CONTRACTOR OS

Technical Whitepaper v1.0

AI-Powered Business Management
for Security Camera Installation Contractors

January 2026
Classification: Technical Documentation
Status: Production Ready

SYSTEM AT A GLANCE

Backend Lines of Code	**15,088**
Frontend Lines of Code	**11,596**
Total API Endpoints	**79**
Database Tables	**23**
Test Coverage	**47 Tests**
AI Intents Supported	**30+**
Development Sprints	**52**

TABLE OF CONTENTS

1. Executive Summary

Contractor OS is a comprehensive, AI-powered business management system purpose-built for security camera installation contractors. The platform unifies client management, job tracking, quoting, invoicing, expense tracking, time management, and intelligent AI assistance into a single, secure, multi-tenant application.

The system addresses a critical gap in the market: security camera installation contractors—a segment of over 68,000 businesses in the US alone—currently rely on fragmented, generic tools that were never designed for trade-specific workflows. Contractor OS replaces that patchwork with a unified platform that understands the language, workflows, and economics of camera installation businesses.

1.1 Key Differentiators

Neural Core AI: A natural language interface for business operations with intelligent intent detection, safety-first confirmation flows, and retrieval-augmented generation (RAG) over business data. Contractors interact with their data conversationally—no forms, no navigation, just language.

Multi-Tenant Architecture: Complete data isolation between companies enforced at every query layer (170+ company_id filters), with role-based access control across admin, manager, and employee tiers.

Safety-First AI: All write operations require explicit user confirmation through a three-layer security model. No silent database modifications. Every AI-generated artifact is flagged and auditable.

Offline-Capable AI: Local Ollama integration eliminates dependency on external AI services. Business data never leaves the deployment environment, ensuring complete data sovereignty.

Professional Document Generation: Automated PDF generation for quotes, invoices, dashboard reports, and expense reports—branded with company details and formatted for client-facing use.

1.2 Technology Stack

Layer	Technology	Purpose
Backend Framework	FastAPI (Python 3.11+)	High-performance async API server
Database	PostgreSQL / SQLite	Relational data with full ACID compliance
ORM	SQLAlchemy 2.0	Type-safe database operations
Frontend Framework	Next.js 14 (React 18)	Server-side rendering, App Router
Styling	Tailwind CSS	Utility-first responsive design
AI Runtime	Ollama (Local)	Privacy-preserving local AI inference
AI Model	Gemma 3 / Llama 3	Efficient language models for chat

Vector Database	ChromaDB	RAG embeddings and semantic search
PDF Generation	ReportLab	Professional document creation
Authentication	JWT + bcrypt	Stateless secure authentication

1.3 Module Overview

Module	Lines of Code	Endpoints	Description
Clients	~800	7	Customer management with timeline and file uploads
Jobs	~1,200	8	Work order tracking with notes, materials, and status
Quotes	~1,500	8	Price proposals with line items, PDF, and email
Invoices	~1,800	8	Billing with payment tracking, PDF, and conversion
Expenses	~600	6	Business expense tracking with categories and reports
Time Tracking	~400	4	Clock in/out for field workers linked to jobs
Dashboard	~500	4	Business metrics, financial summary, and PDF reports
Neural Core	~9,460	6	AI assistant subsystem with RAG and tool use
Users	~300	3	User management within companies
Companies	~200	2	Multi-tenancy and company settings
Auth	~400	3	Registration, login, and JWT authentication

2. Problem Statement & Market Opportunity

2.1 Industry Challenges

Security camera installation contractors operate in a uniquely demanding operational environment. Unlike general contractors, they manage a blend of technical specification (camera models, NVR configurations, network topology), physical installation logistics, and ongoing client relationships that often include service contracts. Generic business software fails to address several critical pain points:

Fragmented Workflows: Contractors juggle separate tools for scheduling, quoting, invoicing, and expense tracking. A single job may touch four or more disconnected systems, creating data silos and reconciliation overhead.

Manual Quote Generation: Creating accurate quotes requires technical knowledge of camera specifications, labor rates, and material costs. This process is time-consuming, error-prone, and inconsistent across team members.

Cash Flow Opacity: Without unified financial tracking, contractors struggle to see which invoices are paid, pending, or overdue. This directly impacts cash flow planning and business health.

Time Tracking Complexity: Field workers need simple clock-in/out functionality that links directly to specific jobs for accurate labor costing and client billing.

Reporting Gaps: No unified view of business health, job profitability, quote conversion rates, or client lifetime value. Decisions are made on gut feeling rather than data.

Technology Barriers: Existing enterprise solutions require significant technical expertise, onboarding time, and subscription costs that are prohibitive for solo contractors and small teams.

2.2 Target Market

Segment	Estimated Size (US)	Primary Pain Point
Solo Contractors	~50,000	Need all-in-one solution, price sensitive
Small Teams (2–10)	~15,000	Coordination, job assignment, time tracking
Growing Companies (10–50)	~3,000	Scalability, reporting, multi-location management

The total addressable market represents over 68,000 businesses in the US alone. The initial go-to-market strategy targets solo contractors and small teams (2–10 employees), where the value proposition of an AI-powered all-in-one platform is most compelling and the sales cycle is shortest.

2.3 Solution Approach

Contractor OS addresses these challenges through four strategic pillars:

Unified Platform: A single system for all business operations—eliminating the need to switch between CRM, accounting, scheduling, and communication tools.

AI-Powered Simplicity: Natural language interface via Neural Core reduces the learning curve to near zero. Contractors can ask “Create a quote for ACME Corp” instead of navigating complex form hierarchies.

Industry-Specific Design: Quote templates, job phases, material tracking, and terminology designed specifically for security camera installers. The system speaks the contractor’s language.

Data Sovereignty: Self-hosted deployment with local AI inference ensures complete data privacy. No business data leaves the contractor’s environment.

2.4 Competitive Landscape

Solution	Strengths	Weaknesses
Generic CRMs (HubSpot, etc.)	Feature-rich, ecosystem	Not industry-specific, expensive, steep learning curve
Accounting Software (QuickBooks)	Strong financial focus	No job management, no AI, limited field ops
Field Service Apps (Jobber, ServiceTitan)	Industry-aware workflows	No AI, subscription-heavy, vendor lock-in
Contractor OS	AI-powered, industry-specific, self-hostable, data sovereign	Newer platform, smaller ecosystem

3. System Architecture

3.1 Architectural Overview

Contractor OS follows a modern three-tier architecture with strict separation of concerns. The system is designed around the principle of progressive enhancement: core CRUD operations function independently of AI, while Neural Core adds intelligent capabilities on top. This ensures full reliability even when AI components are unavailable or degraded.

Tier	Components	Responsibility
Presentation	Next.js 14, React 18, Tailwind CSS	UI rendering, routing, client-side state, responsive layout
Application	FastAPI, Neural Core, SQLAlchemy	Business logic, API endpoints, AI processing, auth
Data	PostgreSQL, ChromaDB, File Storage	Persistent storage, vector embeddings, uploaded files

Communication between tiers follows a strict unidirectional flow: the Presentation tier communicates exclusively via HTTPS REST API calls to the Application tier, which in turn interacts with the Data tier through SQLAlchemy ORM (no raw SQL). This enforces a clean boundary that simplifies testing, deployment, and security auditing.

3.2 Request Flow

Standard API Request

User Action → Next.js Page → API Client → FastAPI Router → Service Layer → SQLAlchemy → Database

Every standard request follows a seven-step lifecycle: (1) User triggers an action in the UI, (2) the frontend's centralized API client constructs the HTTP request with JWT bearer token, (3) FastAPI's router validates the JWT and extracts the current user, (4) Pydantic schemas validate the request body, (5) the service layer executes business logic with company_id scoping, (6) SQLAlchemy commits to the database, and (7) the response propagates back to the UI which updates local state.

AI-Powered Request

Chat Input → Neural Core Router → Intent Translator → [Confirmation Gate] → Intent Executor → Service Layer → Database

AI requests follow an extended pipeline that adds classification, parameter extraction, and mandatory confirmation for write operations. The key difference: natural language is never directly translated to database operations. Instead, it is first converted to a structured intent,

validated, and only executed after explicit user confirmation. This is the system's foundational safety invariant.

3.3 Directory Structure

Backend Structure

The backend follows a modular architecture where each business domain (clients, jobs, quotes, etc.) is encapsulated in its own module with four standard files: models.py (SQLAlchemy ORM), schemas.py (Pydantic validation), service.py (business logic), and router.py (API endpoints). Shared utilities live in the core/ directory.

```
backend/src/
├── core/              # Shared utilities
│   ├── config.py      # Environment configuration
│   ├── database.py    # SQLAlchemy engine and session
│   ├── security.py    # JWT creation, password hashing
│   └── pdf_utils.py   # ReportLab PDF generation (~450 lines)
├── modules/
│   ├── clients/       # models, schemas, service, router
│   ├── jobs/          # models, schemas, service, router
│   ├── quotes/        # includes PDF, email, convert-to-invoice
│   ├── invoices/      # includes mark_as_paid, PDF
│   ├── expenses/      # models, schemas, service, router
│   ├── time_tracking/ # clock_in, clock_out
│   ├── users/         # user management
│   ├── companies/     # multi-tenancy
│   └── neural_core/   # AI subsystem (~9,460 lines)
│       ├── intent_translator.py  # NL → structured intent
│       ├── intent_executor.py    # Intent → DB action
│       ├── llm/ollama_service.py # Ollama integration
│       ├── routing/patterns.py   # Regex intent detection
│       ├── tools/executor.py     # Tool execution (~862 lines)
│       ├── rag/rag_complete.py   # ChromaDB RAG (~551 lines)
│       └── agent/agent_loop.py   # Multi-step execution
├── dashboard/             # metrics and reports
├── alembic/               # database migrations
└── main.py                # FastAPI application entry
```

Frontend Structure

The frontend uses Next.js 14's App Router with route groups to separate authenticated and public flows. A centralized API client (lib/api.ts) with 40+ methods handles all backend communication.

```
frontend/
├── app/
│   ├── (app)/          # Authenticated routes (sidebar layout)
│   │   ├── dashboard/  # Metrics cards, financial summary
│   │   ├── clients/    # List, detail, create, edit
│   │   ├── jobs/       # Status filters, notes, materials
```

```
│   │   ├── quotes/      # Line item editor, PDF, email
│   │   ├── invoices/    # Payment tracking, mark paid
│   │   ├── expenses/    # Category filter, PDF export
│   │   ├── chat/        # Neural Core interface
│   │   └── settings/    # User and company config
│   ├── (auth)/          # Public routes (login, register)
│   └── layout.tsx       # Root layout with providers
├── components/ui/       # Shared UI components
├── lib/
│   ├── api.ts           # Centralized API client (500+ lines)
│   ├── auth.ts          # Token management, logout
│   └── utils.ts         # Helpers (formatting, conversion)
└── tailwind.config.js   # Dark theme configuration
```

4. Database Schema

The database consists of 23 tables organized around a multi-tenant architecture. Every business entity includes a company_id foreign key that enforces data isolation at the query level. The schema uses auto-increment integer primary keys for business entities and UUID string primary keys for Neural Core tables.

4.1 Entity Relationships

The core entity graph follows a hierarchical structure rooted at the Company level:

```
Company (1) ──────< (N) Users
Company (1) ──────< (N) Clients
Client  (1) ──────< (N) Jobs
Client  (1) ──────< (N) Quotes
Client  (1) ──────< (N) Invoices
Job     (1) ──────< (N) TimeEntries
Job     (1) ──────< (N) JobNotes
Job     (1) ──────< (N) JobMaterials
Quote   (1) ──────< (N) QuoteItems
Invoice (1) ──────< (N) InvoiceItems
Quote   (1) ──────< (0..1) Invoice (conversion)
```

4.2 Core Entities

4.2.1 Companies & Users

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
name	VARCHAR(255)	NOT NULL	Company name
email	VARCHAR(255)		Contact email
phone	VARCHAR(50)		Contact phone
address	TEXT		Full address
tax_id	VARCHAR(50)		Tax identification
tax_rate	DECIMAL(5,2)	DEFAULT 0	Default tax rate
created_at	TIMESTAMP	DEFAULT NOW	Creation time
updated_at	TIMESTAMP		Last update

users — Each user belongs to exactly one company (multi-tenant isolation via company_id FK).

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
company_id	INTEGER	FK → companies.id	Tenant isolation
email	VARCHAR(255)	UNIQUE, NOT NULL	Login email

hashed_password	VARCHAR(255)	NOT NULL	bcrypt hash
full_name	VARCHAR(255)		Display name
role	ENUM	DEFAULT 'employee'	admin / manager / employee
is_active	BOOLEAN	DEFAULT true	Account status

4.2.2 Client Management

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
company_id	INTEGER	FK → companies.id	Tenant isolation
name	VARCHAR(255)	NOT NULL	Client name
email	VARCHAR(255)		Contact email
phone	VARCHAR(50)		Contact phone
address	TEXT		Service address
notes	TEXT		Internal notes

Clients also support file attachments (client_files) and activity timelines (client_timeline_events) that automatically aggregate events from quotes, invoices, jobs, and manual notes.

4.2.3 Job Management

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
company_id	INTEGER	FK → companies.id	Tenant isolation
client_id	INTEGER	FK → clients.id, CASCADE	Associated client
title	VARCHAR(255)	NOT NULL	Job title
description	TEXT		Detailed description
status	ENUM	DEFAULT 'pending'	pending / in_progress / completed / cancelled
scheduled_date	DATE		Scheduled start
completed_date	DATE		Actual completion
address	TEXT		Job site address

Each job supports notes (job_notes with user attribution), material tracking (job_materials with quantity and unit cost), and file attachments (job_files).

4.2.4 Financial Entities

quotes — Price proposals with line items, tax calculation, and lifecycle status tracking.

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
company_id	INTEGER	FK → companies.id	Tenant isolation
client_id	INTEGER	FK → clients.id, CASCADE	Target client
job_id	INTEGER	FK → jobs.id, SET NULL	Optional linked job
quote_number	VARCHAR(50)	Auto-generated	QUO-XXXX format
status	ENUM	DEFAULT 'draft'	draft / sent / accepted / rejected / expired
subtotal	DECIMAL(12,2)	DEFAULT 0	Sum of line items
tax_rate	DECIMAL(5,2)	DEFAULT 0	Tax percentage
total	DECIMAL(12,2)	DEFAULT 0	Grand total
created_by_ai	BOOLEAN	DEFAULT false	AI safety flag
reviewed_by_user_id	INTEGER	FK → users.id	Human review audit trail

invoices — Billing documents with payment tracking. Support partial payments via amount_paid / amount_due computation.

Key Fields	Type	Description
invoice_number	VARCHAR(50)	Auto-generated INV-XXXX format
quote_id	INTEGER	FK to source quote (for quote-to-invoice conversion)
status	ENUM	draft / sent / partial / paid / overdue / cancelled
amount_paid	DECIMAL(12,2)	Received payments
amount_due	DECIMAL(12,2)	Computed: total - amount_paid
due_date	DATE	Payment deadline
paid_date	DATE	Full payment date
created_by_ai	BOOLEAN	AI safety flag

expenses — Business expense tracking with categories (materials, gas, tools, labor, other) and optional job linkage for per-job cost analysis.

4.2.5 Time Tracking

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO	Unique identifier
user_id	INTEGER	FK → users.id, CASCADE	Worker

job_id	INTEGER	FK → jobs.id, SET NULL	Optional linked job
clock_in	TIMESTAMP	NOT NULL	Start time
clock_out	TIMESTAMP		End time (null = active)
hours_worked	DECIMAL(5,2)	Computed	Duration in hours

4.2.6 Neural Core Tables

The Neural Core subsystem uses UUID string primary keys (distinct from the integer PKs used by business entities) across three tables:

neural_core_conversations: Tracks conversation sessions with user_id and company_id scoping. Each conversation has a title, creation timestamp, and last activity timestamp.

neural_core_messages: Individual messages within conversations, with role (user/assistant/system), content, and optional JSON metadata for intent data.

neural_core_tool_executions: Complete audit log of every tool execution, including tool name, parameters, result, success status, and error details.

4.3 Data Integrity

Constraint Type	Implementation	Count
Primary Keys	Auto-increment integers (business), UUID strings (neural_core)	23
Foreign Keys	With ondelete CASCADE or SET NULL	41
Unique Constraints	quote_number+client_id, invoice_number+client_id, user email	4
Check Constraints	amount > 0, valid status enums	8
Indexes	company_id on all tenant tables + common query patterns	23+

4.4 Index Strategy

Multi-tenant isolation is the primary indexing concern. Every tenant-scoped table has a company_id index. Additional composite indexes target common query patterns:

```
-- Multi-tenant isolation (critical for query performance)
CREATE INDEX idx_clients_company ON clients(company_id);
CREATE INDEX idx_jobs_company ON jobs(company_id);
CREATE INDEX idx_quotes_company ON quotes(company_id);
CREATE INDEX idx_invoices_company ON invoices(company_id);
CREATE INDEX idx_expenses_company ON expenses(company_id);

-- Common query pattern indexes
CREATE INDEX idx_jobs_status ON jobs(company_id, status);
CREATE INDEX idx_invoices_status ON invoices(company_id, status);
CREATE INDEX idx_time_entries_user ON time_entries(user_id, clock_in);
```

5. API Reference

The system exposes 79 RESTful endpoints across 11 modules. All endpoints except /auth/* require JWT authentication via Bearer token.

5.1 Authentication

JWT tokens are issued upon login and included in the Authorization header of every subsequent request. The token payload includes user_id, company_id, and role—enabling tenant isolation and RBAC without additional database lookups per request.

```
// JWT Payload Structure
{
  "sub": "user@example.com",
  "user_id": 1,
  "company_id": 1,
  "role": "admin",
  "exp": 1735689600
}
```

Method	Endpoint	Description	Auth
POST	/api/v1/auth/register	Create new user and company	No
POST	/api/v1/auth/login	Get JWT token	No
GET	/api/v1/auth/me	Get current user info	Yes

5.2 Resource Endpoints

5.2.1 Clients (7 endpoints)

Method	Endpoint	Description
GET	/api/v1/clients	List all clients (company-scoped)
POST	/api/v1/clients	Create client
GET	/api/v1/clients/{id}	Get client details
PUT	/api/v1/clients/{id}	Update client
DELETE	/api/v1/clients/{id}	Delete client
GET	/api/v1/clients/{id}/stats	Get client statistics (jobs, revenue, quotes)
GET	/api/v1/clients/{id}/timeline	Get activity timeline events

5.2.2 Jobs (8 endpoints)

Method	Endpoint	Description
GET	/api/v1/jobs	List jobs (filterable by status, client_id)
POST	/api/v1/jobs	Create job

GET	/api/v1/jobs/{id}	Get job details with notes and materials
PUT	/api/v1/jobs/{id}	Update job
PATCH	/api/v1/jobs/{id}/status	Update status only
DELETE	/api/v1/jobs/{id}	Delete job
GET	/api/v1/jobs/{id}/notes	Get job notes
POST	/api/v1/jobs/{id}/notes	Add job note

5.2.3 Quotes (8 endpoints)

Method	Endpoint	Description
GET	/api/v1/quotes	List all quotes
POST	/api/v1/quotes	Create quote with line items
GET	/api/v1/quotes/{id}	Get quote details
PUT	/api/v1/quotes/{id}	Update quote and items
DELETE	/api/v1/quotes/{id}	Delete quote
GET	/api/v1/quotes/{id}/pdf	Download as PDF
POST	/api/v1/quotes/{id}/send	Send to client email
POST	/api/v1/quotes/{id}/convert	Convert to invoice

5.2.4 Invoices (8 endpoints)

Method	Endpoint	Description
GET	/api/v1/invoices	List all invoices
POST	/api/v1/invoices	Create invoice
GET	/api/v1/invoices/{id}	Get invoice details
PUT	/api/v1/invoices/{id}	Update invoice
DELETE	/api/v1/invoices/{id}	Delete invoice
GET	/api/v1/invoices/{id}/pdf	Download as PDF
POST	/api/v1/invoices/{id}/send	Send to client
POST	/api/v1/invoices/{id}/mark-paid	Record payment (amount, date)

5.2.5 Expenses (6 endpoints)

Method	Endpoint	Description
GET	/api/v1/expenses	List all expenses
POST	/api/v1/expenses	Create expense
GET	/api/v1/expenses/{id}	Get expense details
PUT	/api/v1/expenses/{id}	Update expense
DELETE	/api/v1/expenses/{id}	Delete expense

GET	/api/v1/expenses/report/pdf	Download expense report PDF (filterable by days, category)

5.2.6 Time Tracking (4 endpoints)

Method	Endpoint	Description
GET	/api/v1/time-tracking	List time entries
POST	/api/v1/time-tracking/clock-in	Start tracking (with optional job_id)
POST	/api/v1/time-tracking/clock-out	Stop tracking
GET	/api/v1/time-tracking/active	Get active entry

5.2.7 Dashboard (4 endpoints)

Method	Endpoint	Description
GET	/api/v1/dashboard	Complete dashboard (financial, jobs, hours, sales)
GET	/api/v1/dashboard/financial	Financial summary only
GET	/api/v1/dashboard/jobs	Jobs summary only
GET	/api/v1/dashboard/pdf	Download dashboard report PDF

5.3 Neural Core API (6 endpoints)

Method	Endpoint	Description
POST	/api/v1/neural-core/chat	Send message to AI (returns intent or conversation)
POST	/api/v1/neural-core/execute	Execute a structured, confirmed intent
POST	/api/v1/neural-core/preview	Preview intent without executing
GET	/api/v1/neural-core/conversations	List conversation history
GET	/api/v1/neural-core/health	System health check (Ollama + RAG status)
GET	/api/v1/neural-core/capabilities	List supported intents and capabilities

6. Neural Core — AI Subsystem

Neural Core is the AI-powered assistant that enables natural language interaction with every aspect of Contractor OS. It is the system's primary differentiator and is designed around a single inviolable principle:

> **Golden Rule**
> Nothing that writes to the database can come directly from natural language. Natural language = UX. Structured intents = Authority.

6.1 Component Architecture

Component	Lines	Purpose
Intent Translator	~315	Converts natural language to structured intents with parameter extraction
Intent Router	~230	Pattern matching for intent detection (no LLM required)
Intent Executor	~550	Executes validated intents deterministically against service layer
Tool Executor	~862	Legacy tool execution engine (read-only operations)
Agent Loop	~494	Multi-step query resolution for complex questions
Ollama Service	~225	LLM integration for open-ended conversations
RAG Retriever	~551	Semantic search over business data via ChromaDB
Service Layer	~463	Orchestrates all components, manages conversation state

Total Neural Core codebase: approximately 9,460 lines across all components. The architecture follows a classify-first philosophy: every incoming message is first classified using lightweight pattern matching before any LLM call is made. This ensures fast responses for common operations and reserves expensive LLM inference for genuinely open-ended queries.

6.2 Intent Processing Pipeline

The complete message processing pipeline follows five stages:

Stage 1 — Short-Circuit Check: Greetings, help requests, and thanks are matched against a pre-defined dictionary and return instant responses (<100ms) without touching the LLM or database.

Stage 2 — Intent Translation: The IntentRouter applies regex pattern matching to detect the user's intent. Parameters are extracted via specialized regex patterns (email addresses, numeric IDs, monetary amounts, names). Confidence scores are assigned.

Stage 3 — Confirmation Gate: If the detected intent is a mutation (is_mutating=true), execution is blocked. The system stores the pending intent and returns a confirmation prompt to the user. Only an explicit "Yes" advances to Stage 4.

Stage 4 — Intent Execution: The IntentExecutor retrieves the pending intent, calls the appropriate service layer method (e.g., ClientService.create_client), and returns the result with a human-readable effect summary.

Stage 5 — Fallback to LLM: If no intent is detected with sufficient confidence, the message is routed to the Ollama service for open-ended conversation, optionally enriched with RAG context from the business data vector store.

6.3 Supported Intents

6.3.1 Query Intents (No Confirmation Required)

Intent	Description	Example Phrases
GET_CLIENTS	List all clients	"show my clients", "list clients"
GET_CLIENT	Get specific client	"show client #5", "client details for ACME"
GET_JOBS	List jobs with filters	"show jobs", "what jobs are pending?"
GET_JOB	Get specific job	"show job #3", "job details"
GET_QUOTES	List quotes	"show my quotes"
GET_QUOTE	Get specific quote	"show quote #10"
GET_INVOICES	List invoices	"show invoices", "unpaid invoices"
GET_INVOICE	Get specific invoice	"show invoice #7"
GET_EXPENSES	List expenses	"show my expenses"
GET_DASHBOARD	Dashboard metrics	"show dashboard", "how's business?"
GET_TIME_ENTRIES	Time tracking history	"show my hours"
ANALYZE_REVENUE	Revenue analysis	"what's my profit?", "revenue this month"
SEARCH	Cross-entity search	"search for ACME"

6.3.2 Mutation Intents (Confirmation Required)

Intent	Description	Required Params
CREATE_CLIENT	Create new client	name, email
UPDATE_CLIENT	Update client info	client_id
DELETE_CLIENT	Delete client	client_id
CREATE_JOB	Create new job	title, client_id
UPDATE_JOB	Update job	job_id
UPDATE_JOB_STATUS	Change job status	job_id, status
DELETE_JOB	Delete job	job_id
CREATE_QUOTE	Generate quote	client_id
UPDATE_QUOTE	Update quote	quote_id
DELETE_QUOTE	Delete quote	quote_id

CREATE_INVOICE	Generate invoice	client_id
UPDATE_INVOICE	Update invoice	invoice_id
MARK_INVOICE_PAID	Record payment	invoice_id
DELETE_INVOICE	Delete invoice	invoice_id
CREATE_EXPENSE	Log expense	description, amount, category
CLOCK_IN	Start time tracking	(none)
CLOCK_OUT	Stop time tracking	(none)

6.4 Safety Mechanisms

6.4.1 Three-Layer Security Model

Layer 1 — Intent Translator: Classifies every intent as READ or WRITE. Write intents are marked is_mutating=true and confirmed=false by default. No mutation can bypass this classification.

Layer 2 — Intent Executor: Checks the confirmed flag before executing any mutation. If confirmed=false, execution is rejected and a preview is returned instead. This is the primary safety gate.

Layer 3 — Tool Executor (Defense in Depth): Maintains a MUTATING_TOOLS blocklist that prevents direct tool calls for any write operation. Even if Layers 1 and 2 were somehow bypassed, the tool executor would reject the call. This is the last line of defense.

6.4.2 Financial Safety Controls

Control	Limit	Purpose
AI Quote Maximum	$50,000	Prevent runaway quotes from AI generation
AI Invoice Maximum	$50,000	Prevent billing errors from AI generation
Human Review Flag	created_by_ai = true	Track all AI-generated financial content
Review Tracking	reviewed_by_user_id	Audit trail for human oversight
Audit Log	neural_core_tool_executions	Complete record of every AI operation

6.5 Performance Optimizations

6.5.1 Lazy Initialization

Heavy components (Ollama service, SentenceTransformers for RAG) are loaded only on first use. This means server startup is fast and memory is conserved when AI features are not actively being used.

6.5.2 Response Time by Input Type

Input Type	Response Time	Method

Greetings (hello, hi, hola)	<100ms	Pre-defined dictionary lookup
Thanks / Goodbye	<100ms	Pre-defined dictionary lookup
Help / Capabilities	<100ms	Pre-defined response template
Data queries (show clients)	<500ms	Direct database query
Mutations with confirmation	<100ms	Template + state storage
General conversation	2–5s	Ollama LLM inference (last resort)

6.6 RAG (Retrieval Augmented Generation)

The RAG subsystem uses ChromaDB with BAAI/bge-m3 embeddings to provide semantic search over business data. Seven collections are maintained: jobs, quotes, invoices, clients, expenses, conversations, and documentation.

Data is automatically indexed on create and update operations. If indexing fails (e.g., ChromaDB is temporarily unavailable), the operation logs a warning but does not block the business transaction. This ensures RAG is additive, never disruptive.

7. Security Model

7.1 Authentication

7.1.1 JWT Implementation

Attribute	Value
Algorithm	HS256 (HMAC-SHA256)
Token Expiration	24 hours (configurable)
Signing Key	SECRET_KEY environment variable
Minimum Key Length	32 characters

7.1.2 Password Security

Aspect	Implementation
Hashing Algorithm	bcrypt with automatic salt generation
Minimum Length	8 characters
Storage	Only hash stored, never plaintext

7.2 Authorization

7.2.1 Multi-Tenancy Enforcement

Every database query includes a company_id filter. The user's company_id is extracted from the JWT token, never from the request body. This means a user cannot access another company's data even by crafting a malicious request. The system enforces 170+ company_id filters across all modules.

7.2.2 Role-Based Access Control

Role	Description	Permissions
admin	Company owner	Full access: all CRUD, user management, settings, AI
manager	Team lead	Standard access: all CRUD on business entities
employee	Field worker	Limited: view data, clock in/out, add notes

7.3 Input Validation

All input is validated through Pydantic schemas at the API boundary. Field-level constraints (min/max length, email format, numeric ranges) are enforced before any business logic executes. SQLAlchemy ORM parameterization prevents SQL injection—there are zero raw SQL strings in the codebase. XSS is mitigated by JSON-only responses and no use of dangerouslySetInnerHTML on the frontend.

7.4 AI-Specific Security Controls

Control	Implementation	Purpose
Confirmation Gate	confirmed=true required for mutations	Prevent accidental writes
Tool Blocking	MUTATING_TOOLS blocklist	Defense in depth
Amount Limits	$50,000 max for AI-generated content	Financial safety
AI Flag	created_by_ai=true	Track AI-generated content
Review Tracking	reviewed_by_user_id	Human oversight audit trail
Audit Log	neural_core_tool_executions	Full accountability

7.5 Known Security Issues

CRITICAL: Authentication Gap

32 endpoints (~40%) lack explicit Depends(get_current_user) authentication decorator.
Recommendation: Audit all endpoints and add authentication. Priority: Sprint 54.

HIGH: Foreign Key Constraints

41 ForeignKey constraints are missing ondelete specification (CASCADE/SET NULL).
Recommendation: Create migration to add constraints. Priority: Sprint 54.

8. Frontend Architecture

8.1 Technology Stack

Technology	Version	Purpose
Next.js	14.x	React framework with App Router, SSR
React	18.x	UI rendering library
TypeScript	5.x	Static type safety
Tailwind CSS	3.x	Utility-first styling with dark theme
Lucide React	0.263.x	Icon library

8.2 Page Overview

Page	Route	Key Features
Dashboard	/dashboard	Metrics cards, financial summary, AI insights, PDF export
Clients	/clients	Search, create modal, detail with timeline
Jobs	/jobs	Status filter tabs, create form, status updates
Quotes	/quotes	Line item editor, PDF download, email send
Invoices	/invoices	Payment tracking, mark paid, PDF generation
Expenses	/expenses	List with totals, category filter, PDF export
Chat	/chat	Neural Core interface, confirmation buttons
Settings	/settings	User profile, company settings

8.3 State Management

The application uses local state plus server refresh—no global state library (Redux, Zustand). Each page manages its own state via useState hooks, and mutations trigger a full re-fetch from the server. This trades some network efficiency for absolute consistency: the UI always reflects the database state.

Authentication state is managed via localStorage for token persistence and a centralized getToken() / logout() utility module. The API client in lib/api.ts (500+ lines, 40+ methods) handles all backend communication with consistent error handling and token injection.

8.4 Design System

Token	Value	Usage
Primary	#CC2D5A	Buttons, accents, brand identity
Background	#0a0a0a	Page backgrounds
Card	bg-zinc-900	Content containers

Border	border-zinc-800	Subtle separators
Text Primary	text-white	Headings and emphasis
Text Secondary	text-zinc-400	Labels and descriptions

The design follows a dark theme with high-contrast accents. Responsive breakpoints at 640px (mobile landscape), 768px (tablet), 1024px (desktop), and 1280px (wide desktop) ensure the application works across devices—critical for contractors who need mobile access in the field.

9. PDF Generation

Four document types are generated server-side using ReportLab, a mature Python PDF library. Each document is built from the company's profile data, the relevant business entity, and standardized layout templates.

Document	Endpoint	Content
Invoice PDF	GET /invoices/{id}/pdf	Company header, bill-to, line items, totals, notes
Quote PDF	GET /quotes/{id}/pdf	Company header, scope of work, line items, terms
Dashboard Report	GET /dashboard/pdf	Financial summary, jobs summary, hours, sales metrics
Expense Report	GET /expenses/report/pdf	Filterable by days and category, itemized expenses

PDF generation uses a BytesIO buffer pattern—the document is built entirely in memory and streamed directly in the HTTP response. No temporary files are created on disk. All documents include the company name, address, and branding in the header, and a "Generated by Contractor OS" footer.

10. Deployment & Operations

10.1 Environment Variables

Variable	Required	Default	Description
DATABASE_URL	Yes	—	PostgreSQL connection string
SECRET_KEY	Yes	—	JWT signing key (min 32 chars)
OLLAMA_BASE_URL	No	localhost:11434	Ollama API URL
OLLAMA_MODEL	No	gemma:2b	AI model name
CORS_ORIGINS	No	*	Allowed frontend origins
NEXT_PUBLIC_API_URL	Yes	—	Backend URL for frontend

10.2 System Requirements

Component	Minimum	Recommended
CPU	2 cores	4+ cores
RAM	4 GB	8+ GB (for Ollama)
Storage	10 GB	50+ GB
Python	3.11+	3.12
Node.js	18+	20 LTS
PostgreSQL	14+	16

10.3 Production Deployment

The recommended production stack is Gunicorn with Uvicorn workers behind Nginx for the backend, and Vercel (or static export) for the frontend. SSL is managed via Let's Encrypt with automatic renewal.

Backend: gunicorn src.main:app -w 4 -k uvicorn.workers.UvicornWorker -b 0.0.0.0:8000

Frontend: Vercel deploy (recommended) or npm run build && npm run export for static hosting.

Ollama: Runs as a separate service on the same host or a dedicated GPU server. Model pulled once, served continuously.

10.4 Monitoring

Tool	Purpose
Health Endpoint	/api/v1/neural-core/health — checks Ollama and RAG status
Sentry	Error tracking and exception alerting
Prometheus + Grafana	Metrics collection and dashboards
Loki	Log aggregation and search

11. Testing Strategy

11.1 Test Suite Overview

Test File	Tests	Coverage Area
test_auth.py	8	Registration, login, JWT validation
test_clients.py	12	CRUD operations, stats, timeline
test_jobs.py	10	CRUD, status changes, notes
test_invoices.py	17	CRUD, payments, PDF generation

Total: 47 automated tests. Tests use an in-memory SQLite database with per-test rollback for isolation. Fixtures provide pre-configured users, companies, and auth headers.

11.2 Coverage Goals

Area	Current	Target
Auth	~80%	90%
Clients	~70%	85%
Jobs	~60%	85%
Invoices	~75%	90%
Quotes	~50%	85%
Neural Core	~30%	70%
Overall	~55%	80%

12. Roadmap

12.1 Completed (Sprints 1–52)

Core CRUD for all entities (Clients, Jobs, Quotes, Invoices, Expenses). Multi-tenant authentication and authorization. Dashboard with business metrics. Real-time time tracking with live timer. PDF generation (invoices, quotes, dashboard, expenses). Neural Core AI with safety-first confirmation flow. RAG semantic search over business data. 47 automated tests. Responsive dark theme UI.

12.2 Short-Term (Sprints 53–60)

Sprint	Focus	Deliverables
54	Security Hardening	Fix 32 endpoints missing auth, add rate limiting
55	Database Integrity	Add missing ondelete constraints, composite indexes
56	Error Handling	Remove bare except blocks, improve error messages
57	TypeScript Cleanup	Remove 39 'any' types, add proper interfaces
58	Test Coverage	Increase to 80%+ coverage
59	Performance	Query optimization, response caching
60	Documentation	OpenAPI docs, user guide

12.3 Medium-Term (Q2 2026)

Mobile Application (React Native for field workers). Client Portal (quote approval and payment without login). Accounting Integration (QuickBooks / Xero sync). Inventory Management (equipment and materials tracking). Calendar View (visual scheduling with drag-and-drop). Email Automation (reminders for quotes and invoices).

12.4 Long-Term Vision (2026–2027)

Multi-Language support (Spanish complete, adding Portuguese and French). Offline Mode (work without internet, sync when connected). AI Estimation (generate quotes from site photos). Crew Dispatch (assign jobs to teams, optimize routes). Customer Scoring (predict customer lifetime value). Predictive Maintenance (alert customers to upgrade opportunities).

13. Appendices

Appendix A: Complete API Endpoint Summary

Module	Endpoints	Methods
Auth	3	POST, POST, GET
Clients	7	GET, POST, GET, PUT, DELETE, GET, GET
Jobs	8	GET, POST, GET, PUT, PATCH, DELETE, GET, POST
Quotes	8	GET, POST, GET, PUT, DELETE, GET, POST, POST
Invoices	8	GET, POST, GET, PUT, DELETE, GET, POST, POST
Expenses	6	GET, POST, GET, PUT, DELETE, GET
Time Tracking	4	GET, POST, POST, GET
Dashboard	4	GET, GET, GET, GET
Neural Core	6	POST, POST, POST, GET, GET, GET
Users	3	GET, PUT, DELETE
Companies	2	GET, PUT

Appendix B: Neural Core Intent Reference

Query Intents (14): GET_CLIENTS, GET_CLIENT, GET_JOBS, GET_JOB, GET_QUOTES, GET_QUOTE, GET_INVOICES, GET_INVOICE, GET_EXPENSES, GET_DASHBOARD, GET_TIME_ENTRIES, ANALYZE_REVENUE, SEARCH

Mutation Intents (17): CREATE/UPDATE/DELETE for Clients, Jobs, Quotes, Invoices, and Expenses, plus MARK_INVOICE_PAID, UPDATE_JOB_STATUS, CLOCK_IN, CLOCK_OUT

Appendix C: Database Migration History

Migration	Description
20251226_000001	Initial schema
20251226_020248	Add phone to users
20251227_xxxxxx	Add AI safety fields (created_by_ai, reviewed_by_user_id)
20251228_xxxxxx	Add expense categories
20260108_xxxxxx	Add neural core tables
20260109_xxxxxx	Add ondelete constraints
20260110_xxxxxx	Merge migration heads

Appendix D: Audit Findings Summary

Severity	Count	Example

Critical	3	32 endpoints without auth
High	7	41 ForeignKeys without ondelete
Medium	9	39 TypeScript 'any' types
Low	8	Missing docstrings

Overall Quality Score: 6.5 / 10

Appendix E: Technology Licenses

Technology	License
FastAPI	MIT
SQLAlchemy	MIT
Next.js	MIT
React	MIT
Tailwind CSS	MIT
ReportLab	BSD
Ollama	MIT
ChromaDB	Apache 2.0
Pydantic	MIT

Appendix F: Glossary

Term	Definition
Intent	Structured representation of a user's desired action derived from natural language
Mutation	Database write operation (create, update, delete) requiring confirmation
Multi-tenancy	Architecture where a single instance serves multiple isolated companies
RAG	Retrieval Augmented Generation — enhancing LLM responses with external data
JWT	JSON Web Token — stateless authentication mechanism
ORM	Object-Relational Mapping — database abstraction layer
Neural Core	The AI subsystem that powers natural language interaction in Contractor OS