CCD SaaS Suite

Technical Documentation

Version 2.0

Crane & Ceeshar Digital

January 31, 2026

Document Type:	Technical Specification
Status:	Final Draft
Audience:	Development Team

Table of Contents

1. Architecture Overview

2. Technology Stack Recommendation

3. Backend Architecture

4. Frontend Architecture

5. Database Design

6. API Design

7. Authentication & Authorisation

8. AI/ML Integration

9. Infrastructure & DevOps

10. Security Considerations

11. Cost Analysis

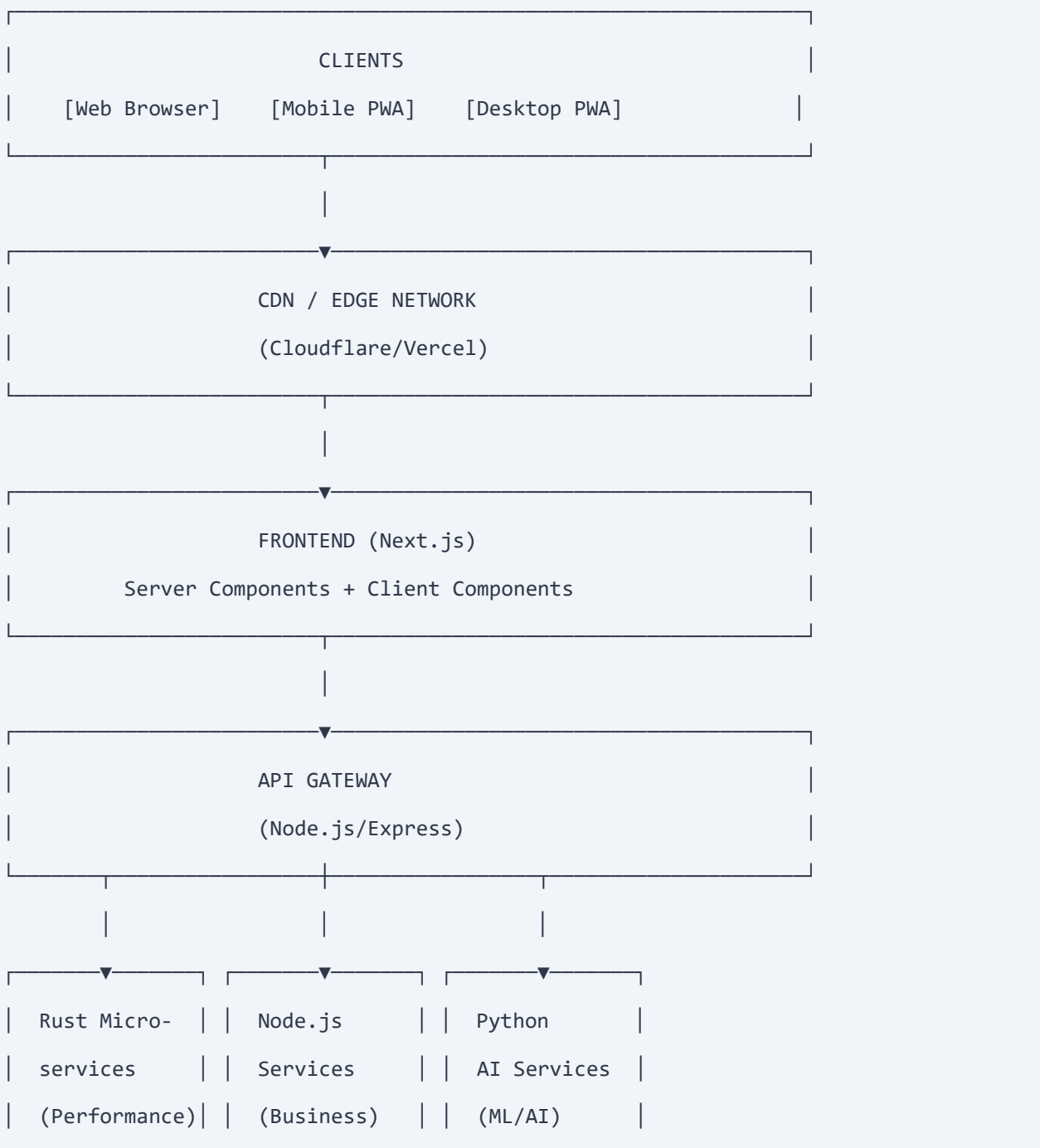
12. Development Roadmap

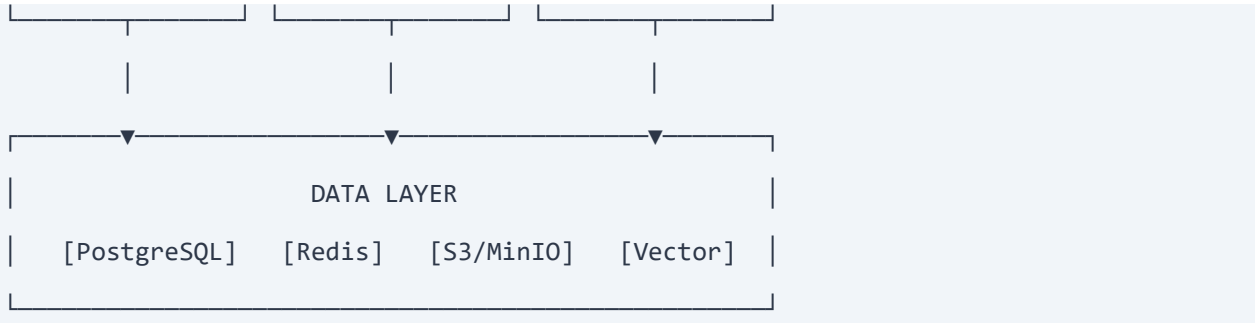
1. Architecture Overview

1.1 High-Level Architecture

CCD SaaS Suite follows a modern microservices architecture with clear separation between frontend, backend services, and data layer. The architecture prioritises scalability, maintainability, and cost-effectiveness.

Architecture Diagram





1.2 Architecture Principles

- Hybrid Microservices: Rust for performance-critical, Node.js for rapid development
- API-First Design: All services communicate via well-defined APIs
- Event-Driven: Async communication for non-blocking operations
- Multi-Tenant: Single codebase serving multiple organisations
- Cloud-Native: Containerised services with Kubernetes orchestration
- Security by Design: Authentication, authorisation, and encryption at every layer

2. Technology Stack Recommendation

2.1 Recommended Stack Overview

Layer	Technology	Rationale
Frontend	Next.js 14+ (React)	SSR, App Router, excellent DX
API Gateway	Node.js + Express/Fastify	Rapid development, ecosystem
High-Performance Services	Rust + Actix-web	Safety, speed, concurrency
AI/ML Services	Python + FastAPI	ML libraries, AI ecosystem
Primary Database	PostgreSQL	ACID, JSON, full-text search
Cache/Sessions	Redis	Speed, pub/sub, sessions
File Storage	MinIO/S3	S3-compatible, self-hosted option
Vector DB (AI)	pgvector	PostgreSQL extension, simple
Message Queue	Redis Streams / BullMQ	Job queues, events
Search	PostgreSQL Full-Text	Built-in, cost-effective

2.2 Why This Hybrid Approach?

Rust for Performance-Critical Services

Rust handles computationally intensive and high-throughput operations:

- Analytics Processing: Real-time data aggregation and calculations
- File Processing: Large file uploads, image processing, PDF generation
- WebSocket Handlers: Real-time collaboration, notifications
- Background Jobs: Heavy batch processing, data migrations
- API Rate Limiting: High-performance request throttling

Node.js for Business Logic

Node.js handles typical CRUD operations and business workflows:

- API Gateway: Request routing, authentication middleware
- Business Logic: CRM, Projects, HR, Finance module logic
- Third-party Integrations: OAuth, payment gateways, external APIs
- Real-time Features: Socket.IO for simpler real-time needs

Python for AI/ML

Python powers all AI and machine learning features:

- LLM Integration: OpenAI, Anthropic, or self-hosted models
- Prediction Models: Lead scoring, forecasting, recommendations
- NLP Tasks: Content analysis, sentiment analysis
- Data Science: Complex analytics and insights generation

2.3 Cost-Effective Tool Selection

Category	Recommended	Alternative	Est. Monthly Cost
Database	Supabase	Neon, Railway	$25-100
Hosting	Railway	Fly.io, DigitalOcean	$50-200
Frontend	Vercel	Netlify, Cloudflare Pages	$0-20
Redis	Upstash	Railway Redis	$0-50
File Storage	Cloudflare R2	Backblaze B2, MinIO	$5-50
Email	Resend	SendGrid, Postmark	$0-50
Monitoring	Sentry + Grafana Cloud	Better Stack	$0-50
AI API	OpenAI / Anthropic	Self-hosted Ollama	$50-500

Estimated Total Infrastructure Cost: $200-1,000/month for MVP

3. Backend Architecture

3.1 Service Breakdown

API Gateway (Node.js)

Central entry point for all client requests.

```
// Technology: Node.js + Fastify

// Responsibilities:

// - Request routing to appropriate microservices

// - Authentication/Authorisation middleware

// - Rate limiting and request validation

// - API versioning

// - Request/Response logging
```

Core Services (Node.js)

Service	Responsibility	Key Endpoints
auth-service	Authentication, sessions, SSO	/auth/*
user-service	User management, profiles	/users/*
tenant-service	Organization management	/tenants/*
crm-service	Deals, contacts, companies	/crm/*
project-service	Projects, tasks, time tracking	/projects/*
content-service	Content, calendar, assets	/content/*
finance-service	Invoicing, payments, expenses	/finance/*
hr-service	Employees, payroll, compliance	/hr/*
notification-service	Email, push, in-app	/notifications/*

Performance Services (Rust)

Service	Responsibility	Why Rust?
analytics-engine	Data aggregation, metrics	CPU-intensive calculations
file-processor	Upload, resize, convert	Memory-efficient streaming
realtime-gateway	WebSocket connections	10K+ concurrent connections
job-runner	Background job processing	High throughput, reliability

search-indexer	Full-text search indexing	Fast batch processing

AI Services (Python)

Service	Responsibility	ML/AI Tasks
llm-service	LLM API integration	Content generation, chat
prediction-service	ML predictions	Lead scoring, forecasting
nlp-service	Text analysis	Sentiment, classification
recommendation-service	Recommendations	Next actions, suggestions

3.2 Inter-Service Communication

Synchronous (REST/gRPC)

- REST for external-facing APIs and simple internal calls
- gRPC for high-performance internal communication

Asynchronous (Message Queue)

- Redis Streams for event-driven communication
- BullMQ for background job processing

Example events: user.created, deal.won, content.published, invoice.paid

3.3 Rust Service Example Structure

```
ccd-analytics-engine/
├── Cargo.toml
├── src/
│   ├── main.rs
│   ├── config.rs
│   ├── routes/
│   │   ├── mod.rs
│   │   ├── metrics.rs
│   │   └── reports.rs
│   ├── services/
│   │   ├── mod.rs
│   │   ├── aggregation.rs
│   │   └── calculation.rs
```

```
│   ├── models/
│   └── db/
└── tests/
```

3.4 Node.js Service Example Structure

```
ccd-crm-service/
├── package.json
├── src/
│   ├── index.ts
│   ├── config/
│   ├── routes/
│   │   ├── deals.routes.ts
│   │   ├── contacts.routes.ts
│   │   └── companies.routes.ts
│   ├── services/
│   ├── models/
│   ├── middleware/
│   └── utils/
└── tests/
```

4. Frontend Architecture

4.1 Next.js 14+ App Router Structure

```
ccd-frontend/
├── app/
│   ├── (auth)/
│   │   ├── login/
│   │   ├── register/
│   │   └── forgot-password/
│   ├── (dashboard)/
│   │   ├── layout.tsx          # Dashboard shell with sidebar
│   │   ├── page.tsx            # Dashboard home
│   │   ├── crm/
│   │   │   ├── page.tsx        # CRM dashboard
│   │   │   ├── pipeline/
│   │   │   ├── deals/
│   │   │   ├── contacts/
│   │   │   └── companies/
│   │   ├── analytics/
│   │   ├── content/
│   │   ├── seo/
│   │   ├── social/
│   │   ├── projects/
│   │   ├── finance/
│   │   ├── hr/
│   │   └── settings/
│   ├── portal/                 # Client portal (separate layout)
│   └── api/                    # API routes (BFF pattern)
├── components/
│   ├── ui/                     # Base UI components
```

```
│   ├── modules/              # Module-specific components
│   └── shared/               # Shared components
├── lib/
│   ├── api/                  # API client
│   ├── hooks/                # Custom hooks
│   └── utils/
└── styles/
```

4.2 Key Frontend Technologies

Purpose	Technology	Rationale
Framework	Next.js 14+	SSR, App Router, great DX
Styling	Tailwind CSS	Utility-first, fast development
Components	shadcn/ui	Accessible, customizable, free
State (Global)	Zustand	Simple, lightweight, TypeScript
State (Server)	TanStack Query	Caching, background updates
Forms	React Hook Form + Zod	Performance, validation
Charts	Recharts	React-native, customizable
Tables	TanStack Table	Headless, powerful
DnD	dnd-kit	Accessible, performant
Rich Text	Tiptap	Extensible, collaborative

4.3 Module-Based Code Splitting

Each module is lazy-loaded to optimise initial bundle size:

```
// app/(dashboard)/crm/page.tsx
import { Suspense } from 'react';
import { CRMDashboard } from '@/components/modules/crm';
import { DashboardSkeleton } from '@/components/ui/skeleton';

export default function CRMPage() {
  return (
```

```
    <Suspense fallback={<DashboardSkeleton />}>
      <CRMDashboard />
    </Suspense>
  );
}
```

4.4 User-Type Based Routing

Middleware handles user-type access control:

```
// middleware.ts
export function middleware(request: NextRequest) {
  const userType = getUserTypeFromToken(request);
  const path = request.nextUrl.pathname;

  const moduleAccess = {
    sales: ['/crm', '/analytics'],
    marketing: ['/content', '/seo', '/social', '/analytics'],
    hr: ['/hr', '/analytics'],
    // ... other user types
  };

  if (!hasAccess(userType, path, moduleAccess)) {
    return NextResponse.redirect('/unauthorized');
  }
}
```

5. Database Design

5.1 Database Recommendation: PostgreSQL

PostgreSQL is the optimal choice for CCD due to:

- ACID Compliance: Critical for financial and HR data
- JSON Support: Flexible schema for varying module needs
- Full-Text Search: Built-in search eliminates the need for Elasticsearch
- Row-Level Security: Native multi-tenant support
- Extensions: pgvector for AI, PostGIS for location, etc.
- Ecosystem: Excellent tools, hosting options, community support

5.2 Multi-Tenant Strategy

We recommend Row-Level Security (RLS) for multi-tenancy:

```
-- Enable RLS on all tables
ALTER TABLE deals ENABLE ROW LEVEL SECURITY;


-- Create policy for tenant isolation
CREATE POLICY tenant_isolation ON deals
  USING (tenant_id = current_setting('app.current_tenant')::uuid);


-- Set tenant context in application
SET app.current_tenant = 'tenant-uuid-here';
```

5.3 Core Schema Overview

Tenant & User Management

```
-- Organisations/Tenants
CREATE TABLE tenants (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR(255) NOT NULL,
  slug VARCHAR(100) UNIQUE NOT NULL,
  plan VARCHAR(50) DEFAULT 'starter',
  settings JSONB DEFAULT '{}',
```

```
  created_at TIMESTAMPTZ DEFAULT NOW()
);


-- Users
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tenant_id UUID REFERENCES tenants(id),
  email VARCHAR(255) NOT NULL,
  user_type VARCHAR(50) NOT NULL, -- 'admin', 'sales', 'marketing', etc.
  profile JSONB DEFAULT '{}',
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(tenant_id, email)
);
```

CRM Module Tables

```
-- Contacts
CREATE TABLE contacts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tenant_id UUID REFERENCES tenants(id),
  company_id UUID REFERENCES companies(id),
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(255),
  phone VARCHAR(50),
  metadata JSONB DEFAULT '{}',
  created_at TIMESTAMPTZ DEFAULT NOW()
);


-- Deals
CREATE TABLE deals (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
  tenant_id UUID REFERENCES tenants(id),
  contact_id UUID REFERENCES contacts(id),
  title VARCHAR(255) NOT NULL,
  value DECIMAL(12, 2),
  stage VARCHAR(50) DEFAULT 'lead',
  probability INTEGER DEFAULT 0,
  expected_close DATE,
  metadata JSONB DEFAULT '{}',
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

5.4 Database Hosting Options

Provider	Free Tier	Paid Starting	Best For
Supabase	500MB, 2 projects	$25/mo	Best overall, includes auth
Neon	512MB, branching	$19/mo	Serverless, scales to zero
Railway	500MB, $5 credit	$5/mo	Simple, good DX
PlanetScale	5GB (MySQL)	$29/mo	If MySQL preferred
Self-hosted	N/A	$20-50/mo	Full control, more work

Recommendation: Start with Supabase for MVP (includes auth, storage, and realtime).

6. API Design

6.1 API Standards

- RESTful Design: Resource-based URLs, standard HTTP methods
- Versioning: URL-based (/api/v1/...)
- Authentication: JWT with refresh tokens
- Rate Limiting: Per-tenant and per-user limits
- Documentation: OpenAPI/Swagger auto-generated

6.2 URL Structure

```
Base URL: https://api.ccd-app.com/v1


CRM Endpoints:
  GET    /crm/deals              - List deals
  POST   /crm/deals              - Create deal
  GET    /crm/deals/:id          - Get deal
  PATCH  /crm/deals/:id          - Update deal
  DELETE /crm/deals/:id          - Delete deal
  POST   /crm/deals/:id/stage    - Change stage


Content Endpoints:
  GET    /content/calendar       - Get calendar items
  POST   /content/posts          - Create content
  POST   /content/posts/:id/schedule - Schedule post


Analytics Endpoints:
  GET    /analytics/dashboard    - Dashboard metrics
  POST   /analytics/reports      - Generate report
```

6.3 Request/Response Format

```
// Successful Response
{
```

```
  "success": true,
  "data": { ... },
  "meta": {
    "page": 1,
    "per_page": 20,
    "total": 150
  }
}

// Error Response
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid email format",
    "details": [...]
  }
}
```

6.4 Rate Limiting

Plan	Requests/Hour	Burst Limit
Starter	1,000	100/minute
Professional	10,000	500/minute
Enterprise	100,000	2,000/minute
Custom	Unlimited	Custom

7. Authentication & Authorisation

7.1 Authentication Strategy

- Primary: Email/Password with secure hashing (Argon2)
- SSO: Google, Microsoft, SAML for enterprise
- MFA: TOTP-based two-factor authentication
- Sessions: JWT access tokens (15 min) + refresh tokens (7 days)

7.2 JWT Token Structure

```
{
  "sub": "user-uuid",
  "tid": "tenant-uuid",
  "type": "marketing",    // user type
  "modules": ["content", "seo", "social", "analytics"],
  "iat": 1234567890,
  "exp": 1234568790
}
```

7.3 User-Type Authorisation

```
// Middleware example
function requireModule(module: string) {
  return (req, res, next) => {
    const { modules } = req.user;
    if (!modules.includes(module)) {
      return res.status(403).json({
        error: 'ACCESS_DENIED',
        message: 'You do not have access to this module'
      });
    }
    next();
  };
}
```

```
// Usage
router.get('/crm/deals', requireModule('crm'), dealsController.list);
```

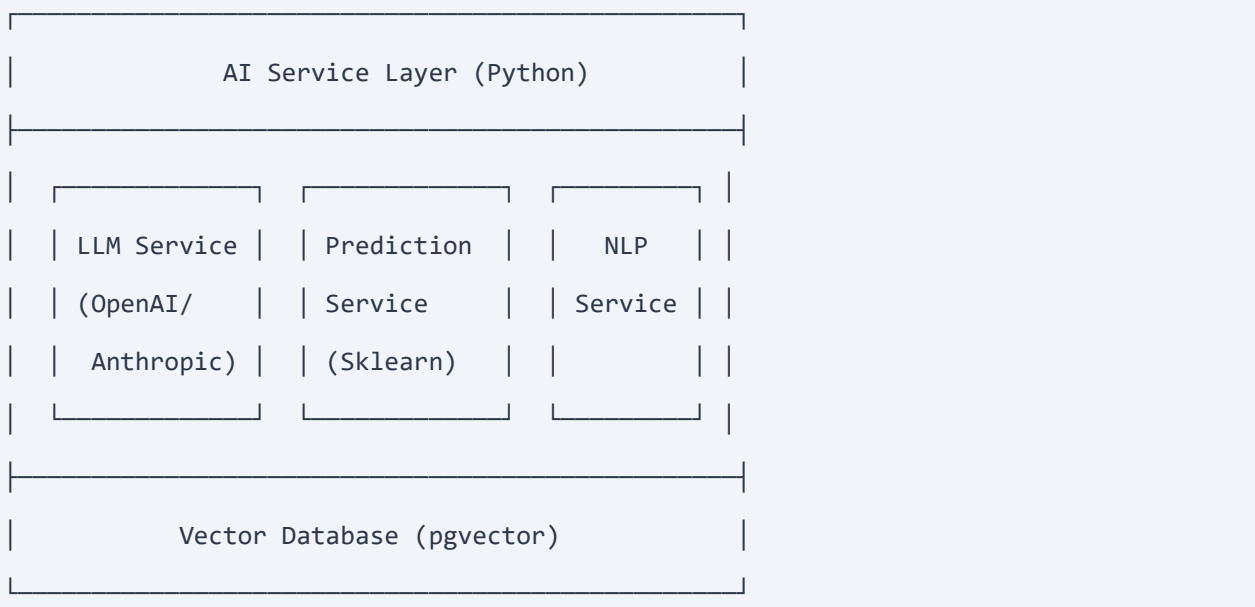
7.4 Client Portal Authentication

Separate authentication flow for external clients:

- Magic link authentication (no passwords)
- Scoped tokens (client portal access only)
- Tenant-specific branding on login page

8. AI/ML Integration

8.1 AI Architecture



8.2 LLM Integration (Content AI)

```
# Python FastAPI example
from openai import OpenAI
from anthropic import Anthropic


class LLMService:
    def __init__(self):
        self.openai = OpenAI()
        self.anthropic = Anthropic()


    async def generate_content(self, prompt: str, context: dict):
        response = await self.openai.chat.completions.create(
            model="gpt-4-turbo",
            messages=[
                {"role": "system", "content": CONTENT_SYSTEM_PROMPT},
```

```
                {"role": "user", "content": prompt}
            ],
            temperature=0.7
        )
        return response.choices[0].message.content
```

8.3 Prediction Models

Model	Purpose	Algorithm	Features
Lead Scoring	Predict conversion	XGBoost	Activity, engagement, profile
Deal Forecast	Revenue prediction	Time Series	Historical deals, seasonality
Churn Risk	Identify at-risk	Random Forest	Usage patterns, support tickets
Task Duration	Estimate time	Regression	Task type, complexity, history

8.4 Cost-Effective AI Strategy

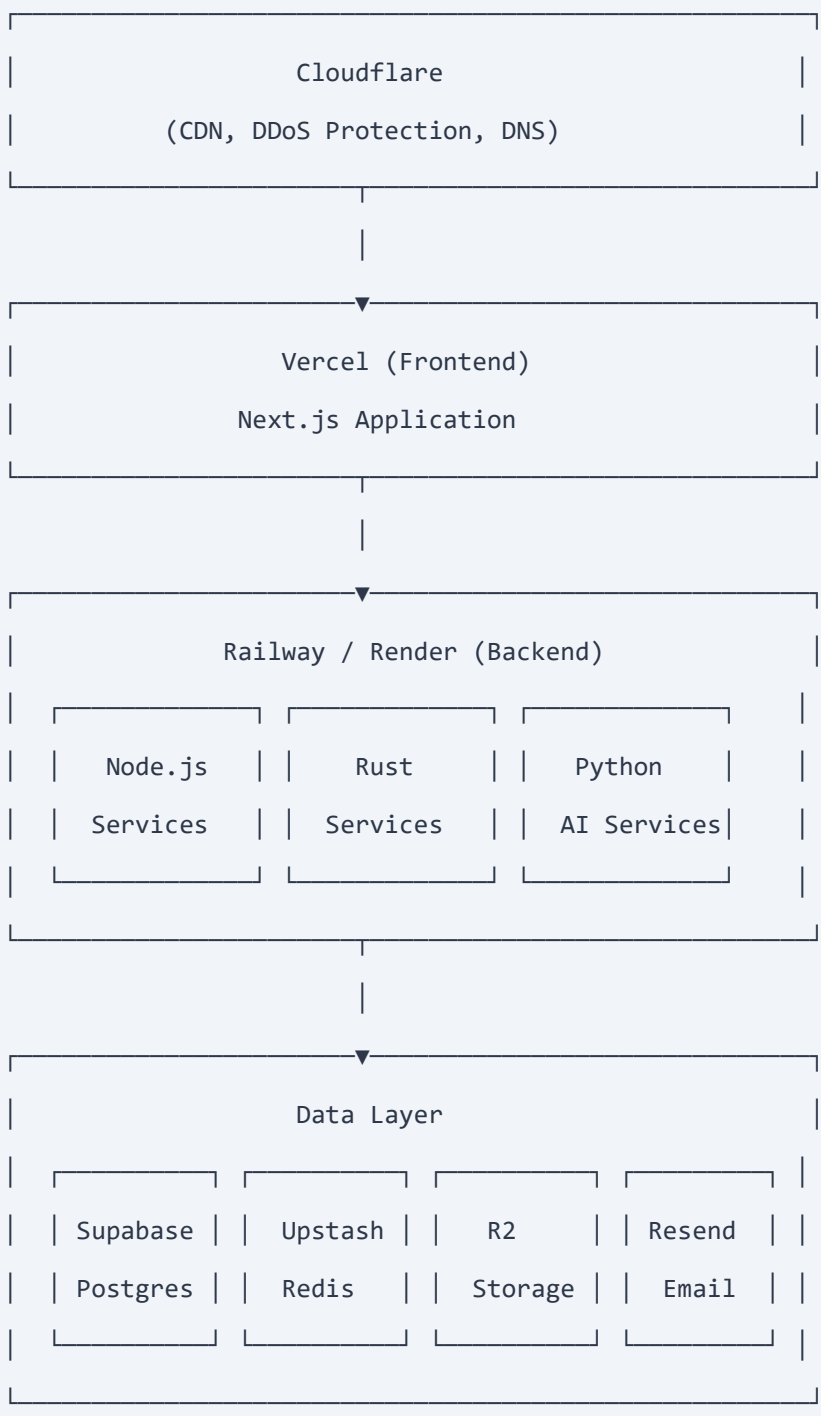
- Tiered Models: Use cheaper models for simple tasks, and more expensive models for complex tasks
- Caching: Cache common AI responses to reduce API calls
- Batching: Batch AI requests where possible
- Rate Limits: Enforce per-tenant AI usage limits
- Local Models: Consider Ollama for some tasks (self-hosted)

Estimated AI API Costs:

Usage Level	Monthly Requests	Estimated Cost
Low (Starter)	0	$0 (no AI)
Medium (Pro)	5,000	$50-100
High (Enterprise)	50,000	$300-500
Very High	500,000+	$2,000+

9. Infrastructure & DevOps

9.1 Deployment Architecture



9.2 CI/CD Pipeline

```
# .github/workflows/deploy.yml
```

```
name: Deploy

on:

  push:

    branches: [main]


jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v4

      - name: Run tests

        run: npm test


  deploy-frontend:

    needs: test

    runs-on: ubuntu-latest

    steps:

      - name: Deploy to Vercel

        uses: amondnet/vercel-action@v25


  deploy-backend:

    needs: test

    runs-on: ubuntu-latest

    steps:

      - name: Deploy to Railway

        uses: bervProject/railway-deploy@v1
```

9.3 Environment Strategy

Environment	Purpose	Infrastructure
Development	Local development	Docker Compose

Staging	Pre-production testing	Railway (preview)
Production	Live application	Railway + Vercel
Demo	Sales demonstrations	Separate Vercel preview

9.4 Monitoring Stack

Tool	Purpose	Cost
Sentry	Error tracking	Free tier available
Grafana Cloud	Metrics & dashboards	Free tier
Better Stack	Uptime monitoring	Free tier
PostHog	Product analytics	Free tier

10. Security Considerations

10.1 Security Measures

Data Protection

- Encryption at rest (AES-256)
- Encryption in transit (TLS 1.3)
- Database-level encryption
- Secure key management (environment variables)

Application Security

- Input validation on all endpoints
- SQL injection prevention (parameterised queries)
- XSS prevention (Content Security Policy)
- CSRF protection (SameSite cookies)
- Rate limiting and DDoS protection

Authentication Security

- Argon2id password hashing
- Secure session management
- MFA support (TOTP)
- Account lockout after failed attempts

10.2 Compliance Considerations

Regulation	Relevance	Key Requirements
GDPR	EU customers	Data privacy, right to deletion
SOC 2	Enterprise sales	Security controls audit
HIPAA	If healthcare clients	PHI protection
PCI DSS	Payment processing	Card data security

10.3 Security Checklist

- Implement CSP headers
- Enable HSTS
- Regular dependency updates
- Security scanning in CI/CD
- Penetration testing (before launch)

- Bug bounty program (post-launch)

11. Cost Analysis

11.1 MVP Infrastructure Costs (Monthly)

Service	Provider	Estimated Cost
Database (PostgreSQL)	Supabase Pro	$25
Backend Hosting	Railway	$50-100
Frontend Hosting	Vercel Pro	$20
Redis Cache	Upstash	$10
File Storage	Cloudflare R2	$5-20
Email Service	Resend	$20
Monitoring	Sentry + Grafana	$0 (free tiers)
Domain & SSL	Cloudflare	$0-15
AI APIs	OpenAI/Anthropic	$50-200
Total MVP		$180-410/month

11.2 Scaling Costs (Growth Stage)

Service	Provider	Estimated Cost
Database	Supabase Pro+	$75-200
Backend Hosting	Railway/Render	$200-500
Frontend	Vercel Pro	$40-100
Redis	Upstash Pro	$50-100
Storage	R2/S3	$50-100
Email	Resend/SendGrid	$50-100
Monitoring	Paid tiers	$50-100
AI APIs	OpenAI/Anthropic	$300-1,000
Total Growth		$815-2,200/month

11.3 Development Costs (Estimated)

Phase	Duration	Team Size	Est. Cost
MVP (Core modules)	4-6 months	3-4 devs	$50-80K
Beta (All modules)	3-4 months	4-5 devs	$40-60K

Launch Polish	2-3 months	4-5 devs	$30-50K
Total to Launch	9-13 months		$120-190K

Note: Costs assume hiring contractors/freelancers. Full-time team would be different.

12. Development Roadmap

12.1 Phase 1: Foundation (Months 1-2)

- Set up development environment and CI/CD
- Implement authentication system (Supabase Auth)
- Build multi-tenant architecture
- Create a user-type access control system
- Develop base UI component library
- Set up database schema and migrations

12.2 Phase 2: Core Modules (Months 3-5)

- CRM Module: Deals, contacts, pipeline
- Projects Module: Tasks, time tracking
- Content Module: Calendar, library, creation
- Analytics Module: Dashboard, basic metrics
- Build a notification system
- Implement file upload/storage

12.3 Phase 3: Extended Modules (Months 6-8)

- Finance Module: Invoicing, payments
- HR Module: Employee management, payroll
- SEO Module: Audits, keyword tracking
- Social Media Module: Publishing, engagement
- Client Portal: External access, approvals

12.4 Phase 4: AI Integration (Months 9-10)

- Integrate LLM for content generation
- Build prediction models (lead scoring)
- Implement AI suggestions system
- Add confidence indicators and explanations
- Implement usage limits by plan

12.5 Phase 5: Polish & Launch (Months 11-13)

- Performance optimisation
- Security audit and penetration testing
- Documentation and help center

- Beta testing with early customers
- Bug fixes and refinements
- Production deployment and launch

12.6 Post-Launch Priorities

- Mobile app optimisation (PWA)
- Third-party integrations (Zapier, etc.)
- Advanced analytics and reporting
- White-label enhancements
- Additional AI capabilities