IT Services Agency Platform - Implementation Prompt Roadmap

Based on your comprehensive gap analysis and the MVP requirements from the PDF, here's a detailed markdown file with precision-engineered prompts for your AI coding agents to systematically implement each missing component.

📋 Critical Path: Tier 0 - Firebreak (Today ASAP)

1. Database Architecture & Schema Migration

Prompt:

```
URGENT: Database Migration & Schema Completion for IT Services Platform

CONTEXT: Currently using Prisma with SQLite, need to implement full PostgreSQL schema per MVP requirements with all missing models.

TASK:
1. Update database provider in schema.prisma from SQLite to PostgreSQL
2. Add ALL missing models from MVP requirements:
   - TimeEntry (id, user_id, task_id, hours, billable, date, description, created_at, updated_at)
   - Invoice (id, client_id, project_id, amount, status, due_date, line_items, created_at, updated_at)
   - Task (id, project_id, assignee_id, title, description, status, priority, due_date, dependencies, created_at, updated_at)
   - ActivityLog (id, user_id, action, entity_type, entity_id, changes, ip_address, created_at)
   - Message (id, sender_id, receiver_id, project_id, content, read_at, created_at)
   - Document (id, project_id, user_id, filename, file_path, file_size, mime_type, version, created_at)
   - TeamMember (id, user_id, project_id, role, hourly_rate, allocated_hours, created_at)
   - Lead (id, company_name, contact_name, email, phone, source, status, value,
```

```
created_at)
   - Deal (id, lead_id, title, value, stage, probability, close_date, created_at)
   - ContactLog (id, client_id, user_id, type, content, created_at)
   - ResourceProfile (id, user_id, skills, hourly_rate, capacity, availability,
created_at)

3. Define proper relationships between all models
4. Create migration files for existing data preservation
5. Update .env.example with PostgreSQL connection variables
6. Create seed script with realistic test data for all models

ACCEPTANCE CRITERIA:
- `npx prisma migrate dev` runs without errors
- `npx prisma db seed` populates all tables with test data
- All model relationships work correctly
- Database can handle complex queries across related tables

TECHNICAL NOTES:
- Use proper indexes for performance
- Add constraints for data integrity
- Include soft delete capabilities where appropriate
- Follow PostgreSQL best practices for naming and types
```

2. Real Data Services Implementation

Prompt:

```
CRITICAL: Replace All Mock APIs with Real Prisma-Backed Services

CONTEXT: Multiple API endpoints return hardcoded mock data instead of querying the
database, breaking the entire platform functionality.

TASK:
Replace these mock endpoints with real Prisma operations:

1. `/api/user/projects` - Currently returns array, should return {projects: [...]} with
real data
2. `/api/user/tasks` - Replace mock task array with Prisma task queries filtered by user
3. `/api/user/invoices` - Replace mock invoices with real Invoice model queries
```

```
4. `/api/user/messages` - Implement real messaging system with Message model
5. `/api/user/documents` - Connect to real Document model with file management
6. `/api/admin/*` - Replace all admin dashboard mock data with live database queries

SPECIFIC FIXES NEEDED:
- Fix API response format mismatch (array vs object with key)
- Add proper pagination (limit, offset, total count)
- Add filtering by status, date range, assigned user
- Add proper error handling for not found, validation errors
- Add proper TypeScript types for all responses
- Include related data (projects with client info, tasks with assignee details)

FOR EACH ENDPOINT:
1. Update route handler to use Prisma client
2. Add Zod validation for request bodies
3. Add proper error responses (400, 404, 500)
4. Add pagination and filtering support
5. Update TypeScript types
6. Add basic rate limiting

ACCEPTANCE CRITERIA:
- All user portal pages show real data from database
- Adding/editing records updates database and reflects in UI immediately
- Error states handled gracefully with user-friendly messages
- API responses match frontend expectations exactly
- No hardcoded mock data remains in production endpoints
```

3. Admin Dashboard Live Data Integration

Prompt:

```
URGENT: Fix Admin Dashboard Static Data Issue

CONTEXT: Admin dashboard shows hardcoded static cards and demo data instead of live
database information, making it unusable for real business operations.

TASK:
Update all admin dashboard components to use real-time data:
```

1. **Dashboard Overview Cards:**
   - Total Projects: Query actual project count with status breakdown
   - Active Clients: Real client count with recent activity
   - Revenue Metrics: Calculate from real invoice and payment data
   - Team Utilization: Compute from actual time entries and resource allocation

2. **Project Management Section:**
   - Replace demo project rows with Prisma queries
   - Add real project creation, editing, status updates
   - Implement project assignment to team members
   - Add project timeline and milestone tracking

3. **Client Management:**
   - Real client CRUD operations
   - Client project history and communication logs
   - Invoice and payment tracking per client

4. **Team & Resource Management:**
   - Real user/team member data with roles and skills
   - Actual workload and utilization calculations
   - Resource allocation and capacity planning

5. **Financial Dashboard:**
   - Real invoice generation and tracking
   - Payment status monitoring
   - Revenue and profitability reporting
   - Expense tracking and categorization

IMPLEMENTATION DETAILS:
- Add loading states for all data fetching
- Implement error boundaries for failed API calls
- Add real-time updates using SWR or React Query
- Include data refresh mechanisms
- Add export functionality for reports

ACCEPTANCE CRITERIA:
- Admin can perform all CRUD operations from UI
- Dashboard reflects real business metrics
- Changes appear immediately without page refresh
- Loading and error states provide clear feedback

```
- All buttons and forms are fully functional
```

4. Authentication & RBAC Enhancement

Prompt:

```
SECURITY CRITICAL: Implement Complete RBAC System

CONTEXT: Current authentication has basic roles but lacks fine-grained permissions and proper access control enforcement.

TASK:
1. **Enhanced Role System:**
   - SUPER_ADMIN: Full system access
   - ADMIN: Company management, user management
   - PROJECT_MANAGER: Project and team management
   - TEAM_MEMBER: Task management, time tracking
   - CLIENT: Portal access, project viewing

2. **Permission-Based Access Control:**
   - Create permissions enum (CREATE_PROJECT, VIEW_FINANCIALS, MANAGE_USERS, etc.)
   - Map roles to permissions in database
   - Implement middleware for route protection

3. **API Route Protection:**
   - Create `lib/auth/middleware.ts` with permission checking
   - Protect every API endpoint with appropriate permission requirements
   - Add user context to all protected routes

4. **Frontend Permission Handling:**
   - Hide/show UI elements based on user permissions
   - Add permission checking hooks (usePermissions, useCanAccess)
   - Implement route guards for protected pages

5. **Session & Token Management:**
   - Implement JWT refresh token rotation
   - Add session timeout handling
   - Include audit logging for authentication events
```

```
CODE STRUCTURE:
```

// lib/auth/permissions.ts
export enum Permission {
CREATE_PROJECT = 'create_project',
VIEW_FINANCIALS = 'view_financials',
MANAGE_USERS = 'manage_users',
// ... all permissions
}

// lib/auth/middleware.ts
export function requirePermissions(permissions: Permission[]) {
// Middleware implementation
}

// Usage in API routes
export async function POST(req: NextRequest) {
const user = await requirePermissions([Permission.CREATE_PROJECT]);
// Route implementation
}

```
ACCEPTANCE CRITERIA:
- Users can only access features allowed by their role
- API endpoints reject unauthorized requests with 403
- UI dynamically shows/hides features based on permissions
- All admin functions require proper authorization
- Audit log captures all permission-related activities
```

📊 Tier 1 - Core Business Features

5. Professional Services Automation (PSA) Core

Prompt:

```
BUSINESS CRITICAL: Implement PSA Core Module

CONTEXT: The central nervous system of the IT services platform - unified dashboard with
```

```
automation and integration capabilities as specified in MVP.

TASK:
1. **Unified Dashboard Implementation:**
   - Real-time project health monitoring with status indicators
   - Resource utilization visualization with capacity warnings
   - Financial performance metrics with profit/loss tracking
   - Client satisfaction scores and feedback integration
   - Automated alert system for project risks and deadlines

2. **Process Automation Engine:**
   - Workflow templates for common IT service processes
   - Automated task creation based on project milestones
   - Email notifications for project updates and deadlines
   - Automated invoice generation from time entries
   - Client onboarding automation workflows

3. **Integration Hub Setup:**
   - Webhook infrastructure for third-party integrations
   - API endpoints for external tool connections
   - Data synchronization services
   - Integration marketplace foundation

4. **Business Intelligence Dashboard:**
   - Project profitability analysis with drill-down capabilities
   - Resource efficiency reports and recommendations
   - Client lifetime value calculations
   - Predictive analytics for project completion dates

IMPLEMENTATION STRUCTURE:
```

// lib/psa/dashboard.ts
export class PSADashboard {
async getProjectHealth(): Promise<ProjectHealthMetrics>
async getResourceUtilization(): Promise<ResourceMetrics>
async getFinancialSummary(): Promise<FinancialMetrics>
async getClientSatisfaction(): Promise<ClientMetrics>
}

// lib/psa/automation.ts
export class WorkflowEngine {
async executeWorkflow(workflowId: string, context: any)
async createWorkflowTemplate(template: WorkflowTemplate)
async scheduleAutomatedTasks(projectId: string)
}

```
ACCEPTANCE CRITERIA:
- Dashboard provides real-time business insights
- Automation reduces manual administrative tasks by 60%
- Integration hub accepts webhook connections
- All metrics update automatically and accurately
- Workflow templates streamline common processes
```

6. Advanced Project Management System

Prompt:

```
PROJECT MANAGEMENT CORE: Implement Complete Project Lifecycle Management

CONTEXT: Need full project management capabilities with Gantt charts, dependencies, risk
management, and multi-methodology support per MVP requirements.

TASK:
1. **Project Creation & Templates:**
   - Project wizard with methodology selection (Agile, Waterfall, Hybrid)
   - Pre-built templates for common IT services (web development, system integration,
etc.)
   - Resource estimation and budget planning tools
   - Client collaboration setup during project initialization

2. **Advanced Project Features:**
   - Gantt chart visualization with drag-and-drop task scheduling
   - Task dependencies and critical path calculation
   - Milestone tracking with automated progress updates
   - Risk register with impact/probability scoring
   - Change request management workflow
```

```
3. **Team Collaboration Tools:**
   - Real-time project chat and file sharing
   - Task assignment with skill-based recommendations
   - Progress tracking with time estimation accuracy
   - Team workload balancing and conflict detection

4. **Client-Facing Project Views:**
   - Client portal project dashboard with filtered information
   - Progress reports with customizable frequency
   - Document sharing with version control
   - Feedback collection and approval workflows

TECHNICAL IMPLEMENTATION:
- Use libraries like `@dhtmlx/gantt` or `frappe-gantt` for Gantt charts
- Implement WebSocket connections for real-time updates
- Create comprehensive project analytics and reporting
- Add project templates and cloning functionality

DATABASE EXTENSIONS:
```

```
-- Additional project-related tables
CREATE TABLE project_templates (
id UUID PRIMARY KEY,
name VARCHAR NOT NULL,
methodology VARCHAR NOT NULL,
tasks_template JSONB,
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE task_dependencies (
id UUID PRIMARY KEY,
task_id UUID REFERENCES tasks(id),
depends_on_task_id UUID REFERENCES tasks(id),
dependency_type VARCHAR DEFAULT 'finish_to_start'
);

CREATE TABLE project_risks (
id UUID PRIMARY KEY,
```

project_id UUID REFERENCES projects(id),
title VARCHAR NOT NULL,
description TEXT,
impact_score INTEGER CHECK (impact_score BETWEEN 1 AND 5),
probability_score INTEGER CHECK (probability_score BETWEEN 1 AND 5),
mitigation_plan TEXT,
status VARCHAR DEFAULT 'open'
);

```
ACCEPTANCE CRITERIA:
- Projects can be created using methodology-specific templates
- Gantt charts display accurate dependencies and critical paths
- Risk management functionality identifies and tracks project risks
- Client portal provides appropriate project visibility
- Resource allocation prevents team member overallocation
```

7. Financial Management & Billing Engine

Prompt:

```
FINANCIAL CORE: Implement Complete Billing and Financial Management

CONTEXT: Currently mock invoices only. Need full financial engine with multiple billing
models, automated invoicing, and profitability tracking per MVP requirements.

TASK:
1. **Multi-Model Billing System:**
   - Hourly rate billing with time entry integration
   - Fixed-fee project billing with milestone-based invoicing
   - Retainer billing with usage tracking
   - Recurring subscription billing for maintenance contracts
   - Mixed billing models within single projects

2. **Automated Invoice Generation:**
   - Time-based invoice creation from approved time entries
   - Milestone-based invoice generation on project completion
   - Recurring invoice automation for retainer clients
   - Invoice customization with company branding
```

```
   - Multi-currency support for international clients

3. **Payment Processing Integration:**
   - Stripe integration for online payments
   - PayPal support for client flexibility
   - Bank transfer and check payment tracking
   - Payment reminder automation
   - Late fee calculation and application

4. **Financial Analytics:**
   - Project profitability analysis with cost breakdown
   - Client lifetime value calculations
   - Revenue forecasting based on pipeline
   - Cash flow projections and analysis
   - Expense tracking and categorization

IMPLEMENTATION STRUCTURE:
```

// lib/billing/engine.ts
export class BillingEngine {
async generateInvoice(projectId: string, billingType: BillingType): Promise<Invoice>
async processPayment(invoiceId: string, paymentDetails: PaymentDetails): Promise<PaymentResult>
async calculateProjectProfitability(projectId: string): Promise<ProfitabilityMetrics>
async generateRecurringInvoices(): Promise<Invoice[]>
}

// lib/billing/payment-processor.ts
export class PaymentProcessor {
async processStripePayment(amount: number, currency: string): Promise<PaymentResult>
async processPayPalPayment(amount: number, currency: string): Promise<PaymentResult>
async recordManualPayment(invoiceId: string, details: ManualPaymentDetails): Promise<void>
}

```
DATABASE SCHEMA:
```

CREATE TABLE invoices (
id UUID PRIMARY KEY,

```
client_id UUID REFERENCES clients(id),
project_id UUID REFERENCES projects(id),
invoice_number VARCHAR UNIQUE NOT NULL,
amount DECIMAL(10,2) NOT NULL,
currency VARCHAR(3) DEFAULT 'USD',
status VARCHAR DEFAULT 'draft', -- draft, sent, paid, overdue, cancelled
due_date DATE NOT NULL,
line_items JSONB,
tax_amount DECIMAL(10,2) DEFAULT 0,
discount_amount DECIMAL(10,2) DEFAULT 0,
created_at TIMESTAMP DEFAULT NOW(),
paid_at TIMESTAMP
);

CREATE TABLE payments (
id UUID PRIMARY KEY,
invoice_id UUID REFERENCES invoices(id),
amount DECIMAL(10,2) NOT NULL,
payment_method VARCHAR NOT NULL, -- stripe, paypal, bank_transfer, check
payment_reference VARCHAR,
processed_at TIMESTAMP DEFAULT NOW(),
status VARCHAR DEFAULT 'completed'
);

CREATE TABLE expenses (
id UUID PRIMARY KEY,
project_id UUID REFERENCES projects(id),
user_id UUID REFERENCES users(id),
category VARCHAR NOT NULL,
amount DECIMAL(10,2) NOT NULL,
description TEXT,
receipt_url VARCHAR,
date DATE NOT NULL,
reimbursed BOOLEAN DEFAULT FALSE,
created_at TIMESTAMP DEFAULT NOW()
);
```

```
ACCEPTANCE CRITERIA:
- Invoices generate automatically from time entries and project milestones
- Multiple payment methods process successfully
- Financial reports provide accurate profitability insights
- Recurring billing works automatically for retainer clients
- Integration with accounting software (QuickBooks/Xero) ready
```

8. Enhanced CRM System

Prompt:

```
CRM IMPLEMENTATION: Complete Lead-to-Client Lifecycle Management

CONTEXT: No CRM capabilities currently exist. Need full pipeline management, lead
tracking, and 360-degree client view per MVP requirements.

TASK:
1. **Lead Management System:**
   - Lead capture from multiple sources (web forms, referrals, cold outreach)
   - Lead qualification scoring based on company size, budget, timeline
   - Automated lead nurturing sequences with email campaigns
   - Lead assignment to sales team members
   - Conversion tracking from lead to opportunity to client

2. **Sales Pipeline Management:**
   - Customizable pipeline stages (Prospecting, Qualification, Proposal, Negotiation,
Closed)
   - Probability weighting for revenue forecasting
   - Activity tracking and follow-up reminders
   - Proposal generation and tracking
   - Win/loss analysis and improvement recommendations

3. **360-Degree Client View:**
   - Complete interaction history across all touchpoints
   - Project history with performance metrics
   - Communication log with team members
   - Invoice and payment history
   - Support ticket and issue resolution tracking
```

```
4. **Contract Management:**
   - Digital contract creation from templates
   - Electronic signature integration (DocuSign/HelloSign)
   - Contract renewal tracking and notifications
   - Amendment and version control
   - Compliance and legal review workflows

DATABASE IMPLEMENTATION:
```

CREATE TABLE leads (
id UUID PRIMARY KEY,
company_name VARCHAR NOT NULL,
contact_name VARCHAR NOT NULL,
email VARCHAR NOT NULL,
phone VARCHAR,
source VARCHAR, -- website, referral, cold_call, etc.
status VARCHAR DEFAULT 'new', -- new, contacted, qualified, converted, lost
qualification_score INTEGER,
estimated_value DECIMAL(10,2),
estimated_close_date DATE,
assigned_to UUID REFERENCES users(id),
notes TEXT,
created_at TIMESTAMP DEFAULT NOW(),
last_contacted_at TIMESTAMP
);

CREATE TABLE opportunities (
id UUID PRIMARY KEY,
lead_id UUID REFERENCES leads(id),
title VARCHAR NOT NULL,
description TEXT,
value DECIMAL(10,2) NOT NULL,
currency VARCHAR(3) DEFAULT 'USD',
stage VARCHAR DEFAULT 'prospecting',
probability INTEGER CHECK (probability BETWEEN 0 AND 100),
expected_close_date DATE,

```
actual_close_date DATE,
owner_id UUID REFERENCES users(id),
competitor_info TEXT,
next_step TEXT,
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE contact_logs (
id UUID PRIMARY KEY,
client_id UUID REFERENCES clients(id),
lead_id UUID REFERENCES leads(id),
user_id UUID REFERENCES users(id),
contact_type VARCHAR NOT NULL, -- email, call, meeting, demo
subject VARCHAR NOT NULL,
content TEXT,
outcome VARCHAR,
next_action TEXT,
next_action_date DATE,
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE contracts (
id UUID PRIMARY KEY,
client_id UUID REFERENCES clients(id),
opportunity_id UUID REFERENCES opportunities(id),
title VARCHAR NOT NULL,
contract_type VARCHAR NOT NULL,
value DECIMAL(10,2),
start_date DATE,
end_date DATE,
auto_renewal BOOLEAN DEFAULT FALSE,
status VARCHAR DEFAULT 'draft', -- draft, sent, signed, active, expired
contract_url VARCHAR,
signed_at TIMESTAMP,
created_at TIMESTAMP DEFAULT NOW()
);
```

```
FRONTEND COMPONENTS:
- Pipeline kanban board with drag-and-drop
- Lead scoring dashboard with automation rules
- Client interaction timeline with filtering
- Contract management interface with e-signature
- Revenue forecasting with pipeline analytics

ACCEPTANCE CRITERIA:
- Leads can be captured and managed through complete lifecycle
- Sales pipeline provides accurate revenue forecasting
- Client profiles show comprehensive interaction history
- Contracts can be generated, sent, and signed digitally
- CRM data integrates seamlessly with project management
```

9. Time Tracking & Resource Management

Prompt:

```
TIME & RESOURCE OPTIMIZATION: Implement Complete Resource Planning System

CONTEXT: No time tracking exists currently. Need comprehensive time management, resource
allocation, and utilization optimization per MVP requirements.

TASK:
1. **Time Tracking System:**
   - Built-in timer with task categorization
   - Manual time entry with approval workflows
   - Mobile-friendly time logging
   - Time entry validation and correction
   - Integration with project tasks and billing

2. **Resource Management:**
   - Team member skill matrix and competency tracking
   - Capacity planning with vacation and availability management
   - Workload balancing with overallocation warnings
   - Resource allocation optimization suggestions
   - Cross-project resource sharing and coordination
```

```
3. **Utilization Analytics:**
   - Individual and team utilization reporting
   - Billable vs non-billable time analysis
   - Project efficiency metrics and benchmarking
   - Resource cost analysis and profitability impact
   - Capacity forecasting for future projects

4. **Advanced Planning Features:**
   - Resource demand forecasting based on project pipeline
   - Skill gap analysis and training recommendations
   - Optimal team composition for new projects
   - Contractor and freelancer management
   - Resource budget planning and cost control

IMPLEMENTATION:
```

// lib/time-tracking/timer.ts
export class TimeTracker {
async startTimer(userId: string, taskId: string): Promise<TimeEntry>
async stopTimer(entryId: string): Promise<TimeEntry>
async logManualEntry(entry: TimeEntryInput): Promise<TimeEntry>
async getWeeklyTimesheet(userId: string, weekStart: Date): Promise<TimeEntry[]>
}

// lib/resource-management/allocator.ts
export class ResourceAllocator {
async optimizeTeamAllocation(projectId: string): Promise<AllocationPlan>
async checkResourceAvailability(userId: string, dateRange: DateRange): Promise<AvailabilityInfo>
async suggestTeamComposition(projectRequirements: ProjectRequirements): Promise<TeamSuggestion>
async calculateUtilization(userId: string, period: Period): Promise<UtilizationMetrics>
}

```
DATABASE SCHEMA:
```

CREATE TABLE time_entries (
id UUID PRIMARY KEY,
user_id UUID REFERENCES users(id),

```
task_id UUID REFERENCES tasks(id),
project_id UUID REFERENCES projects(id),
start_time TIMESTAMP NOT NULL,
end_time TIMESTAMP,
duration_minutes INTEGER,
description TEXT,
billable BOOLEAN DEFAULT TRUE,
hourly_rate DECIMAL(8,2),
approved BOOLEAN DEFAULT FALSE,
approved_by UUID REFERENCES users(id),
approved_at TIMESTAMP,
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE resource_profiles (
id UUID PRIMARY KEY,
user_id UUID REFERENCES users(id),
skills JSONB, -- {"javascript": 5, "react": 4, "node": 3}
hourly_rate DECIMAL(8,2),
weekly_capacity_hours INTEGER DEFAULT 40,
availability_schedule JSONB, -- weekly schedule with hours
cost_per_hour DECIMAL(8,2), -- internal cost for profitability calculations
certifications TEXT[],
years_experience INTEGER,
updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE resource_allocations (
id UUID PRIMARY KEY,
user_id UUID REFERENCES users(id),
project_id UUID REFERENCES projects(id),
role VARCHAR NOT NULL,
allocated_hours_per_week DECIMAL(4,1),
start_date DATE,
end_date DATE,
billing_rate DECIMAL(8,2),
```

created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE availability_calendar (
id UUID PRIMARY KEY,
user_id UUID REFERENCES users(id),
date DATE NOT NULL,
available_hours DECIMAL(4,1) DEFAULT 8.0,
status VARCHAR DEFAULT 'available', -- available, pto, sick, holiday, training
notes TEXT,
UNIQUE(user_id, date)
);

```
UI COMPONENTS:
- Timer widget with task selection and project categorization
- Resource allocation calendar with drag-and-drop scheduling
- Utilization dashboard with team and individual metrics
- Capacity planning interface with visual workload distribution
- Skills matrix management with competency tracking

ACCEPTANCE CRITERIA:
- Team members can track time accurately with minimal friction
- Resource allocation prevents overallocation and optimizes utilization
- Managers can plan resource needs for future projects
- Utilization reports provide actionable insights for efficiency improvement
- Integration with billing ensures accurate invoicing from time data
```

10. Document Management System

Prompt:

```
DOCUMENT MANAGEMENT: Implement Secure File Storage and Collaboration

CONTEXT: Document management is referenced but not implemented. Need complete file
management with S3 integration, versioning, and permissions per MVP requirements.

TASK:
1. **File Storage Infrastructure:**
```

```
   - AWS S3 integration with secure bucket configuration
   - File upload with progress tracking and validation
   - Multiple file format support with type validation
   - Automatic virus scanning integration
   - CDN integration for fast file delivery

2. **Document Organization:**
   - Hierarchical folder structure per project/client
   - Document categorization and tagging system
   - Advanced search with metadata and content indexing
   - Document templates and standardization
   - Bulk operations for file management

3. **Version Control System:**
   - Automatic version tracking for document updates
   - Version comparison and diff visualization
   - Rollback capabilities to previous versions
   - Collaborative editing with conflict resolution
   - Document approval workflows

4. **Access Control & Security:**
   - Granular permissions per document/folder
   - Client-specific document sharing controls
   - Audit trail for all document activities
   - Download tracking and restrictions
   - Watermarking for sensitive documents

IMPLEMENTATION:
```

// lib/storage/s3-manager.ts
export class S3DocumentManager {
async uploadFile(file: File, projectId: string, userId: string): Promise<DocumentRecord>
async downloadFile(documentId: string, userId: string): Promise<DownloadUrl>
async deleteFile(documentId: string, userId: string): Promise<void>
async createFolder(name: string, parentId?: string, projectId: string): Promise<Folder>
async searchDocuments(query: string, filters: SearchFilters): Promise<DocumentRecord[]>
}

```
// lib/documents/version-manager.ts
export class DocumentVersionManager {
async createNewVersion(documentId: string, file: File, userId: string): Promise<DocumentVersion>
async getVersionHistory(documentId: string): Promise<DocumentVersion[]>
async revertToVersion(documentId: string, versionId: string): Promise<DocumentRecord>
async compareVersions(versionId1: string, versionId2: string): Promise<ComparisonResult>
}
```

```
DATABASE SCHEMA:
```

```
CREATE TABLE documents (
id UUID PRIMARY KEY,
project_id UUID REFERENCES projects(id),
client_id UUID REFERENCES clients(id),
folder_id UUID,
name VARCHAR NOT NULL,
description TEXT,
file_path VARCHAR NOT NULL,
file_size BIGINT NOT NULL,
mime_type VARCHAR NOT NULL,
checksum VARCHAR NOT NULL,
tags TEXT[],
current_version INTEGER DEFAULT 1,
uploaded_by UUID REFERENCES users(id),
status VARCHAR DEFAULT 'active', -- active, archived, deleted
access_level VARCHAR DEFAULT 'project', -- public, project, restricted, private
download_count INTEGER DEFAULT 0,
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE document_versions (
id UUID PRIMARY KEY,
document_id UUID REFERENCES documents(id),
version_number INTEGER NOT NULL,
```

```
file_path VARCHAR NOT NULL,
file_size BIGINT NOT NULL,
checksum VARCHAR NOT NULL,
changes_summary TEXT,
uploaded_by UUID REFERENCES users(id),
created_at TIMESTAMP DEFAULT NOW(),
UNIQUE(document_id, version_number)
);

CREATE TABLE document_permissions (
id UUID PRIMARY KEY,
document_id UUID REFERENCES documents(id),
user_id UUID REFERENCES users(id),
client_id UUID REFERENCES clients(id),
permission_type VARCHAR NOT NULL, -- read, write, delete, share
granted_by UUID REFERENCES users(id),
expires_at TIMESTAMP,
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE document_folders (
id UUID PRIMARY KEY,
project_id UUID REFERENCES projects(id),
parent_id UUID REFERENCES document_folders(id),
name VARCHAR NOT NULL,
description TEXT,
created_by UUID REFERENCES users(id),
created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE document_activities (
id UUID PRIMARY KEY,
document_id UUID REFERENCES documents(id),
user_id UUID REFERENCES users(id),
action VARCHAR NOT NULL, -- upload, download, delete, share, view
details JSONB,
ip_address INET,
```

user_agent TEXT,
created_at TIMESTAMP DEFAULT NOW()
);

```
SECURITY CONFIGURATIONS:
```

// S3 Bucket Configuration
const s3BucketConfig = {
versioning: true,
encryption: 'AES256',
corsRules: [/* CORS for frontend uploads
/],lifecycleRules: [/ Auto-archive old versions */],
publicReadWrite: false, // All access through signed URLs
};

// File upload validation
const fileValidation = {
maxSize: 100 * 1024 * 1024, // 100MB
allowedTypes: [
'application/pdf',
'application/msword',
'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
'image/jpeg',
'image/png',
'text/plain'
],
scanForViruses: true,
quarantineOnThreat: true
};

```
UI FEATURES:
- Drag-and-drop file upload with progress bars
- File explorer with breadcrumb navigation
- Document preview for supported file types
- Version history with visual timeline
```

```
- Permission management interface
- Advanced search with filters and sorting

ACCEPTANCE CRITERIA:
- Files upload securely to S3 with proper organization
- Version control tracks all document changes accurately
- Permissions system controls access appropriately
- Search functionality finds documents quickly and accurately
- Client portal shows only authorized documents
- Audit trail captures all document interactions
```

☸ Tier 2 - Technical Infrastructure

11. DevOps & CI/CD Pipeline

Prompt:

```
DEVOPS FOUNDATION: Complete CI/CD and Deployment Infrastructure

CONTEXT: No DevOps infrastructure exists. Need complete pipeline setup for automated
testing, building, and deployment per MVP requirements.

TASK:
1. **Containerization:**
   - Create multi-stage Dockerfile for production optimization
   - Docker Compose for local development environment
   - Container health checks and resource limits
   - Security scanning for container vulnerabilities

2. **CI/CD Pipeline (GitHub Actions):**
   - Automated linting and type checking
   - Unit and integration test execution
   - Build and deployment automation
   - Environment-specific deployment strategies
   - Rollback mechanisms for failed deployments

3. **Infrastructure as Code:**
   - Terraform or CloudFormation templates
   - Environment provisioning automation
```

```
   - Database migration handling
   - SSL certificate management
   - Load balancer and auto-scaling configuration

4. **Monitoring & Observability:**
   - Application performance monitoring (APM)
   - Error tracking and alerting
   - Log aggregation and analysis
   - Uptime monitoring and health checks
   - Performance metrics and KPI dashboards

FILES TO CREATE:
```

.github/workflows/ci-cd.yml

name: CI/CD Pipeline

on:

push:

branches: [main, develop]

pull_request:

branches: [main]

jobs:

lint-and-test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: '18'

- name: Install dependencies

run: npm ci

- name: Run linting

run: npm run lint

- name: Run type checking

run: npm run type-check

- name: Run tests
run: npm run test:ci
env:
DATABASE_URL: ${{ secrets.TEST_DATABASE_URL }}

build-and-deploy:
needs: lint-and-test
runs-on: ubuntu-latest
if: github.ref == 'refs/heads/main'
steps:
- name: Build Docker image
run: docker build -t $secrets.DOCKER_REGISTRY/it-services-platform$:{{ github.sha }} .
- name: Push to registry
run: docker push $secrets.DOCKER_REGISTRY/it-services-platform$:{{ github.sha }}
- name: Deploy to production
run: |
Deployment script

Dockerfile

FROM node:18-alpine AS base
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production && npm cache clean --force

FROM node:18-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM base AS production
COPY --from=build /app/dist ./dist
COPY --from=build /app/public ./public

```
EXPOSE 3000
CMD ["npm", "start"]
```

docker-compose.yml

```
version: '3.8'
services:
app:
build: .
ports:
- "3000:3000"
environment:
- DATABASE_URL=postgresql://user:password@postgres:5432/it_services_db
- REDIS_URL=redis://redis:6379
depends_on:
- postgres
- redis

postgres:
image: postgres:15-alpine
environment:
- POSTGRES_DB=it_services_db
- POSTGRES_USER=user
- POSTGRES_PASSWORD=password
volumes:
- postgres_data:/var/lib/postgresql/data
ports:
- "5432:5432"

redis:
image: redis:7-alpine
ports:
- "6379:6379"
volumes:
- redis_data:/data
```

volumes:
postgres_data:
redis_data:

```
MONITORING SETUP:
```

// lib/monitoring/apm.ts
import { NodeSDK } from '@opentelemetry/sdk-node';
import { instrumentations } from './instrumentations';

export const sdk = new NodeSDK({
instrumentations: [instrumentations],
serviceName: 'it-services-platform',
serviceVersion: process.env.APP_VERSION || '1.0.0',
});

// lib/monitoring/health-check.ts
export async function healthCheck() {
const checks = [
checkDatabase(),
checkRedis(),
checkS3(),
checkExternalAPIs()
];

const results = await Promise.allSettled(checks);
return {
status: results.every(r => r.status === 'fulfilled') ? 'healthy' : 'unhealthy',
checks: results
};
}

```
ACCEPTANCE CRITERIA:
- Code changes trigger automated testing and deployment
- Production deployments are zero-downtime
- Container images are security-scanned and optimized
```

```
- Monitoring provides real-time visibility into system health
- Rollback procedures work reliably in case of issues
```

12. Testing Framework Implementation

Prompt:

```
TESTING INFRASTRUCTURE: Complete Test Suite Setup

CONTEXT: No testing framework exists. Need comprehensive unit, integration, and E2E
testing per MVP quality standards.

TASK:
1. **Unit Testing Setup:**
   - Jest configuration with TypeScript support
   - React Testing Library for component tests
   - API endpoint testing with supertest
   - Database testing with test containers
   - Mock services for external dependencies

2. **Integration Testing:**
   - Full API workflow testing
   - Database integration testing
   - Authentication flow testing
   - File upload and processing testing
   - Third-party integration testing

3. **End-to-End Testing:**
   - Cypress or Playwright setup for browser automation
   - User journey testing for critical paths
   - Cross-browser compatibility testing
   - Performance testing under load
   - Mobile responsiveness testing

4. **Test Data Management:**
   - Factory patterns for test data generation
   - Database seeding for consistent test environments
   - Test isolation and cleanup procedures
   - Snapshot testing for UI components
```

```
CONFIGURATION FILES:
```

// jest.config.js
module.exports = {
preset: 'ts-jest',
testEnvironment: 'node',
setupFilesAfterEnv: ['<rootDir>/tests/setup.ts'],
testMatch: [
'**/tests/**
*.ts','**/?(*.)+(spec|test).ts'
],
collectCoverageFrom: [
'src/
/*.{js,ts}','!src/
*.d.ts','!src/types/**/*'
],
coverageThreshold: {
global: {
branches: 80,
functions: 80,
lines: 80,
statements: 80
}
}
};

// jest.config.frontend.js
module.exports = {
preset: 'ts-jest',
testEnvironment: 'jsdom',
setupFilesAfterEnv: ['<rootDir>/tests/frontend-setup.ts'],
moduleNameMapping: {
'^@/(.*)$': '<rootDir>/src/': 'ts-jest'
}
};

```
// tests/setup.ts
import { PrismaClient } from '@prisma/client';
import { execSync } from 'child_process';

const prisma = new PrismaClient();

beforeAll(async () => {
// Setup test database
execSync('npx prisma migrate reset --force', { stdio: 'inherit' });
await prisma.$connect();
});

afterAll(async () => {
await prisma.$disconnect();
});

beforeEach(async () => {
// Clean up data before each test
const tableNames = await prisma.$queryRawSELECT tablename FROM pg_tables WHERE schemaname = 'public';;

for (const { tablename } of tableNames) {
if (tablename !== '_prisma_migrations') {
await prisma.executeRawUnsafe(`TRUNCATETABLE"{tablename}" CASCADE;`);
}
}
});
```

```
SAMPLE TESTS:
```

```
// tests/api/projects.test.ts
import request from 'supertest';
import { app } from '../../src/app';
import { createTestUser, createTestClient } from '../factories';
```

describe('Project API', () => {
let testUser: any;
let testClient: any;
let authToken: string;

beforeEach(async () => {
testUser = await createTestUser({ role: 'ADMIN' });
testClient = await createTestClient();
authToken = generateTestToken(testUser.id);
});

describe('POST /api/projects', () => {
it('should create a new project', async () => {
const projectData = {
name: 'Test Project',
clientId: testClient.id,
budget: 50000,
startDate: '2023-01-01',
endDate: '2023-06-01'
};

```
  const response = await request(app)
    .post('/api/projects')
    .set('Authorization', `Bearer ${authToken}`)
    .send(projectData)
    .expect(201);

  expect(response.body).toMatchObject({
    id: expect.any(String),
    name: projectData.name,
    clientId: testClient.id,
    status: 'ACTIVE'
  });
});

it('should validate required fields', async () => {
  const response = await request(app)
    .post('/api/projects')
    .set('Authorization', `Bearer ${authToken}`)
```

```
    .send({})
    .expect(400);

  expect(response.body.errors).toContain('Name is required');
});
```

});

});

// **tests**/components/ProjectCard.test.tsx

import { render, screen, fireEvent } from '@testing-library/react';

import { ProjectCard } from '../../src/components/ProjectCard';

import { mockProject } from '../mocks';

describe('ProjectCard', () => {

it('renders project information correctly', () => {

render();

```
expect(screen.getByText(mockProject.name)).toBeInTheDocument();
expect(screen.getByText(mockProject.client.name)).toBeInTheDocument();
expect(screen.getByText(`$${mockProject.budget.toLocaleString()}`)).toBeInTheDocument();
```

});

it('calls onEdit when edit button is clicked', () => {

const onEdit = jest.fn();

render();

```
fireEvent.click(screen.getByRole('button', { name: /edit/i }));
expect(onEdit).toHaveBeenCalledWith(mockProject.id);
```

});

});

```
E2E TEST SETUP:
```

```
// cypress/e2e/project-management.cy.ts
describe('Project Management', () => {
beforeEach(() => {
cy.login('admin@test.com', 'password');
cy.visit('/dashboard/projects');
});

it('should create a new project end-to-end', () => {
cy.get('[data-testid="create-project-btn"]').click();
cy.get('[data-testid="project-name"]').type('E2E Test Project');
cy.get('[data-testid="client-select"]').select('Test Client');
cy.get('[data-testid="budget-input"]').type('50000');
cy.get('[data-testid="submit-btn"]').click();

cy.get('[data-testid="success-message"]').should('contain', 'Project created
successfully');
cy.get('[data-testid="project-list"]').should('contain', 'E2E Test Project');

});

it('should update project status', () => {
cy.get('[data-testid="project-row"]').first().within(() => {
cy.get('[data-testid="status-select"]').select('COMPLETED');
});

cy.get('[data-testid="save-btn"]').click();
cy.get('[data-testid="success-message"]').should('be.visible');

});
});


ACCEPTANCE CRITERIA:
- All critical user journeys covered by E2E tests
- Unit test coverage above 80% for core business logic
- Integration tests validate API contracts and data flow
- Test suite runs in under 5 minutes
```

```
- Failed tests provide clear, actionable error messages
```

13. Security Hardening Implementation

Prompt:

```
SECURITY HARDENING: Implement Enterprise-Grade Security Measures

CONTEXT: Current implementation lacks security hardening. Need comprehensive security
measures per MVP requirements including OWASP compliance.

TASK:
1. **Web Security Headers:**
   - Content Security Policy (CSP) implementation
   - HTTP Strict Transport Security (HSTS)
   - X-Frame-Options and X-Content-Type-Options
   - CORS configuration with domain whitelisting
   - Security headers middleware for all responses

2. **Input Validation & Sanitization:**
   - Comprehensive input validation using Zod schemas
   - SQL injection prevention with parameterized queries
   - XSS protection with output encoding
   - File upload security with type and size validation
   - Rate limiting to prevent abuse and DoS attacks

3. **Authentication Security:**
   - Password hashing with bcrypt and salt
   - JWT token security with rotation and expiration
   - Multi-factor authentication (MFA) support
   - Account lockout policies for failed attempts
   - Session management with secure cookies

4. **Data Protection:**
   - Encryption at rest for sensitive data
   - PII anonymization and pseudonymization
   - Secure backup and recovery procedures
   - Data retention and deletion policies
   - GDPR compliance measures
```

```
IMPLEMENTATION:
```

```
// lib/security/headers.ts
export const securityHeaders = {
'Content-Security-Policy': default-src 'self'; script-src 'self' 'unsafe-inline' https://js.stripe.com; style-src 'self' 'unsafe-inline'; img-src 'self' data: https:; connect-src 'self' https://api.stripe.com; font-src 'self'; object-src 'none'; frame-ancestors 'none'; upgrade-insecure-requests;.replace(/\s+/g, ' ').trim(),
'Strict-Transport-Security': 'max-age=63072000; includeSubDomains; preload',
'X-Content-Type-Options': 'nosniff',
'X-Frame-Options': 'DENY',
'X-XSS-Protection': '1; mode=block',
'Referrer-Policy': 'strict-origin-when-cross-origin',
'Permissions-Policy': 'camera=(), microphone=(), geolocation=()'
};

// middleware/security.ts
import helmet from 'helmet';
import rateLimit from 'express-rate-limit';

export const rateLimiter = rateLimit({
windowMs: 15 * 60 * 1000, // 15 minutes
max: 100, // Limit each IP to 100 requests per windowMs
message: 'Too many requests from this IP, please try again later.',
standardHeaders: true,
legacyHeaders: false,
});

export const securityMiddleware = [
helmet({
contentSecurityPolicy: {
directives: {
defaultSrc: ["'self'"],
scriptSrc: ["'self'", "'unsafe-inline'", "https://js.stripe.com"],
styleSrc: ["'self'", "'unsafe-inline'"],
imgSrc: ["'self'", "data:", "https:"],
```

```
connectSrc: ["'self'", "https://api.stripe.com"],
},
},
hsts: {
maxAge: 63072000,
includeSubDomains: true,
preload: true
}
}),
rateLimiter
];
```

```
// lib/security/validation.ts
import { z } from 'zod';
import DOMPurify from 'isomorphic-dompurify';

// Input validation schemas
export const userRegistrationSchema = z.object({
email: z.string().email().max(254),
password: z.string()
.min(8, 'Password must be at least 8 characters')
.regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@!!%*?&]/,
'Password must contain uppercase, lowercase, number and special character'),
firstName: z.string().min(1).max(50).refine(val => !/<|>|&/.test(val)),
lastName: z.string().min(1).max(50).refine(val => !/<|>|&/.test(val))
});

export const projectSchema = z.object({
name: z.string().min(1).max(100),
description: z.string().optional(),
budget: z.number().positive(),
clientId: z.string().uuid(),
startDate: z.string().datetime(),
endDate: z.string().datetime()
});
```

```
// Input sanitization
export function sanitizeInput(input: string): string {
return DOMPurify.sanitize(input, { ALLOWED_TAGS: [] });
}

export function validateAndSanitize<T>(schema: z.ZodSchema<T>, data: unknown): T {

const validated = schema.parse(data);

// Recursively sanitize string fields
function sanitizeObject(obj: any): any {
if (typeof obj === 'string') {
return sanitizeInput(obj);
}
if (typeof obj === 'object' && obj !== null) {
const sanitized: any = {};
for (const [key, value] of Object.entries(obj)) {
sanitized[key] = sanitizeObject(value);
}
return sanitized;
}
return obj;
}

return sanitizeObject(validated);
}



// lib/security/encryption.ts
import crypto from 'crypto';
import bcrypt from 'bcrypt';

const ENCRYPTION_KEY = process.env.ENCRYPTION_KEY || crypto.randomBytes(32);
const IV_LENGTH = 16;

export async function hashPassword(password: string): Promise<string> {
const saltRounds = 12;
```

```
return bcrypt.hash(password, saltRounds);
}

export async function verifyPassword(password: string, hash: string): Promise<boolean> {
return bcrypt.compare(password, hash);
}

export function encryptPII(text: string): { encrypted: string; iv: string } {
const iv = crypto.randomBytes(IV_LENGTH);
const cipher = crypto.createCipher('aes-256-cbc', ENCRYPTION_KEY);
let encrypted = cipher.update(text, 'utf8', 'hex');
encrypted += cipher.final('hex');

return {
encrypted,
iv: iv.toString('hex')
};
}

export function decryptPII(encrypted: string, ivHex: string): string {
const iv = Buffer.from(ivHex, 'hex');
const decipher = crypto.createDecipher('aes-256-cbc', ENCRYPTION_KEY);
let decrypted = decipher.update(encrypted, 'hex', 'utf8');
decrypted += decipher.final('utf8');
return decrypted;
}
```

```
// lib/security/audit-logging.ts
export interface AuditLogEntry {
userId?: string;
action: string;
resource: string;
resourceId?: string;
changes?: Record<string, any>;
ipAddress: string;
```

userAgent: string;
timestamp: Date;
severity: 'low' | 'medium' | 'high' | 'critical';
}

export class AuditLogger {
static async log(entry: AuditLogEntry): Promise<void> {
await prisma.activityLog.create({
data: {
userId: entry.userId,
action: entry.action,
entityType: entry.resource,
entityId: entry.resourceId,
changes: entry.changes,
ipAddress: entry.ipAddress,
userAgent: entry.userAgent,
severity: entry.severity,
createdAt: entry.timestamp
}
});

```
// Also send to external monitoring for critical events
if (entry.severity === 'critical') {
  await this.sendToSIEM(entry);
}
```

}

private static async sendToSIEM(entry: AuditLogEntry): Promise<void> {
// Send to external SIEM system for security monitoring
}
}

```
OWASP COMPLIANCE CHECKLIST:
- [ ] A1 - Injection: Parameterized queries, input validation
- [ ] A2 - Broken Authentication: Secure session management, MFA
- [ ] A3 - Sensitive Data Exposure: Encryption at rest/transit
```

```
- [ ] A4 - XML External Entities: N/A (no XML processing)
- [ ] A5 - Broken Access Control: RBAC implementation
- [ ] A6 - Security Misconfiguration: Security headers, hardened config
- [ ] A7 - Cross-Site Scripting: Output encoding, CSP
- [ ] A8 - Insecure Deserialization: Safe JSON parsing
- [ ] A9 - Known Vulnerabilities: Dependency scanning
- [ ] A10 - Insufficient Logging: Comprehensive audit logging

ACCEPTANCE CRITERIA:
- OWASP ZAP scan shows no critical or high vulnerabilities
- All user inputs validated and sanitized
- Security headers implemented and verified
- Audit logging captures all sensitive operations
- Rate limiting prevents abuse attacks
- Data encryption protects PII at rest and in transit
```

14. Performance Optimization & Caching

Prompt:

```
PERFORMANCE OPTIMIZATION: Implement Comprehensive Caching and Speed Optimization

CONTEXT: No performance optimization exists. Need Redis caching, CDN integration, and
sub-2-second load times per MVP requirements.

TASK:
1. **Redis Caching Layer:**
   - Session storage in Redis for scalability
   - API response caching with TTL management
   - Database query result caching
   - Real-time data synchronization
   - Cache invalidation strategies

2. **Database Optimization:**
   - Query optimization with proper indexes
   - Connection pooling configuration
   - N+1 query prevention with eager loading
   - Database performance monitoring
   - Query execution plan analysis
```

```
3. **Frontend Performance:**
   - Code splitting and lazy loading
   - Image optimization and compression
   - CDN integration for static assets
   - Browser caching strategies
   - Critical CSS inlining

4. **API Performance:**
   - Response compression with gzip
   - Pagination for large datasets
   - API response time monitoring
   - Background job processing
   - Resource preloading strategies

REDIS IMPLEMENTATION:
```

```
// lib/cache/redis-client.ts
import Redis from 'ioredis';

export class RedisCache {
private client: Redis;

constructor() {
this.client = new Redis({
host: process.env.REDIS_HOST || 'localhost',
port: parseInt(process.env.REDIS_PORT || '6379'),
password: process.env.REDIS_PASSWORD,
retryDelayOnFailover: 100,
lazyConnect: true,
maxRetriesPerRequest: 3
});
}

async get<T>(key: string): Promise<T | null> {
const value = await this.client.get(key);
return value ? JSON.parse(value) : null;
}
```

```
async set(key: string, value: any, ttlSeconds: number = 3600): Promise<void> {
await this.client.setex(key, ttlSeconds, JSON.stringify(value));
}

async del(key: string): Promise<void> {
await this.client.del(key);
}

async invalidatePattern(pattern: string): Promise<void> {
const keys = await this.client.keys(pattern);
if (keys.length > 0) {
await this.client.del(...keys);
}
}

// Cache wrapper for database queries
async cacheQuery<T>(
key: string,
queryFn: () => Promise<T>,
ttl: number = 600
): Promise<T> {
const cached = await this.get<T>(key);
if (cached) return cached;

const result = await queryFn();
await this.set(key, result, ttl);
return result;

}
}

export const cache = new RedisCache();



// lib/cache/strategies.ts
export class CacheStrategies {
// Cache user dashboard data
```

```
static async getUserDashboard(userId: string) {
return cache.cacheQuery(
`dashboard:user:${userId}`,
async () => {
const [projects, tasks, timeEntries] = await Promise.all([
prisma.project.findMany({
where: { teamMembers: { some: { userId } } },
include: { client: true, _count: { select: { tasks: true } } }
}),
prisma.task.findMany({
where: { assigneeId: userId, status: { not: 'COMPLETED' } },
include: { project: { select: { name: true, client: { select: { name: true } } } } }
}),
prisma.timeEntry.findMany({
where: { userId, createdAt: { gte: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000) } },
include: { task: { select: { title: true } } }
})
]);

    return { projects, tasks, timeEntries };
  },
  300 // 5 minutes TTL
);


}

// Cache project analytics
static async getProjectAnalytics(projectId: string) {
return cache.cacheQuery(
`analytics:project:${projectId}`,
async () => {
const [tasks, timeEntries, expenses] = await Promise.all([
prisma.task.groupBy({
by: ['status'],
where: { projectId },
_count: { id: true }
```

}),
prisma.timeEntry.aggregate({
where: { task: { projectId } },
_sum: { durationMinutes: true },
_count: { id: true }
}),
prisma.expense.aggregate({
where: { projectId },
_sum: { amount: true }
})
]);

```
    return { tasks, timeEntries, expenses };
  },
  600 // 10 minutes TTL
);
```

}

// Invalidate related caches on data changes
static async invalidateUserCaches(userId: string): Promise<void> {
await cache.invalidatePattern(`dashboard:user:${userId}`);
await cache.invalidatePattern(`user:${userId}:*`);
}

static async invalidateProjectCaches(projectId: string): Promise<void> {
await cache.invalidatePattern(`analytics:project:${projectId}`);
await cache.invalidatePattern(`project:${projectId}:*`);
}
}

```
DATABASE OPTIMIZATION:
```

-- Performance indexes
CREATE INDEX CONCURRENTLY idx_projects_client_status ON projects(client_id, status);
CREATE INDEX CONCURRENTLY idx_tasks_project_assignee ON tasks(project_id, assignee_id);

CREATE INDEX CONCURRENTLY idx_time_entries_user_date ON time_entries(user_id, created_at DESC);
CREATE INDEX CONCURRENTLY idx_invoices_client_status ON invoices(client_id, status);
CREATE INDEX CONCURRENTLY idx_documents_project_name ON documents(project_id, name);

-- Composite indexes for complex queries
CREATE INDEX CONCURRENTLY idx_tasks_status_priority_due ON tasks(status, priority, due_date);
CREATE INDEX CONCURRENTLY idx_time_entries_billable_approved ON time_entries(billable, approved, created_at);

-- Partial indexes for common filters
CREATE INDEX CONCURRENTLY idx_projects_active ON projects(created_at) WHERE status = 'ACTIVE';
CREATE INDEX CONCURRENTLY idx_invoices_unpaid ON invoices(due_date) WHERE status IN ('SENT', 'OVERDUE');

```
```

// lib/database/query-optimization.ts
export class OptimizedQueries {
// Efficient project list with pagination
static async getProjectsPaginated(
userId: string,
page: number = 1,
limit: number = 20,
filters: ProjectFilters = {}
) {
const offset = (page - 1) * limit;

```
const whereClause = {
  ...filters,
  teamMembers: { some: { userId } }
};

const [projects, total] = await Promise.all([
  prisma.project.findMany({
    where: whereClause,
    include: {
      client: { select: { name: true, id: true } },
      _count: {
```

```
        select: {
          tasks: true,
          teamMembers: true
        }
      }
    },
    orderBy: { updatedAt: 'desc' },
    skip: offset,
    take: limit
  }),
  prisma.project.count({ where: whereClause })
]);

return {
  projects,
  pagination: {
    page,
    limit,
    total,
    totalPages: Math.ceil(total / limit)
  }
};
```

}

// Prevent N+1 queries in task lists
static async getTasksWithRelations(projectId: string) {
return prisma.task.findMany({
where: { projectId },
include: {
assignee: { select: { firstName: true, lastName: true, email: true } },
project: { select: { name: true, client: { select: { name: true } } } },
dependencies: {
include: {
dependsOnTask: { select: { title: true, status: true } }
}
},
_count: { select: { timeEntries: true } }

```
},
orderBy: [
{ priority: 'desc' },
{ dueDate: 'asc' }
]
});
}
}
```

```
FRONTEND OPTIMIZATION:
```

```
// lib/performance/lazy-loading.ts
import { lazy, Suspense } from 'react';
import { Spinner } from '@/components/ui/Spinner';

// Lazy load heavy components
export const ProjectGanttChart = lazy(() => import('@/components/project/GanttChart'));
export const FinancialDashboard = lazy(() => import('@/components/financial/Dashboard'));
export const AnalyticsReports = lazy(() => import('@/components/analytics/Reports'));

// HOC for lazy loading with error boundaries
export function withLazyLoading<T extends object>(
Component: React.ComponentType<T>,
fallback: React.ComponentType = Spinner
) {
return function LazyComponent(props: T) {
return (
<Suspense fallback={}>
<Component {...props} />
</Suspense>
);
};
}

// Image optimization hook
export function useOptimizedImage(src: string, options: ImageOptions = {}) {
```

const {
width = 800,
height = 600,
quality = 85,
format = 'webp'
} = options;

const optimizedSrc = useMemo(() => {
if (src.startsWith('http')) return src; // External images

```
const params = new URLSearchParams({
  w: width.toString(),
  h: height.toString(),
  q: quality.toString(),
  f: format
});

return `/api/images/optimize?src=${encodeURIComponent(src)}&${params}`;
```

}, [src, width, height, quality, format]);

return optimizedSrc;
}

```
CDN CONFIGURATION:
```

// lib/cdn/configuration.ts
export const cdnConfig = {
domains: ['cdn.yourdomain.com'],
zones: {
static: {
origin: process.env.S3_BUCKET_URL,
cacheTtl: 31536000, // 1 year for static assets
compress: true,
optimizeImages: true
},
api: {

```
origin: process.env.API_ORIGIN,
cacheTtl: 300, // 5 minutes for API responses
cacheKey: ['url', 'headers.authorization'],
bypassOnCookie: ['session', 'auth-token']
}
}
};

// Middleware for CDN cache headers
export function setCacheHeaders(req: Request, res: Response, next: NextFunction) {
if (req.path.startsWith('/api/')) {
// API endpoints - short cache
res.set('Cache-Control', 'public, max-age=300, s-maxage=300');
} else if (req.path.includes('/static/')) {
// Static assets - long cache
res.set('Cache-Control', 'public, max-age=31536000, immutable');
} else {
// HTML pages - no cache
res.set('Cache-Control', 'no-cache, no-store, must-revalidate');
}

next();
}
```

```
PERFORMANCE MONITORING:
```

```
// lib/monitoring/performance.ts
export class PerformanceMonitor {
static async trackApiResponse(
endpoint: string,
method: string,
duration: number,
statusCode: number
) {
await prisma.performanceMetric.create({
data: {
```

endpoint,
method,
duration,
statusCode,
timestamp: new Date()
}
});

```
// Alert if response time is too slow
if (duration > 2000) { // 2 seconds
  await this.sendSlowResponseAlert(endpoint, duration);
}
```

}

static async getPerformanceReport(timeRange: string = '24h') {
const since = new Date(Date.now() - this.parseTimeRange(timeRange));

```
return prisma.performanceMetric.aggregate({
  where: { timestamp: { gte: since } },
  _avg: { duration: true },
  _max: { duration: true },
  _count: { id: true }
});
```

}

private static parseTimeRange(range: string): number {
const value = parseInt(range);
const unit = range.slice(-1);

```
switch (unit) {
  case 'h': return value * 60 * 60 * 1000;
  case 'd': return value * 24 * 60 * 60 * 1000;
  case 'w': return value * 7 * 24 * 60 * 60 * 1000;
  default: return 24 * 60 * 60 * 1000;
}
```

}
}

```
ACCEPTANCE CRITERIA:
- Average page load time under 2 seconds
- API response times under 500ms for 95th percentile
- Redis cache hit ratio above 80%
- Database query count optimized (no N+1 queries)
- CDN reduces asset load times by 50%+
- Performance monitoring alerts on slow responses
```

🚀 Execute This Plan for Midnight Launch

Use these prompts in sequence, starting with **Tier 0** which are your critical blockers. Each prompt is designed to be handed directly to your AI coding agent (Claude/Copilot) and will produce working, production-ready code.

Priority execution order:

1. **Database & Schema** (Prompt 1) - Foundation for everything
2. **Real Data Services** (Prompt 2) - Fixes broken portal flows
3. **Admin Dashboard** (Prompt 3) - Makes admin functionality work
4. **Auth & RBAC** (Prompt 4) - Security essentials
5. Continue with remaining prompts based on business priority

Each prompt includes acceptance criteria so you can verify completion before moving to the next item. Good luck with your midnight launch! 🎯

⁂

1. IT-Services-Agency-Platform-MVP-Complete-Develop.pdf