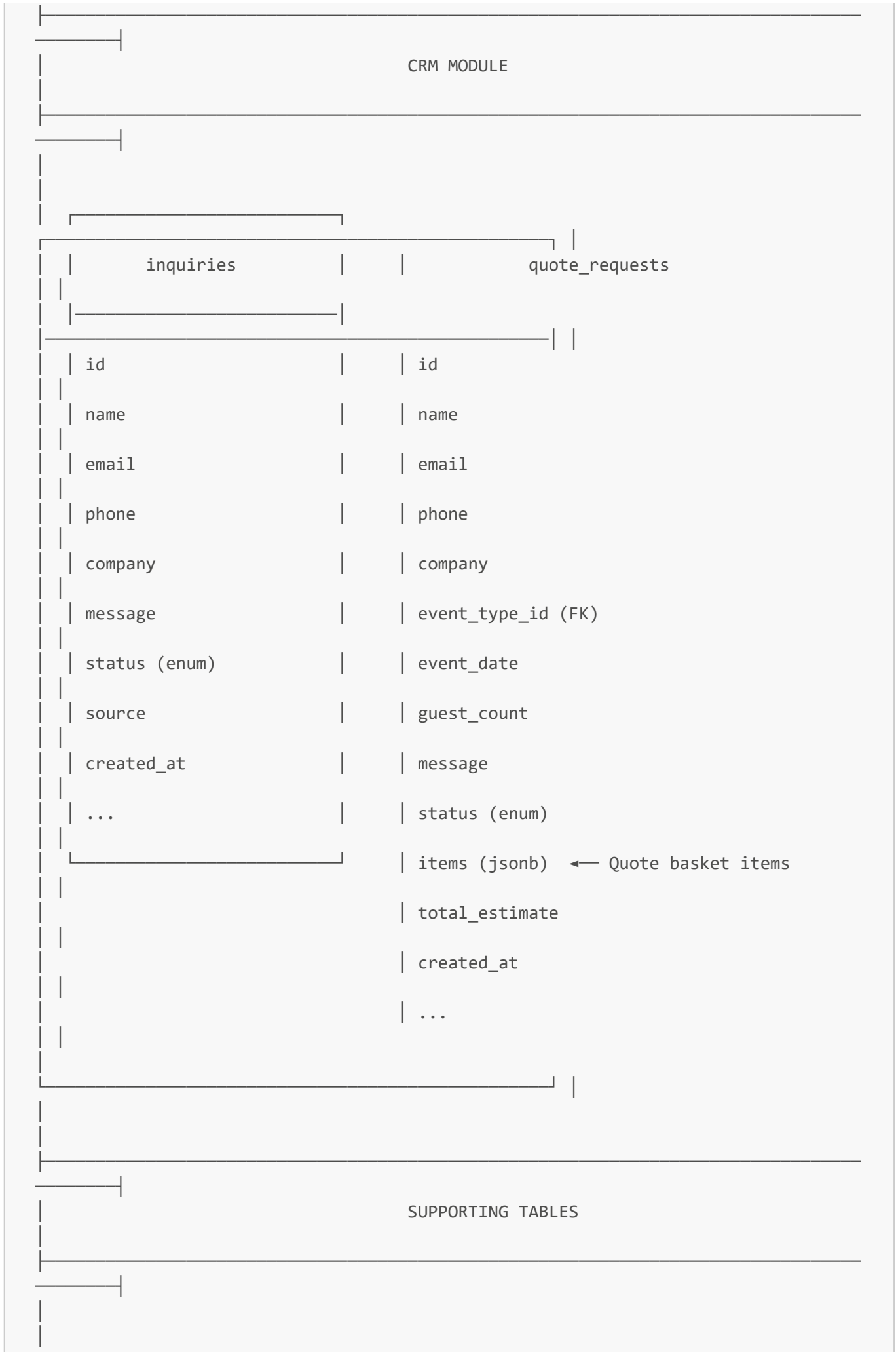
Entity Relationship Diagram

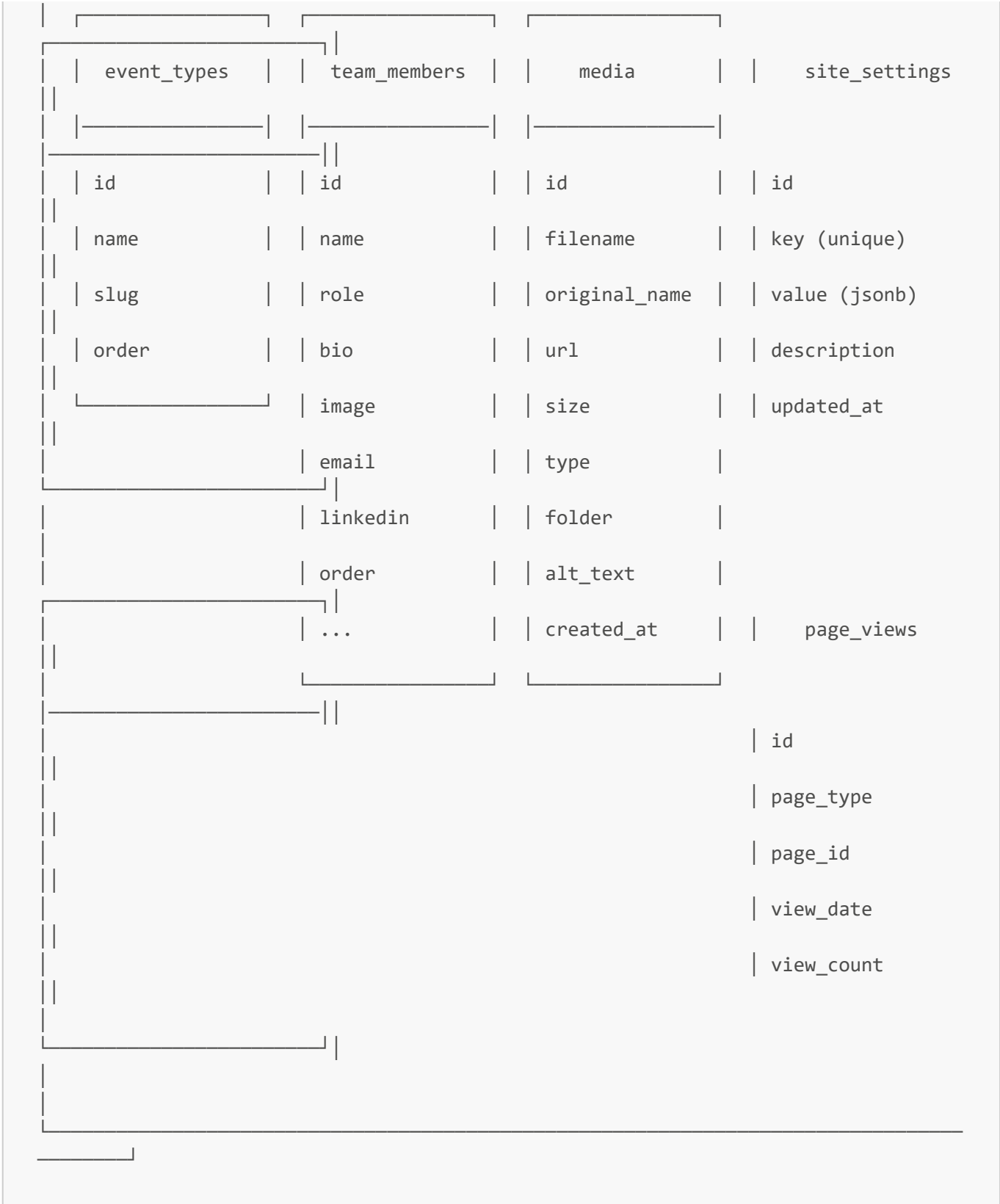

CORE ENTITIES
services
service_use_cases
portfolio_projects
id
name
slug
tagline
description
icon
thumbnail
...
id
service_id (FK)
title
description
image_url
order
id
title
event_type_id (FK)
venue
date
thumbnail
description
...
service_process
id
service_id (FK)
step
title
description
portfolio_services
(Junction Table)
portfolio_id (FK)
service_id (FK)


RENTAL SYSTEM
rental_categories
rental_items
rental_item_tags
id
name
slug
order
id
name
slug
sku
category_id (FK)
subcategory
thumbnail
description
specs (jsonb)
popular
...
rental_item_id (FK)
tag_id (FK)
tags
id
name
slug
rental_event_types
(Junction Table)
rental_item_id
event_type_id


CONTENT MODULES
venues
id
name
slug
location
city
thumbnail
description
capacity_min
capacity_max
managed
external_link
...
testimonials
id
quote
author_name
author_role
company
author_image
portfolio_id (FK)
featured
...
faqs
id
question
answer
category_id (FK)
order
...
faq_categories
id
name
order
venue_event_types
(Junction Table)


venue_id
event_type_id
BLOG SYSTEM
blog_categories
blog_posts
blog_post_tags
id
name
slug
order
id
title
slug
excerpt
content (jsonb)
thumbnail
category_id (FK)
author_id (FK)
read_time
published_at
view_count
seo_title
seo_description
...
blog_post_id (FK)
tag_id (FK)
TipTap JSON content


CRM MODULE
inquiries
id
name
email
phone
company
message
status (enum)
source
created_at
...
quote_requests
id
name
email
phone
company
event_type_id (FK)
event_date
guest_count
message
status (enum)
items (jsonb) ◄── Quote basket items
total_estimate
created_at
...
SUPPORTING TABLES



Data Flow Patterns

Public Site Data Flow

```
User Request → Next.js Route → useQuery Hook → API Route → Supabase → Response →
Cache → UI
                                    │
                                    └── Cache Hit? → Return cached data
```

CMS Data Flow

```
Admin Action → Form (React Hook Form + Zod) → useMutation → API Route → Auth Check
→ Supabase → Invalidate Cache → Update UI
                                                                          │
                                                                          └──

RLS Policies → Verify Admin Role
```

API Design

API Conventions

URL Structure

- **Public API:** `/api/public/{resource}` - Read-only, no auth required
- **CMS API:** `/api/cms/{resource}` - Full CRUD, auth required

HTTP Methods

Method	Purpose	Example
GET	Retrieve data	`GET /api/public/services`
POST	Create new record	`POST /api/cms/services`
PUT	Full update	`PUT /api/cms/services/[id]`
PATCH	Partial update	`PATCH /api/cms/services/[id]`
DELETE	Soft delete	`DELETE /api/cms/services/[id]`

Response Format

```
// Success Response
{
  success: true,
  data: T | T[],
  meta?: {
    total: number,
    page: number,
    limit: number
  }
}

// Error Response
{
  success: false,
```

```
    error: {
      code: string,
      message: string,
      details?: Record<string, string[]>
    }
  }
```

Public API Endpoints

Endpoint	Method	Description
`/api/public/services`	GET	List all services
`/api/public/services/[slug]`	GET	Get service by slug
`/api/public/rentals`	GET	List rentals (with filters)
`/api/public/rentals/[slug]`	GET	Get rental by slug
`/api/public/rentals/categories`	GET	List rental categories
`/api/public/portfolio`	GET	List portfolio projects
`/api/public/portfolio/[slug]`	GET	Get project by slug
`/api/public/venues`	GET	List venues
`/api/public/testimonials`	GET	List testimonials
`/api/public/faqs`	GET	List FAQs
`/api/public/blog`	GET	List blog posts
`/api/public/blog/[slug]`	GET	Get post by slug
`/api/public/quote-request`	POST	Submit quote request
`/api/public/contact`	POST	Submit contact form
`/api/public/analytics/track`	POST	Track page view

CMS API Endpoints

All CMS endpoints follow the same CRUD pattern:

Endpoint	Method	Description
`/api/cms/{resource}`	GET	List with pagination, search, filters
`/api/cms/{resource}`	POST	Create new record
`/api/cms/{resource}/[id]`	GET	Get by ID
`/api/cms/{resource}/[id]`	PUT	Full update
`/api/cms/{resource}/[id]`	DELETE	Soft delete

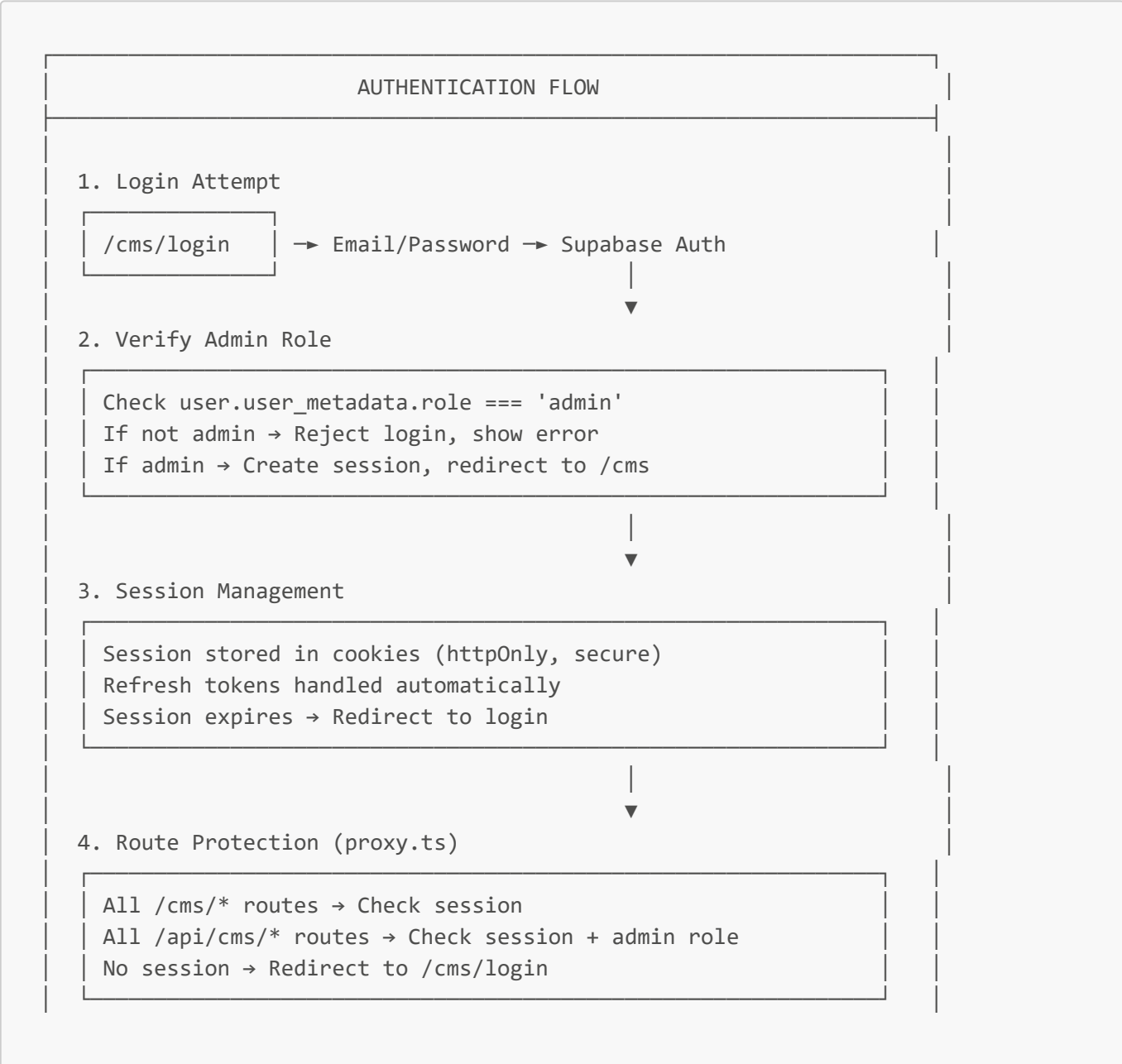
Endpoint	Method	Description
`/api/cms/{resource}/[id]/restore`	POST	Restore soft-deleted

Additional CMS endpoints:

Endpoint	Method	Description
`/api/cms/media`	POST	Upload file
`/api/cms/media/[id]`	DELETE	Delete file
`/api/cms/analytics`	GET	Dashboard statistics
`/api/cms/settings`	GET/PUT	Site settings

Authentication & Authorization

Auth Flow



```
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

User Metadata Structure

```
interface UserMetadata {
  role: 'admin';  // Single role for V2.0
  full_name: string;
  avatar_url?: string;
}
```

proxy.ts Implementation Pattern

```
// app/proxy.ts
import { createServerClient } from '@supabase/ssr';
import { NextResponse, type NextRequest } from 'next/server';

export async function middleware(request: NextRequest) {
  const { pathname } = request.nextUrl;

  // Only protect CMS routes
  if (!pathname.startsWith('/cms') && !pathname.startsWith('/api/cms')) {
    return NextResponse.next();
  }

  // Allow login page
  if (pathname === '/cms/login') {
    return NextResponse.next();
  }

  // Check session
  const supabase = createServerClient(/* config */);
  const { data: { session } } = await supabase.auth.getSession();

  if (!session) {
    return NextResponse.redirect(new URL('/cms/login', request.url));
  }

  // Verify admin role
  const role = session.user.user_metadata?.role;
  if (role !== 'admin') {
    return NextResponse.redirect(new URL('/cms/login?error=unauthorized',
request.url));
  }

  return NextResponse.next();
}
```

Component Architecture

Component Hierarchy

Component Design Principles

1. **Single Responsibility:** Each component does one thing well
2. **Composition Over Inheritance:** Build complex UIs from simple components

3. **Props Interface First:** Define TypeScript interfaces before implementation
4. **Controlled Components:** Forms use React Hook Form for state
5. **Loading States:** All data components have skeleton loading states
6. **Error Boundaries:** Graceful error handling at component level
7. **Accessibility:** ARIA labels, keyboard navigation, screen reader support

---

State Management

TanStack Query Configuration

```
// lib/query-client.ts
import { QueryClient } from '@tanstack/react-query';

export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 60 * 1000,        // 1 minute
      gcTime: 10 * 60 * 1000,      // 10 minutes (formerly cacheTime)
      retry: 1,
      refetchOnWindowFocus: false,
    },
    mutations: {
      onError: (error) => {
        // Global error handler
        console.error('Mutation error:', error);
      },
    },
  },
});
```

Query Keys Convention

```
// lib/constants/query-keys.ts
export const queryKeys = {
  // Public
  services: {
    all: ['services'] as const,
    list: () => [...queryKeys.services.all, 'list'] as const,
    detail: (slug: string) => [...queryKeys.services.all, 'detail', slug] as
const,
  },
  rentals: {
    all: ['rentals'] as const,
    list: (filters?: RentalFilters) => [...queryKeys.rentals.all, 'list', filters]
as const,
    detail: (slug: string) => [...queryKeys.rentals.all, 'detail', slug] as const,
    categories: () => [...queryKeys.rentals.all, 'categories'] as const,
  },
```

```
  // ... other resources

  // CMS
  cms: {
    services: {
      all: ['cms', 'services'] as const,
      list: (params?: ListParams) => [...queryKeys.cms.services.all, 'list',
params] as const,
      detail: (id: string) => [...queryKeys.cms.services.all, 'detail', id] as
const,
    },
    // ... other CMS resources
  },
};
```

Hook Pattern Example

```
// hooks/public/use-services.ts
import { useQuery } from '@tanstack/react-query';
import { queryKeys } from '@/lib/constants/query-keys';
import type { Service } from '@/lib/types/service.types';

export function useServices() {
  return useQuery({
    queryKey: queryKeys.services.list(),
    queryFn: async (): Promise<Service[]> => {
      const res = await fetch('/api/public/services');
      if (!res.ok) throw new Error('Failed to fetch services');
      const { data } = await res.json();
      return data;
    },
  });
}

// hooks/cms/use-service-mutations.ts
import { useMutation, useQueryClient } from '@tanstack/react-query';
import { queryKeys } from '@/lib/constants/query-keys';
import { toast } from 'sonner';

export function useCreateService() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (data: CreateServiceInput) => {
      const res = await fetch('/api/cms/services', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data),
      });
      if (!res.ok) throw new Error('Failed to create service');
      return res.json();
```

```
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: queryKeys.cms.services.all });
      queryClient.invalidateQueries({ queryKey: queryKeys.services.all });
      toast.success('Service created successfully');
    },
    onError: (error) => {
      toast.error('Failed to create service');
    },
  });
}
```

Caching Strategy

Cache Invalidation Rules

Action	Invalidate
Create service	`cms.services.all`, `services.all`
Update service	`cms.services.all`, `cms.services.detail(id)`, `services.all`, `services.detail(slug)`
Delete service	`cms.services.all`, `services.all`
Upload media	`cms.media.all`

Optimistic Updates

For better UX, implement optimistic updates on mutations:

```
useMutation({
  mutationFn: updateService,
  onMutate: async (newData) => {
    await queryClient.cancelQueries({ queryKey: queryKeys.cms.services.detail(id)
});
    const previous = queryClient.getQueryData(queryKeys.cms.services.detail(id));
    queryClient.setQueryData(queryKeys.cms.services.detail(id), newData);
    return { previous };
  },
  onError: (err, newData, context) => {
    queryClient.setQueryData(queryKeys.cms.services.detail(id),
context?.previous);
  },
  onSettled: () => {
    queryClient.invalidateQueries({ queryKey: queryKeys.cms.services.detail(id)
});
  },
});
```

Error Handling

API Error Handling

```
// lib/utils/api-error.ts
export class ApiError extends Error {
  constructor(
    public statusCode: number,
    public code: string,
    message: string,
    public details?: Record<string, string[]>
  ) {
    super(message);
    this.name = 'ApiError';
  }

  static badRequest(message: string, details?: Record<string, string[]>) {
    return new ApiError(400, 'BAD_REQUEST', message, details);
  }

  static unauthorized(message = 'Unauthorized') {
    return new ApiError(401, 'UNAUTHORIZED', message);
  }

  static forbidden(message = 'Forbidden') {
    return new ApiError(403, 'FORBIDDEN', message);
  }

  static notFound(resource = 'Resource') {
    return new ApiError(404, 'NOT_FOUND', `${resource} not found`);
  }

  static internal(message = 'Internal server error') {
    return new ApiError(500, 'INTERNAL_ERROR', message);
  }
}
```

Client-Side Error Handling

```
// Global error boundary for React Query
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      throwOnError: false,
    },
    mutations: {
      throwOnError: false,
      onError: (error) => {
        if (error instanceof ApiError) {
```

```
            toast.error(error.message);
          } else {
            toast.error('An unexpected error occurred');
          }
        },
      },
    },
  });
```

File Storage

Supabase Storage Structure

```
storage/
├── public/                    # Publicly accessible
│   ├── services/              # Service images
│   │   ├── thumbnails/
│   │   └── gallery/
│   ├── rentals/
│   │   ├── thumbnails/
│   │   └── gallery/
│   ├── portfolio/
│   │   ├── thumbnails/
│   │   └── gallery/
│   ├── blog/
│   │   ├── thumbnails/
│   │   └── content/
│   ├── team/
│   ├── venues/
│   └── testimonials/
│
└── private/                   # Authenticated access only
    └── documents/             # Any internal documents
```

Storage Policies

```sql
-- Public bucket: anyone can read
CREATE POLICY "Public read access"
ON storage.objects FOR SELECT
USING (bucket_id = 'public');

-- Only authenticated admins can upload/delete
CREATE POLICY "Admin write access"
ON storage.objects FOR INSERT
WITH CHECK (
  bucket_id = 'public' AND
  auth.role() = 'authenticated' AND
  (auth.jwt() -> 'user_metadata' ->> 'role') = 'admin'
```

```
);

CREATE POLICY "Admin delete access"
ON storage.objects FOR DELETE
USING (
  bucket_id = 'public' AND
  auth.role() = 'authenticated' AND
  (auth.jwt() -> 'user_metadata' ->> 'role') = 'admin'
);
```

Image Upload Utility

```
// lib/utils/storage.ts
import { createClient } from '@/lib/supabase/client';

interface UploadResult {
  url: string;
  path: string;
}

export async function uploadImage(
  file: File,
  folder: string,
  filename?: string
): Promise<UploadResult> {
  const supabase = createClient();
  const ext = file.name.split('.').pop();
  const name = filename || `${Date.now()}-${Math.random().toString(36).slice(2)}`;
  const path = `${folder}/${name}.${ext}`;

  const { error } = await supabase.storage
    .from('public')
    .upload(path, file, {
      cacheControl: '3600',
      upsert: false,
    });

  if (error) throw error;

  const { data: { publicUrl } } = supabase.storage
    .from('public')
    .getPublicUrl(path);

  return { url: publicUrl, path };
}

export async function deleteImage(path: string): Promise<void> {
  const supabase = createClient();
  const { error } = await supabase.storage.from('public').remove([path]);
  if (error) throw error;
}
```

Analytics Tracking

Page View Tracking

```
// Track view on page load
// hooks/public/use-track-view.ts
import { useEffect } from 'react';

export function useTrackView(pageType: string, pageId: string) {
  useEffect(() => {
    const track = async () => {
      await fetch('/api/public/analytics/track', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ pageType, pageId }),
      });
    };
    track();
  }, [pageType, pageId]);
}
```

Dashboard Analytics

The CMS dashboard will display:

- Total views by resource type
- Most viewed services/rentals/blog posts
- Quote request trends
- Inquiry volume over time
- Recent activity feed

Security Considerations

Row Level Security (RLS)

All tables will have RLS enabled with these policies:

1. **Public Read:** Anyone can read published content
2. **Admin Full Access:** Authenticated admins have full CRUD access
3. **Soft Delete Protection:** Deleted records hidden from public API

Input Validation

- All inputs validated with Zod schemas on both client and server
- SQL injection protected through parameterized queries (Supabase default)
- XSS prevention through React's default escaping

Rate Limiting

Consider implementing rate limiting on:

- Quote request submissions
- Contact form submissions
- Analytics tracking endpoint

Environment Variables

```
# .env.local
NEXT_PUBLIC_SUPABASE_URL=https://xxx.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=xxx

# Server-only
SUPABASE_SERVICE_ROLE_KEY=xxx
```

Appendix

Glossary

Term	Definition
CMS	Content Management System
RLS	Row Level Security
CRM	Customer Relationship Management
CRUD	Create, Read, Update, Delete

Related Documents

- WORKFLOW_V2.md - Implementation workflow
- SCHEMA_V1.sql - Database schema
- CMS_UI_TEMPLATE.md - UI/UX guidelines

This document is a living reference and will be updated as the architecture evolves.