Builders Stream Pro — Development Prompt Guide

Master Reference for Claude Code Development Sessions

Stack: Django 5.x, Django REST Framework, PostgreSQL, Celery + Redis, React 18 (Vite + TailwindCSS), Stripe, AWS S3 **Architecture:** Multi-tenant SaaS with Row-Level Security, modular Django apps **Source PRD:** Builders Stream Pro v3.0 — Construction & Renovation Management Platform

> **How to use this file:** Each section below is a self-contained development phase. Copy the prompt in the `### Claude Code Prompt` block and paste it directly into Claude Code. Prompts are ordered by dependency — complete them in sequence. Each prompt assumes prior phases are already built.

TABLE OF CONTENTS

#	Section	Priority	Phase
15	Service & Warranty Management	🔵 Later	4
16	Analytics & Reporting Engine	🔵 Later	4
17	Integration Ecosystem & API	🔵 Later	4
18	Mobile / PWA Experience	🔵 Later	4

PHASE 1 — FOUNDATION (Sections 1–5)

Section 1: Project Scaffolding & Core Configuration

Goal: Initialize the Django project with a scalable multi-app structure, dev tooling, Docker, environment config, and a React frontend shell.

Key Decisions:

- Django project named `builderstream` with a `config/` settings package (base/dev/prod)
- Each module becomes its own Django app inside an `apps/` directory
- React frontend as a separate `frontend/` directory, API-driven via DRF
- PostgreSQL with connection pooling, Redis for caching and Celery broker
- Docker Compose for local dev (postgres, redis, django, celery worker, celery beat, react dev server)

Claude Code Prompt

Create a production-ready Django 5.x SaaS project called "builderstream" for a construction management platform. Use this exact structure:

```
builderstream/
├── config/
│   ├── __init__.py
│   ├── settings/
│   │   ├── __init__.py
│   │   ├── base.py          # Shared settings
│   │   ├── development.py   # Local dev overrides
│   │   └── production.py    # Production settings
│   ├── urls.py              # Root URL config
│   ├── wsgi.py
│   ├── asgi.py
│   └── celery.py            # Celery app configuration
├── apps/
│   ├── __init__.py
│   ├── core/                # Shared models, mixins, utilities, permissions
│   ├── tenants/             # Organization / multi-tenant models
│   ├── accounts/            # Custom User, auth, registration, roles
│   ├── billing/             # Stripe subscriptions, plans, module activation
│   ├── projects/            # Project Command Center, lifecycle
│   ├── crm/                 # Leads, pipeline, contacts, marketing
│   ├── estimating/          # Takeoffs, cost database, proposals
│   ├── scheduling/          # Gantt, resource allocation, crews
│   ├── financials/          # Job costing, invoicing, accounting, change orders
│   ├── clients/             # Client portal, approvals, selections
│   ├── documents/           # Doc management, photos, RFIs, submittals
│   ├── field_ops/           # Daily logs, time tracking, expenses
│   ├── quality_safety/      # Inspections, safety, OSHA compliance
│   ├── payroll/             # Payroll processing, certified payroll
│   ├── service/             # Service tickets, warranty, maintenance
│   └── analytics/           # Reporting engine, dashboards, KPIs
├── frontend/                # React 18 + Vite + TailwindCSS (separate)
├── templates/               # Django templates (admin, emails)
├── static/
├── media/
├── requirements/
│   ├── base.txt
│   ├── development.txt
│   └── production.txt
├── docker-compose.yml
├── Dockerfile
```

```
├── .env.example
├── manage.py
└── README.md
```

Requirements for base.txt:
- Django>=5.0, djangorestframework, django-cors-headers, django-filter
- psycopg[binary], dj-database-url
- celery[redis], django-celery-beat, redis
- django-storages, boto3 (S3 file uploads)
- stripe, django-allauth (social auth)
- djangorestframework-simplejwt (JWT auth for API)
- django-environ (env var management)
- Pillow, django-imagekit
- django-extensions, django-debug-toolbar (dev)
- gunicorn, whitenoise (production)
- drf-spectacular (OpenAPI docs)

In config/settings/base.py:
- Use django-environ to read from .env
- Configure DATABASES for PostgreSQL with connection pooling
- Set AUTH_USER_MODEL = 'accounts.User'
- Configure DRF with JWT as default auth, pagination (page size 25), django-filter as default filter backend
- Configure Celery to use Redis broker with JSON serializer
- Configure django-storages for S3 with presigned URL support
- Set up CORS to allow the React frontend origin
- Register drf-spectacular for auto API docs at /api/docs/

In config/urls.py:
- Mount all app API endpoints under /api/v1/ namespace
- Mount DRF spectacular schema and Swagger UI at /api/docs/
- Mount Django admin at /admin/

In config/celery.py:
- Standard Celery app with autodiscover_tasks() for all apps

In apps/core/:
- Create a TimeStampedModel abstract base (created_at, updated_at)
- Create a TenantModel abstract base that extends TimeStampedModel with an 'organization' FK and a custom manager that auto-filters by organization
- Create standard permission classes: IsOrganizationMember, IsOrganizationAdmin, IsOrganizationOwner
- Create a TenantViewSetMixin that auto-injects organization filtering into querysets and serializer context

In docker-compose.yml:
- Services: db (postgres:16), redis (redis:7-alpine), web (Django), celery_worker, celery_beat, frontend (node:20)

- Mount volumes for postgres data persistence and code hot-reload
- Expose Django on 8000, React on 5173, Postgres on 5432

Create the .env.example with all required variables documented.
Create a basic README.md with setup instructions.
Initialize each app directory with __init__.py, models.py, views.py, serializers.py, urls.py, admin.py, tasks.py, signals.py, and tests/ directory.

Section 2: Multi-Tenant Architecture & Organization Model

Goal: Build the tenant isolation layer — Organization model, Prisma-style middleware for auto-filtering, PostgreSQL Row-Level Security, and tenant context management.

Key Decisions:

- Shared database, shared schema with `organization_id` FK on all tenant data
- Django middleware injects tenant context from authenticated user
- Custom model manager auto-filters all queries by organization
- RLS policies as defense-in-depth at the database level

Claude Code Prompt

Build the multi-tenant architecture for Builders Stream Pro in the apps/tenants/ Django app. This is the data isolation foundation — every other app depends on it.

MODELS (apps/tenants/models.py):

1. Organization model:
   - id (UUID primary key)
   - name (CharField, max 200)
   - slug (SlugField, unique — used in URLs)
   - owner (FK to User, related_name='owned_organizations')
   - logo (ImageField, nullable, upload to 'org_logos/')
   - address_line1, address_line2, city, state, zip_code, country
   - phone, email, website (all optional)
   - industry_type (CharField choices: RESIDENTIAL_REMODEL, CUSTOM_HOME, COMMERCIAL_GC, SPECIALTY_TRADE, ROOFING_EXTERIOR, ENTERPRISE — controls default module activation)
   - stripe_customer_id (CharField, nullable, unique)
   - stripe_subscription_id (CharField, nullable)
   - subscription_plan (CharField choices: STARTER, PROFESSIONAL, ENTERPRISE, TRIAL)
   - subscription_status (CharField choices: ACTIVE, PAST_DUE, CANCELED, TRIALING)
   - trial_ends_at (DateTimeField, nullable)
   - max_users (IntegerField, default=5)
   - is_active (BooleanField, default=True)
   - settings (JSONField, default=dict — stores org-level config like fiscal year start, timezone, date format, default markup percentage)
   - Extends TimeStampedModel from core

2. OrganizationMembership model (through table for User <-> Organization):
   - id (UUID pk)
   - user (FK to User)
   - organization (FK to Organization)
   - role (CharField choices: OWNER, ADMIN, PROJECT_MANAGER, ESTIMATOR, FIELD_WORKER, ACCOUNTANT, READ_ONLY)
   - is_active (BooleanField, default=True)
   - invited_by (FK to User, nullable)
   - invited_at (DateTimeField, nullable)
   - accepted_at (DateTimeField, nullable)
   - unique_together = [user, organization]

3. ActiveModule model (tracks which modules an org has enabled):
   - organization (FK to Organization)
   - module_key (CharField choices: PROJECT_CENTER, CRM, ESTIMATING, SCHEDULING, FINANCIALS, CLIENT_PORTAL, DOCUMENTS, FIELD_OPS, QUALITY_SAFETY, PAYROLL, SERVICE_WARRANTY, ANALYTICS)

- is_active (BooleanField, default=True)
- activated_at (DateTimeField)
- Notes: PROJECT_CENTER and ANALYTICS are always active and cannot be deactivated

TENANT MANAGER & MIXIN (apps/core/models.py — update the existing TenantModel):
- TenantManager: Custom manager with get_queryset() that checks for _tenant_org on the model's query context. If thread-local storage has current_organization, auto-filter by organization=current_org.
- TenantModel (abstract): organization FK, default manager = TenantManager. Override save() to auto-set organization from thread-local if not set.

MIDDLEWARE (apps/tenants/middleware.py):
- TenantMiddleware: On each request, if user is authenticated, read the 'X-Organization-ID' header (or fallback to user's default org). Validate user has active membership. Store organization in thread-local storage AND on request.organization. If no org found for authenticated user, return 403.
- Use Python threading.local() for thread-safe tenant context.

CONTEXT MANAGER (apps/tenants/context.py):
- set_current_organization(org) / get_current_organization() / clear_current_organization()
- tenant_context(org) context manager for use in Celery tasks and management commands

SERIALIZERS (apps/tenants/serializers.py):
- OrganizationSerializer (full CRUD)
- OrganizationMembershipSerializer (with nested user info)
- InviteMemberSerializer (email, role — sends invitation)
- ModuleActivationSerializer

VIEWS (apps/tenants/views.py — DRF ViewSets):
- OrganizationViewSet: Create org (auto-assign current user as OWNER), update, retrieve. Only OWNER/ADMIN can update.
- MembershipViewSet: List members, invite new member (sends email), update roles, deactivate members. Enforce max_users limit from subscription.
- ModuleViewSet: List active modules, activate/deactivate (validate against subscription plan limits)
- SwitchOrganizationView: APIView that lets users switch their active org context (for users belonging to multiple orgs)

SIGNALS (apps/tenants/signals.py):
- On Organization creation: auto-create OrganizationMembership with OWNER role, activate default modules (PROJECT_CENTER + ANALYTICS), create Stripe customer

PERMISSIONS (apps/tenants/permissions.py):
- HasModuleAccess(module_key): Check if the request's organization has the specified module active. Use as a DRF permission class. Example: permission_classes = [IsAuthenticated, HasModuleAccess('CRM')]

URLS: Mount everything under /api/v1/organizations/

Write Django migrations. Add comprehensive admin registration with list_display, list_filter, and search_fields for all models. Write a management command 'create_demo_org' that creates a demo organization with sample data.

Section 3: Authentication, Registration & User Roles

Goal: Custom User model, registration with org creation flow, JWT auth, social OAuth (Google/GitHub), email verification, password reset, role-based permissions.

Key Decisions:

- Email-based login (no username)
- Registration creates User + Organization + first Membership in one transaction
- JWT tokens for API auth, session auth for Django admin
- Role-based permissions per organization (not global)

Claude Code Prompt

Build the authentication and user management system for Builders Stream Pro in apps/accounts/. This handles user registration, login, JWT auth, social auth, email verification, and role-based access control.

MODELS (apps/accounts/models.py):

1. Custom User model (extends AbstractBaseUser + PermissionsMixin):
   - id (UUID primary key)
   - email (EmailField, unique — this is the USERNAME_FIELD)
   - first_name, last_name (CharField, max 150)
   - phone (CharField, max 20, optional)
   - avatar (ImageField, upload to 'avatars/', nullable)
   - job_title (CharField, max 100, optional)
   - is_active, is_staff, is_superuser (standard Django)
   - email_verified (BooleanField, default=False)
   - email_verification_token (UUIDField, nullable)
   - last_active_organization (FK to Organization, nullable — remembers last used org)
   - timezone (CharField, default='America/Chicago', choices from common US timezones)
   - notification_preferences (JSONField, default=dict — email_daily_digest, push_approvals, etc.)
   - date_joined (DateTimeField, auto_now_add)
   - USERNAME_FIELD = 'email', REQUIRED_FIELDS = ['first_name', 'last_name']

2. Custom UserManager:
   - create_user(email, password, first_name, last_name, **extra)
   - create_superuser(email, password, **extra)
   - Normalize email to lowercase

SERIALIZERS (apps/accounts/serializers.py):

1. RegisterSerializer:
   - Fields: email, password, password_confirm, first_name, last_name, phone, company_name, industry_type
   - Validate passwords match and meet strength requirements (8+ chars, 1 upper, 1 number)
   - On create: wrap in transaction.atomic() — create User, create Organization (using company_name + industry_type), create OWNER membership, send verification email

2. LoginSerializer:
   - Fields: email, password
   - Returns JWT access + refresh tokens plus user profile and organization list

3. UserProfileSerializer:
   - Full user profile with read-only organization memberships nested
   - Writable fields: first_name, last_name, phone, avatar, job_title, timezone, notification_preferences

4. ChangePasswordSerializer, ForgotPasswordSerializer, ResetPasswordSerializer

5. InviteAcceptSerializer:
   - Fields: token, password, first_name, last_name
   - Validate invitation token, create/update user, activate membership

VIEWS (apps/accounts/views.py):

1. RegisterView (APIView, AllowAny):
   - POST: Create user + org + membership, send verification email, return JWT tokens
   - Rate limit: 5 registrations per IP per hour

2. LoginView (use SimpleJWT TokenObtainPairView customized):
   - Add user profile and organizations list to token response
   - Check email_verified — if not, still allow login but include warning

3. VerifyEmailView (APIView, AllowAny):
   - GET with token param: verify email, redirect to frontend

4. ProfileView (RetrieveUpdateAPIView):
   - GET/PATCH current user profile

5. PasswordViews: ChangePasswordView, ForgotPasswordView (sends reset email), ResetPasswordView

6. GoogleOAuthCallbackView, GitHubOAuthCallbackView:
   - Handle OAuth callback, create or link user, return JWT tokens
   - If new user from OAuth, prompt for company_name on frontend before creating org

7. UserOrganizationsView (ListAPIView):
   - List all organizations the current user belongs to with their roles

TASKS (apps/accounts/tasks.py — Celery):
- send_verification_email(user_id): Send email with verification link
- send_password_reset_email(user_id, token): Send password reset
- send_invitation_email(membership_id): Send org invitation with accept link

PERMISSIONS UTILITY (apps/core/permissions.py — extend):
- Create a role_required(min_role) decorator/permission that checks the user's role in the current organization against a hierarchy: OWNER > ADMIN > PROJECT_MANAGER > ESTIMATOR > ACCOUNTANT > FIELD_WORKER > READ_ONLY
- Example usage: @role_required('PROJECT_MANAGER') means PM, ADMIN, and OWNER can access

URLS: Mount under /api/v1/auth/ for login/register/verify/password and /api/v1/users/ for profile

Also create a FRONTEND component plan (don't build yet, just document):

- /login — email/password form with Google/GitHub OAuth buttons
- /register — multi-step: (1) email + password, (2) company name + industry type, (3) verify email prompt
- /dashboard — redirect target after login (built in Section 5)
- /invite/accept/:token — accept invitation form
- /forgot-password, /reset-password/:token

Write comprehensive tests: test registration creates user+org+membership, test JWT login/refresh, test email verification flow, test role-based permission checks, test invitation accept flow.

Section 4: Subscription & Billing (Stripe Integration)

Goal: Stripe subscription management with tiered plans, module-gating, user limits, trial periods, and a billing portal.

Key Decisions:

- 3 tiers: Starter ($15/user/mo, 5 users), Professional ($50/user/mo, 25 users), Enterprise ($125/user/mo, unlimited)
- 14-day free trial with full access
- Module access gated by plan tier
- Stripe Customer Portal for self-service billing management
- Webhooks for subscription lifecycle events

Claude Code Prompt

Build the Stripe subscription billing system for Builders Stream Pro in apps/billing/. This gates feature access by plan tier and enforces user limits.

PLAN CONFIGURATION (apps/billing/plans.py — constants, not a model):

```
PLAN_CONFIG = {
    'TRIAL': {
        'name': 'Free Trial',
        'max_users': 5,
        'price_monthly': 0,
        'modules': ['PROJECT_CENTER', 'CRM', 'ESTIMATING', 'SCHEDULING', 'FINANCIALS', 'CLIENT_PORTAL',
'DOCUMENTS', 'FIELD_OPS', 'QUALITY_SAFETY', 'ANALYTICS'],
        'trial_days': 14,
    },
    'STARTER': {
        'name': 'Starter',
        'max_users': 5,
        'stripe_price_monthly': 'price_starter_monthly',  # Stripe Price ID placeholder
        'stripe_price_annual': 'price_starter_annual',
        'price_monthly_per_user': 1500,  # $15.00 in cents
        'price_annual_per_user': 14400,  # $144.00/yr ($12/mo effective)
        'modules': ['PROJECT_CENTER', 'CRM', 'ESTIMATING', 'CLIENT_PORTAL', 'ANALYTICS'],
    },
    'PROFESSIONAL': {
        'name': 'Professional',
        'max_users': 25,
        'stripe_price_monthly': 'price_pro_monthly',
        'stripe_price_annual': 'price_pro_annual',
        'price_monthly_per_user': 5000,
        'price_annual_per_user': 48000,
        'modules': ['PROJECT_CENTER', 'CRM', 'ESTIMATING', 'SCHEDULING', 'FINANCIALS', 'CLIENT_PORTAL',
'DOCUMENTS', 'FIELD_OPS', 'QUALITY_SAFETY', 'ANALYTICS'],
    },
    'ENTERPRISE': {
        'name': 'Enterprise',
        'max_users': 999,
        'stripe_price_monthly': 'price_enterprise_monthly',
        'stripe_price_annual': 'price_enterprise_annual',
        'price_monthly_per_user': 12500,
        'price_annual_per_user': 120000,
        'modules': ['PROJECT_CENTER', 'CRM', 'ESTIMATING', 'SCHEDULING', 'FINANCIALS', 'CLIENT_PORTAL',
'DOCUMENTS', 'FIELD_OPS', 'QUALITY_SAFETY', 'PAYROLL', 'SERVICE_WARRANTY', 'ANALYTICS'],
    },
```

}

MODELS (apps/billing/models.py):

1. SubscriptionEvent (audit log):
   - organization (FK)
   - event_type (CharField: CREATED, UPDATED, CANCELED, PAYMENT_SUCCEEDED, PAYMENT_FAILED, TRIAL_ENDING, TRIAL_ENDED)
   - stripe_event_id (CharField, unique)
   - data (JSONField — raw event payload)
   - created_at (DateTimeField)

2. Invoice (synced from Stripe):
   - organization (FK)
   - stripe_invoice_id (CharField, unique)
   - amount_due, amount_paid (IntegerField — cents)
   - currency (CharField, default='usd')
   - status (CharField: DRAFT, OPEN, PAID, VOID, UNCOLLECTIBLE)
   - period_start, period_end (DateTimeField)
   - hosted_invoice_url (URLField, nullable)
   - pdf_url (URLField, nullable)
   - created_at (DateTimeField)

SERVICES (apps/billing/services.py):

1. StripeService class:
   - create_customer(organization): Create Stripe customer, save stripe_customer_id on org
   - create_subscription(organization, plan_key, billing_interval='monthly'): Create Stripe subscription with per-seat pricing for current user count
   - update_subscription(organization, new_plan_key): Upgrade/downgrade with proration
   - cancel_subscription(organization, at_period_end=True): Cancel at period end
   - create_billing_portal_session(organization): Return Stripe Customer Portal URL
   - sync_subscription_status(organization): Pull latest status from Stripe
   - update_seat_count(organization): Update Stripe subscription quantity when members change
   - handle_trial_conversion(organization, plan_key): Convert trial to paid plan

2. ModuleGateService:
   - check_module_access(organization, module_key) -> bool: Check plan allows module
   - get_available_modules(organization) -> list: List accessible modules
   - check_user_limit(organization) -> bool: Can org add more users?
   - get_plan_limits(organization) -> dict: Return all limits for current plan

WEBHOOK HANDLER (apps/billing/webhooks.py):

- StripeWebhookView (APIView, no auth): Verify Stripe signature, handle events:
  - customer.subscription.created -> Update org plan + status + activate modules
  - customer.subscription.updated -> Sync plan changes, module access
  - customer.subscription.deleted -> Set org to canceled, schedule data retention notice
  - invoice.payment_succeeded -> Create Invoice record, update status to ACTIVE
  - invoice.payment_failed -> Set status to PAST_DUE, send failure notification email
  - customer.subscription.trial_will_end -> Send "trial ending" email 3 days before

VIEWS (apps/billing/views.py):

1. SubscriptionView (APIView, OWNER/ADMIN only):
   - GET: Return current plan, status, usage stats (users used/max, modules active)
   - POST: Create new subscription (plan_key, billing_interval)
   - PATCH: Change plan (upgrade/downgrade)
   - DELETE: Cancel subscription

2. BillingPortalView (APIView, OWNER only):
   - POST: Return Stripe Customer Portal session URL for self-service management

3. InvoiceListView (ListAPIView, OWNER/ADMIN/ACCOUNTANT):
   - List all invoices for the org with pagination

4. PlanComparisonView (APIView, AllowAny):
   - GET: Return all plans with features, pricing, and module lists (public endpoint for pricing page)

TASKS (apps/billing/tasks.py):
- check_expiring_trials(): Daily task — find orgs where trial_ends_at is within 3 days, send reminder email
- expire_trials(): Daily task — deactivate orgs past trial_ends_at that haven't subscribed
- sync_all_subscriptions(): Weekly task — reconcile all active subscriptions with Stripe

MIDDLEWARE INTEGRATION:
- Create SubscriptionRequiredMiddleware: For API requests (not webhook, not auth endpoints), check that the org's subscription is ACTIVE or TRIALING. If PAST_DUE, allow read-only access. If CANCELED/EXPIRED, return 402 with upgrade prompt.

URLS: Mount subscription views under /api/v1/billing/, webhook at /api/v1/webhooks/stripe/

Write tests: test plan gating logic, test module access checks, test user limit enforcement, test webhook signature verification and event handling, test trial expiration flow.

Section 5: Project Command Center — Dashboard & Lifecycle

Goal: The main dashboard contractors see after login — project cards, financial snapshot, action items, activity stream, weather. Plus the full project lifecycle state machine.

Key Decisions:

- Dashboard is widget-based with user-configurable layouts saved per user
- Project follows a strict lifecycle: Lead → Prospect → Estimate → Proposal → Contract → Production → Punch List → Closeout
- Stage-gate controls require approvals/documents before progression
- Real-time data aggregation via Celery tasks for performance

Claude Code Prompt

Build the Project Command Center for Builders Stream Pro in apps/projects/. This is the main dashboard and project lifecycle management — the core of the entire platform.

MODELS (apps/projects/models.py):

1. Project (extends TenantModel):
   - id (UUID pk)
   - name (CharField, max 200)
   - project_number (CharField, unique per org — auto-generated like "BSP-2026-001")
   - status (CharField choices: LEAD, PROSPECT, ESTIMATE, PROPOSAL, CONTRACT, PRODUCTION, PUNCH_LIST, CLOSEOUT, COMPLETED, CANCELED)
   - project_type (CharField choices: RESIDENTIAL_REMODEL, KITCHEN_BATH, ADDITION, NEW_HOME, COMMERCIAL_BUILDOUT, COMMERCIAL_RENOVATION, ROOFING, EXTERIOR, SPECIALTY)
   - client (FK to 'crm.Contact', nullable — linked when client exists)
   - description (TextField)
   - address_line1, address_line2, city, state, zip_code (job site address)
   - latitude, longitude (DecimalField, nullable — for weather/mapping)
   - estimated_value (DecimalField, max_digits=12, decimal_places=2, default=0)
   - actual_revenue (DecimalField, default=0)
   - estimated_cost (DecimalField, default=0)
   - actual_cost (DecimalField, default=0)
   - start_date (DateField, nullable)
   - estimated_completion (DateField, nullable)
   - actual_completion (DateField, nullable)
   - project_manager (FK to User, nullable)
   - assigned_team (M2M to User through ProjectTeamMember)
   - health_score (IntegerField, 0-100, auto-calculated)
   - health_status (CharField: GREEN, YELLOW, RED — derived from score)
   - completion_percentage (IntegerField, 0-100)
   - tags (JSONField, default=list)
   - custom_fields (JSONField, default=dict)
   - is_archived (BooleanField, default=False)

2. ProjectTeamMember:
   - project (FK to Project)
   - user (FK to User)
   - role (CharField: PROJECT_MANAGER, SUPERINTENDENT, ESTIMATOR, FOREMAN, SUBCONTRACTOR, OTHER)
   - added_at (DateTimeField)
   - unique_together = [project, user]

3. ProjectStageTransition (audit log):
   - project (FK to Project)

- from_status, to_status (CharField)
- transitioned_by (FK to User)
- notes (TextField, optional)
- requirements_met (JSONField — list of checks that passed)
- created_at (DateTimeField)

4. ProjectMilestone:
- project (FK to Project)
- name (CharField)
- description (TextField, optional)
- due_date (DateField)
- completed_date (DateField, nullable)
- is_completed (BooleanField, default=False)
- sort_order (IntegerField)
- notify_client (BooleanField, default=False)

5. ActionItem:
- project (FK to Project, nullable — can be org-wide)
- organization (FK to Organization)
- title (CharField)
- description (TextField)
- item_type (CharField: TASK, APPROVAL, OVERDUE_INVOICE, WEATHER_ALERT, DEADLINE, CLIENT_MESSAGE, EXPIRING_BID, INSPECTION_DUE)
- priority (CharField: CRITICAL, HIGH, MEDIUM, LOW)
- assigned_to (FK to User, nullable)
- due_date (DateTimeField, nullable)
- is_resolved (BooleanField, default=False)
- resolved_at (DateTimeField, nullable)
- source_type (CharField — generic relation source, e.g., 'estimate', 'invoice')
- source_id (UUIDField, nullable)
- created_at (DateTimeField)

6. ActivityLog:
- organization (FK to Organization)
- project (FK to Project, nullable)
- user (FK to User)
- action (CharField: CREATED, UPDATED, STATUS_CHANGED, UPLOADED, APPROVED, REJECTED, COMMENTED, CHECKED_IN, CHECKED_OUT)
- entity_type (CharField — e.g., 'project', 'estimate', 'daily_log', 'photo')
- entity_id (UUIDField)
- description (CharField, max 500 — human-readable summary)
- metadata (JSONField, default=dict — extra context)
- created_at (DateTimeField, db_index=True)

7. DashboardLayout:
   - user (FK to User)
   - organization (FK to Organization)
   - layout (JSONField — array of widget configs with position, size, widget_type)
   - is_default (BooleanField)
   - unique_together = [user, organization]

SERVICES (apps/projects/services.py):

1. ProjectLifecycleService:
   - transition_status(project, new_status, user, notes=''): Validate the transition is allowed (define valid transitions map), check stage-gate requirements, create audit log, trigger status change signals
   - Valid transitions: LEAD->PROSPECT, PROSPECT->ESTIMATE, ESTIMATE->PROPOSAL, PROPOSAL->CONTRACT, CONTRACT->PRODUCTION, PRODUCTION->PUNCH_LIST, PUNCH_LIST->CLOSEOUT, CLOSEOUT->COMPLETED. Also allow CANCELED from any status.
   - Stage-gate requirements (soft/configurable): ESTIMATE->PROPOSAL requires at least one estimate exists. PROPOSAL->CONTRACT requires signed proposal. CONTRACT->PRODUCTION requires deposit payment received.
   - calculate_health_score(project): Score based on budget variance (actual vs estimated costs), schedule variance (% complete vs % time elapsed), action items overdue count, client response times. Return 0-100 score + GREEN/YELLOW/RED status.

2. DashboardService:
   - get_dashboard_data(organization, user): Aggregate all widget data:
     - active_projects: list with health scores, completion %, budget status
     - financial_snapshot: total revenue, total costs, gross margin, cash flow forecast (next 30/60/90 days)
     - schedule_overview: projects with upcoming milestones this week
     - action_items: top 20 by priority, filtered to user's assignments
     - activity_stream: last 50 activities across all projects
     - weather: forecasts for active job sites (stub for now, implement in integrations phase)

3. ProjectNumberService:
   - generate_project_number(organization): Auto-increment per org per year. Format: "BSP-{YEAR}-{SEQ:03d}"

SERIALIZERS (apps/projects/serializers.py):
- ProjectListSerializer (compact: id, name, number, status, health_status, completion_percentage, estimated_value, project_manager name, client name)
- ProjectDetailSerializer (full model with nested milestones, team members, recent activity)
- ProjectCreateSerializer (name, type, client, address fields, description, start_date, estimated_completion)
- ProjectStatusTransitionSerializer (new_status, notes)
- MilestoneSerializer, ActionItemSerializer, ActivityLogSerializer
- DashboardSerializer (nested structure for all dashboard widgets)

VIEWS (apps/projects/views.py):

1. ProjectViewSet (ModelViewSet):
   - list: Filter by status, project_type, project_manager, date range. Search by name/number.
   - create: Auto-generate project number, set initial status to LEAD
   - retrieve: Full detail with related data
   - update/partial_update: Standard CRUD
   - Custom actions:
     - @action transition_status (POST): Use ProjectLifecycleService
     - @action team_members (GET/POST/DELETE): Manage team
     - @action milestones (GET/POST): Manage milestones
     - @action activity (GET): Activity log for this project

2. DashboardView (APIView):
   - GET: Return full dashboard data for current org/user via DashboardService
   - Uses caching (Redis, 60s TTL) to avoid expensive aggregation on every request

3. DashboardLayoutView (RetrieveUpdateAPIView):
   - GET/PUT: User's custom dashboard widget layout

4. ActionItemViewSet (ModelViewSet):
   - List (filterable by project, type, priority, assigned_to, is_resolved), update, resolve

5. ActivityStreamView (ListAPIView):
   - Paginated activity stream for the org, filterable by project and user

TASKS (apps/projects/tasks.py):
- calculate_all_health_scores(): Hourly task — recalculate health scores for all active projects
- generate_action_items(): Every 30 min — scan for overdue items, upcoming deadlines, unanswered messages, create/update ActionItems
- cache_dashboard_data(organization_id): Called after significant data changes to pre-warm dashboard cache

SIGNALS (apps/projects/signals.py):
- On project status change: Create ActivityLog entry, send notifications to team, update client portal if CLIENT_PORTAL module active
- On project create: Log activity, create default milestones based on project_type template

URLS: /api/v1/projects/, /api/v1/dashboard/, /api/v1/action-items/, /api/v1/activity/

Write Django admin with custom filters (by status, health, PM). Write model tests for lifecycle transitions and health scoring. Write API tests for dashboard aggregation.

PHASE 2 — CLIENT EXPERIENCE (Sections 6–9)

Section 6: CRM & Lead Management

Goal: Lead capture from multiple sources, sales pipeline with drag-drop stages, contact management, automated follow-ups, and marketing tools.

Claude Code Prompt

Build the CRM & Lead Management module for Builders Stream Pro in apps/crm/. This module requires HasModuleAccess('CRM') permission on all endpoints.

MODELS (apps/crm/models.py):

1. Contact (extends TenantModel):
   - id (UUID pk)
   - contact_type (CharField: LEAD, CLIENT, SUBCONTRACTOR, VENDOR, ARCHITECT, OTHER)
   - first_name, last_name, email, phone, mobile_phone
   - company_name (CharField, nullable)
   - job_title (CharField, nullable)
   - address_line1, address_line2, city, state, zip_code
   - source (CharField: WEBSITE_FORM, PHONE, EMAIL, REFERRAL, HOME_ADVISOR, ANGI, HOUZZ, HOME_SHOW, SOCIAL_MEDIA, WALK_IN, OTHER)
   - referred_by (FK to Contact, nullable, related_name='referrals')
   - lead_score (IntegerField, 0-100, default=0)
   - tags (JSONField, default=list)
   - custom_fields (JSONField, default=dict)
   - notes (TextField, blank)
   - is_active (BooleanField, default=True)

2. Company (extends TenantModel):
   - name, phone, email, website
   - address fields
   - company_type (CharField: CLIENT_COMPANY, SUBCONTRACTOR, VENDOR, SUPPLIER, ARCHITECT_FIRM)
   - contacts (reverse FK from Contact via company FK)
   - insurance_expiry (DateField, nullable)
   - license_number (CharField, nullable)
   - license_expiry (DateField, nullable)
   - performance_rating (DecimalField, 1-5, nullable)
   - notes (TextField)

3. PipelineStage (extends TenantModel):
   - name (CharField)
   - sort_order (IntegerField)
   - color (CharField, hex color)
   - is_won_stage (BooleanField, default=False)
   - is_lost_stage (BooleanField, default=False)
   - auto_actions (JSONField — triggers when lead enters stage)
   - Default stages seeded on org creation: New Lead, Contacted, Site Visit Scheduled, Site Visit Complete, Estimate Sent, Negotiating, Won, Lost

4. Lead (extends TenantModel):
   - contact (FK to Contact)
   - project_type (CharField, same choices as Project.project_type)
   - pipeline_stage (FK to PipelineStage)
   - assigned_to (FK to User, nullable)
   - estimated_value (DecimalField, nullable)
   - estimated_start (DateField, nullable)
   - description (TextField)
   - urgency (CharField: HOT, WARM, COLD)
   - lost_reason (CharField, nullable — populated when moved to lost stage)
   - lost_to_competitor (CharField, nullable)
   - converted_project (FK to Project, nullable — set when lead converts)
   - last_contacted_at (DateTimeField, nullable)
   - next_follow_up (DateTimeField, nullable)

5. Interaction (extends TenantModel — communication log):
   - contact (FK to Contact)
   - lead (FK to Lead, nullable)
   - interaction_type (CharField: EMAIL, PHONE_CALL, SMS, SITE_VISIT, MEETING, NOTE)
   - direction (CharField: INBOUND, OUTBOUND)
   - subject (CharField)
   - body (TextField)
   - logged_by (FK to User)
   - occurred_at (DateTimeField)

6. AutomationRule (extends TenantModel):
   - name (CharField)
   - trigger_type (CharField: STAGE_CHANGE, TIME_DELAY, LEAD_SCORE_CHANGE, NO_ACTIVITY)
   - trigger_config (JSONField — e.g., {"stage": "site_visit_complete"} or {"days_inactive": 7})
   - action_type (CharField: SEND_EMAIL, SEND_SMS, CREATE_TASK, ASSIGN_LEAD, CHANGE_STAGE, NOTIFY_USER)
   - action_config (JSONField — e.g., {"template_id": "...", "assign_to": "senior_estimator"})
   - is_active (BooleanField)

7. EmailTemplate (extends TenantModel):
   - name, subject, body (supports {{first_name}}, {{project_type}}, {{company_name}} variables)
   - template_type (CharField: FOLLOW_UP, THANK_YOU, ESTIMATE_REMINDER, REVIEW_REQUEST, MARKETING)

SERVICES:
- LeadScoringService: Calculate lead score based on estimated_value, urgency, source quality, engagement (interactions count), response time. Update score on relevant changes.
- LeadConversionService: Convert lead to project — create Project from Lead data, link contact as client, transition lead to won stage, create activity log.

- AutomationEngine: Process automation rules — called by Celery tasks on triggers.

SERIALIZERS: Full CRUD serializers for all models. LeadSerializer includes nested contact info and current stage. PipelineBoardSerializer returns stages with leads grouped per stage (for kanban view).

VIEWS:
- ContactViewSet: Full CRUD with search (name, email, phone, company), filter by type/source/tags
- CompanyViewSet: Full CRUD with nested contacts list
- LeadViewSet: Full CRUD plus:
  - @action move_stage (POST): Move lead to new pipeline stage, trigger automations
  - @action convert_to_project (POST): Use LeadConversionService
  - @action log_interaction (POST): Create Interaction record
  - @action pipeline_board (GET): Return kanban-style board data grouped by stage
- InteractionViewSet: List/create interactions for a contact
- AutomationRuleViewSet: CRUD for automation rules (ADMIN only)
- LeadAnalyticsView: Conversion rates by source, by salesperson, by project type, win/loss reasons

TASKS:
- process_time_based_automations(): Every 15 min — check for leads that match time-based triggers
- calculate_lead_scores(): Hourly — recalculate all active lead scores
- send_follow_up_reminders(): Daily — notify assigned users of leads needing follow-up

SIGNALS:
- On Lead stage change: Run automation engine, create activity log
- On Contact creation from external source: Auto-create Lead

Seed default pipeline stages on org creation. Write tests for lead scoring, pipeline movement, and conversion flow.

Section 7: Estimating & Digital Takeoff Engine

Goal: Cost assemblies, material/labor cost database, proposal generation with e-signature, Excel/PDF export.

Claude Code Prompt

Build the Estimating & Digital Takeoff module for Builders Stream Pro in apps/estimating/. Requires HasModuleAccess('ESTIMATING').

MODELS (apps/estimating/models.py):

1. CostCode (extends TenantModel — hierarchical cost code structure):
   - code (CharField, e.g., "03-100")
   - name (CharField, e.g., "Concrete Foundations")
   - parent (FK to self, nullable — for hierarchy)
   - description (TextField)
   - default_unit (CharField: SQFT, LNFT, EACH, HOUR, CUYD, ALLOW, LOT)
   - sort_order (IntegerField)

2. CostItem (extends TenantModel — material/labor/equipment rate):
   - cost_code (FK to CostCode, nullable)
   - name (CharField)
   - item_type (CharField: MATERIAL, LABOR, EQUIPMENT, SUBCONTRACTOR, OTHER)
   - unit (CharField, same choices as CostCode.default_unit)
   - unit_cost (DecimalField, max_digits=10, decimal_places=2)
   - supplier (FK to 'crm.Company', nullable)
   - last_updated (DateTimeField)
   - notes (TextField)

3. Assembly (extends TenantModel — pre-built cost groups):
   - name (CharField, e.g., "Install Kitchen Cabinet - Standard")
   - description (TextField)
   - category (CharField: KITCHEN, BATH, FLOORING, ELECTRICAL, PLUMBING, HVAC, FRAMING, ROOFING, EXTERIOR, GENERAL)
   - items (reverse FK from AssemblyItem)
   - total_cost (DecimalField — calculated from items)

4. AssemblyItem:
   - assembly (FK to Assembly)
   - cost_item (FK to CostItem)
   - quantity (DecimalField)
   - sort_order (IntegerField)

5. Estimate (extends TenantModel):
   - project (FK to Project)
   - name (CharField, e.g., "Kitchen Remodel - Option A")
   - version (IntegerField, default=1)
   - status (CharField: DRAFT, REVIEW, SENT, ACCEPTED, REJECTED, EXPIRED)
   - subtotal (DecimalField)

- markup_percentage (DecimalField)
- markup_amount (DecimalField)
- tax_percentage (DecimalField, default=0)
- tax_amount (DecimalField)
- total (DecimalField)
- confidence_level (CharField: LOW, MEDIUM, HIGH — based on historical accuracy)
- low_estimate, high_estimate (DecimalField — confidence range)
- notes (TextField)
- internal_notes (TextField — not shown to client)
- prepared_by (FK to User)
- valid_until (DateField, nullable)

6. EstimateSection (group line items):
- estimate (FK to Estimate)
- name (CharField, e.g., "Demolition", "Framing", "Finishes")
- sort_order (IntegerField)
- subtotal (DecimalField — calculated)

7. EstimateLineItem:
- section (FK to EstimateSection)
- cost_code (FK to CostCode, nullable)
- description (CharField)
- quantity (DecimalField)
- unit (CharField)
- unit_cost (DecimalField)
- total (DecimalField — quantity * unit_cost)
- markup_percentage (DecimalField, nullable — override estimate-level markup)
- is_optional (BooleanField — client can add/remove optional items)
- is_visible_to_client (BooleanField, default=True)
- notes (TextField)
- sort_order (IntegerField)

8. Proposal (extends TenantModel):
- estimate (OneToOneField to Estimate)
- project (FK to Project)
- template (FK to ProposalTemplate)
- presentation_mode (CharField: ITEMIZED, LUMP_SUM, PHASED)
- cover_letter (TextField)
- terms_and_conditions (TextField)
- payment_schedule (JSONField — array of {milestone, percentage, amount})
- sent_at (DateTimeField, nullable)
- viewed_at (DateTimeField, nullable)
- signed_at (DateTimeField, nullable)
- signature_data (JSONField, nullable — e-signature capture)

- signer_name (CharField, nullable)
- signer_ip (GenericIPAddressField, nullable)
- expiry_date (DateField)
- public_token (UUIDField, unique — for unauthenticated client access)

9. ProposalTemplate (extends TenantModel):
- name, description
- template_content (JSONField — sections, styling, logo placement)
- is_default (BooleanField)

SERVICES:
- EstimateCalculationService: Recalculate all totals when line items change. Apply markup, tax. Generate confidence ranges based on historical data (compare to actual costs from completed projects with similar cost codes).
- ProposalService: Generate proposal from estimate, send via email with public link, track views, handle e-signature capture, convert signed proposal to contract (transition project status).
- ExportService: Export estimate to Excel (.xlsx with formatted sheets per section) and PDF (professional layout with company branding).
- AssemblyService: Apply assembly to estimate — creates line items from assembly items with current costs.

SERIALIZERS: Full CRUD for all models. EstimateDetailSerializer includes nested sections with line items. ProposalPublicSerializer (limited fields for unauthenticated client view).

VIEWS:
- CostCodeViewSet, CostItemViewSet, AssemblyViewSet: Full CRUD for cost database management
- EstimateViewSet: Full CRUD plus:
  - @action add_assembly (POST): Apply assembly to a section
  - @action duplicate (POST): Create new version of estimate
  - @action recalculate (POST): Force recalculation of all totals
  - @action export_excel (GET): Download .xlsx
  - @action export_pdf (GET): Download PDF
- ProposalViewSet:
  - @action send_to_client (POST): Email proposal with public link
  - @action sign (POST, AllowAny with token): E-signature submission from client
- ProposalPublicView (RetrieveAPIView, AllowAny): Public proposal view by token

TASKS:
- check_expiring_proposals(): Daily — alert on proposals expiring within 3 days
- update_cost_database(): Weekly — flag cost items not updated in 90+ days

Seed sample cost codes (CSI Division structure) and common assemblies for new orgs. Write tests for calculation accuracy, proposal signing flow, and export generation.

Section 8: Client Collaboration Portal

Goal: White-labeled client-facing portal with project visibility, selection management, approvals, payments, and communication.

Claude Code Prompt

Build the Client Collaboration Portal for Builders Stream Pro in apps/clients/. Requires HasModuleAccess('CLIENT_PORTAL'). This is a separate authenticated experience for the contractor's clients (homeowners, commercial clients) — NOT the contractor's internal users.

MODELS (apps/clients/models.py):

1. ClientPortalAccess (extends TenantModel):
   - contact (FK to 'crm.Contact')
   - project (FK to Project)
   - access_token (UUIDField, unique — for magic link login)
   - email (EmailField — may differ from contact email)
   - pin_code (CharField, max 6, nullable — optional PIN for extra security)
   - is_active (BooleanField, default=True)
   - last_login (DateTimeField, nullable)
   - permissions (JSONField, default: {"view_photos": true, "view_schedule": true, "view_documents": true, "make_payments": true, "approve_selections": true, "send_messages": true})

2. Selection (extends TenantModel):
   - project (FK to Project)
   - category (CharField: FLOORING, COUNTERTOPS, CABINETS, FIXTURES, PAINT, TILE, HARDWARE, APPLIANCES, LIGHTING, OTHER)
   - name (CharField, e.g., "Master Bath Vanity")
   - description (TextField)
   - options (reverse FK from SelectionOption)
   - selected_option (FK to SelectionOption, nullable)
   - status (CharField: PENDING, CLIENT_REVIEW, APPROVED, ORDERED, INSTALLED)
   - due_date (DateField, nullable)
   - assigned_to_client (BooleanField, default=True)
   - sort_order (IntegerField)

3. SelectionOption:
   - selection (FK to Selection)
   - name (CharField, e.g., "Quartz - Calacatta Gold")
   - description (TextField)
   - price (DecimalField)
   - price_difference (DecimalField — diff from base/standard option)
   - lead_time_days (IntegerField, nullable)
   - supplier (CharField)
   - image (ImageField, nullable)
   - spec_sheet_url (URLField, nullable)
   - is_recommended (BooleanField, default=False)
   - sort_order (IntegerField)

4. ClientApproval (extends TenantModel):
- project (FK to Project)
- approval_type (CharField: CHANGE_ORDER, SELECTION, DRAW_REQUEST, DESIGN_MODIFICATION, SCHEDULE_CHANGE, OTHER)
- title (CharField)
- description (TextField)
- source_type (CharField — generic relation to change order, selection, etc.)
- source_id (UUIDField)
- status (CharField: PENDING, APPROVED, REJECTED, EXPIRED)
- requested_at (DateTimeField)
- responded_at (DateTimeField, nullable)
- response_notes (TextField)
- client_signature (JSONField, nullable)
- reminded_count (IntegerField, default=0)

5. ClientMessage (extends TenantModel):
- project (FK to Project)
- sender_type (CharField: CONTRACTOR, CLIENT)
- sender_user (FK to User, nullable — for contractor messages)
- sender_contact (FK to Contact, nullable — for client messages)
- subject (CharField)
- body (TextField)
- is_read (BooleanField, default=False)
- read_at (DateTimeField, nullable)
- attachments (JSONField, default=list — list of S3 file keys)

6. ClientSatisfactionSurvey (extends TenantModel):
- project (FK to Project)
- contact (FK to Contact)
- milestone (CharField — which milestone triggered this)
- rating (IntegerField, 1-10)
- nps_score (IntegerField, 0-10, nullable)
- feedback (TextField)
- submitted_at (DateTimeField)

7. PortalBranding (extends TenantModel):
- logo (ImageField)
- primary_color, secondary_color (CharField, hex)
- company_name_override (CharField, nullable)
- welcome_message (TextField)
- custom_domain (CharField, nullable — for white-label)
- OneToOne with Organization

SERVICES:

- ClientAuthService: Generate magic link tokens, validate access, create portal sessions (short-lived JWT with client scope), send magic link emails.
- SelectionService: Present selections to client, record choices, calculate price impact on project total, notify contractor of decisions.
- ApprovalService: Create approval requests, send notifications, track response times, auto-remind at configurable intervals.
- ClientNotificationService: Send automated updates to clients — milestone reached, photos uploaded, schedule changes, payment due. Respect frequency preferences (real-time, daily digest, weekly).

SERIALIZERS:
- ClientPortalSerializer: Everything the client sees — project status, milestones, photos, schedule, documents, messages, pending approvals, selections, payment history. All in client-friendly language (no internal jargon).
- SelectionSerializer, ClientApprovalSerializer, ClientMessageSerializer
- PaymentHistorySerializer: List of invoices with status and pay links

VIEWS (two sets — contractor-facing and client-facing):

Contractor-facing (standard auth):
- ClientPortalAccessViewSet: Create/manage portal access for project clients
- SelectionViewSet: Create/manage selections and options
- ClientApprovalViewSet: Create approval requests, view status
- ClientMessageViewSet: Send messages, view conversation threads

Client-facing (token auth — /api/v1/portal/):
- ClientLoginView (AllowAny): Accept magic link token, return scoped JWT
- ClientDashboardView: Project overview, milestones, recent activity
- ClientSelectionsView: View selections, choose options, approve
- ClientApprovalsView: View pending approvals, approve/reject with signature
- ClientPhotosView: View project photo galleries
- ClientDocumentsView: View shared documents
- ClientMessagesView: View/send messages
- ClientPaymentsView: View invoices, initiate payment (Stripe Checkout session)
- ClientScheduleView: View simplified project schedule

TASKS:
- send_client_daily_digest(): Daily at 8am — compile updates for clients with daily digest preference
- send_approval_reminders(): Daily — remind clients of pending approvals older than configured threshold
- send_satisfaction_survey(): Triggered by milestone completion — send survey after configurable delay

Write a custom JWT authentication backend for client portal tokens (separate from contractor auth — scoped to specific project + contact). Write tests for magic link flow, selection approval, and message threading.

Section 9: Document & Photo Control

Goal: Construction-grade document management with version control, RFI tracking, photo management with AI categorization, and S3 storage.

Claude Code Prompt

Build the Document & Photo Control module for Builders Stream Pro in apps/documents/. Requires HasModuleAccess('DOCUMENTS'). Uses AWS S3 for all file storage.

MODELS (apps/documents/models.py):

1. DocumentFolder (extends TenantModel):
   - project (FK to Project, nullable — org-level folders if null)
   - name (CharField)
   - parent (FK to self, nullable)
   - folder_type (CharField: GENERAL, PLANS, PERMITS, CONTRACTS, SUBMITTALS, RFI, PHOTOS, SAFETY, WARRANTY)
   - access_level (CharField: ALL_TEAM, MANAGERS_ONLY, ADMIN_ONLY, CLIENT_VISIBLE)
   - sort_order (IntegerField)

2. Document (extends TenantModel):
   - folder (FK to DocumentFolder)
   - project (FK to Project, nullable)
   - title (CharField)
   - description (TextField)
   - file_key (CharField — S3 object key)
   - file_name (CharField — original filename)
   - file_size (BigIntegerField — bytes)
   - content_type (CharField — MIME type)
   - version (IntegerField, default=1)
   - is_current_version (BooleanField, default=True)
   - previous_version (FK to self, nullable)
   - uploaded_by (FK to User)
   - tags (JSONField, default=list)
   - requires_acknowledgment (BooleanField, default=False)
   - status (CharField: DRAFT, ACTIVE, SUPERSEDED, ARCHIVED)

3. DocumentAcknowledgment:
   - document (FK to Document)
   - user (FK to User)
   - acknowledged_at (DateTimeField)

4. RFI (Request for Information, extends TenantModel):
   - project (FK to Project)
   - rfi_number (IntegerField — auto-increment per project)
   - subject (CharField)
   - question (TextField)
   - answer (TextField, nullable)
   - status (CharField: DRAFT, OPEN, ANSWERED, CLOSED)

- priority (CharField: URGENT, HIGH, NORMAL, LOW)
- requested_by (FK to User)
- assigned_to (FK to User, nullable — e.g., architect)
- cost_impact (DecimalField, nullable)
- schedule_impact_days (IntegerField, nullable)
- due_date (DateField)
- answered_at (DateTimeField, nullable)
- attachments (reverse FK from Document via generic relation)
- distribution_list (M2M to User)

5. Submittal (extends TenantModel):
- project (FK to Project)
- submittal_number (IntegerField)
- title (CharField)
- spec_section (CharField — specification reference)
- status (CharField: DRAFT, SUBMITTED, APPROVED, APPROVED_AS_NOTED, REVISE_RESUBMIT, REJECTED)
- submitted_by (FK to User)
- reviewer (FK to User, nullable)
- due_date (DateField)
- reviewed_at (DateTimeField, nullable)
- review_notes (TextField)
- documents (M2M to Document)

6. Photo (extends TenantModel):
- project (FK to Project)
- file_key (CharField — S3 key)
- thumbnail_key (CharField — S3 key for thumbnail)
- file_name, file_size, content_type
- caption (CharField, nullable)
- taken_at (DateTimeField — from EXIF or upload time)
- latitude, longitude (DecimalField, nullable — from EXIF GPS)
- phase (CharField, nullable — project phase when taken)
- category (CharField: PROGRESS, BEFORE, AFTER, DEFICIENCY, DELIVERY, INSPECTION, SAFETY, OTHER)
- ai_tags (JSONField, default=list — auto-generated categories)
- uploaded_by (FK to User)
- is_client_visible (BooleanField, default=True)
- linked_daily_log (FK to 'field_ops.DailyLog', nullable)
- annotations (JSONField, nullable — markup data: arrows, text, highlights)

7. PhotoAlbum (extends TenantModel):
- project (FK to Project)
- name (CharField)
- description (TextField)

- cover_photo (FK to Photo, nullable)
- is_auto_generated (BooleanField — AI-created albums)
- photos (M2M to Photo)

SERVICES:
- FileUploadService: Generate presigned S3 upload URLs for direct browser-to-S3 uploads. Validate file types and sizes. On upload complete callback: create Document/Photo record, generate thumbnail for images, extract EXIF data for photos.
- VersionControlService: Upload new version of document — set previous as superseded, increment version number, maintain chain via previous_version FK.
- RFIService: Create RFI with auto-numbering, route to assigned reviewer, track response times, distribute answers to affected parties, log as activity.
- PhotoProcessingService: On photo upload — extract EXIF metadata (GPS, timestamp), generate thumbnail (400px wide), optionally call AI categorization (stub for now — return based on project phase and basic image analysis).

SERIALIZERS: Full CRUD for all models. DocumentSerializer includes presigned download URL. PhotoSerializer includes thumbnail URL. Bulk upload serializers for photos.

VIEWS:
- DocumentFolderViewSet: CRUD with nested document listing
- DocumentViewSet: CRUD plus:
  - @action upload_url (POST): Get presigned S3 upload URL
  - @action upload_complete (POST): Confirm upload and create record
  - @action new_version (POST): Upload new version
  - @action download (GET): Generate presigned download URL
  - @action acknowledge (POST): Record user acknowledgment
  - @action versions (GET): List all versions of a document
- RFIViewSet: CRUD plus @action answer, @action close, @action distribute
- SubmittalViewSet: CRUD plus @action review (approve/reject/revise)
- PhotoViewSet: CRUD with bulk upload plus:
  - @action bulk_upload_urls (POST): Get multiple presigned URLs
  - @action bulk_create (POST): Create multiple photo records
  - @action annotate (PATCH): Save markup annotations
- PhotoAlbumViewSet: CRUD with photo management

TASKS:
- generate_thumbnails(photo_ids): Generate thumbnails for uploaded photos
- check_rfi_due_dates(): Daily — create action items for overdue RFIs
- check_document_expirations(): Daily — alert on expiring permits, insurance certs, contracts

Write tests for presigned URL generation, version control chain integrity, RFI workflow, and bulk photo upload.

PHASE 3 — OPERATIONS EXCELLENCE (Sections 10–13)

Section 10: Scheduling & Resource Management

Claude Code Prompt

Build the Scheduling & Resource Management module in apps/scheduling/. Requires HasModuleAccess('SCHEDULING').

MODELS:

1. Task (extends TenantModel):
   - project (FK to Project)
   - name, description
   - parent_task (FK to self, nullable — for subtasks/WBS)
   - task_type (CharField: TASK, MILESTONE, PHASE, SUMMARY)
   - status (CharField: NOT_STARTED, IN_PROGRESS, COMPLETED, ON_HOLD, CANCELED)
   - start_date, end_date (DateField)
   - actual_start, actual_end (DateField, nullable)
   - duration_days (IntegerField)
   - completion_percentage (IntegerField, 0-100)
   - assigned_crew (FK to Crew, nullable)
   - assigned_users (M2M to User)
   - cost_code (FK to 'estimating.CostCode', nullable)
   - estimated_hours (DecimalField, nullable)
   - actual_hours (DecimalField, default=0)
   - is_critical_path (BooleanField, default=False)
   - sort_order, wbs_code (CharField — work breakdown structure code)
   - color (CharField, nullable — for Gantt display)

2. TaskDependency:
   - predecessor (FK to Task)
   - successor (FK to Task)
   - dependency_type (CharField: FINISH_TO_START, START_TO_START, FINISH_TO_FINISH, START_TO_FINISH)
   - lag_days (IntegerField, default=0)

3. Crew (extends TenantModel):
   - name (CharField)
   - trade (CharField: GENERAL, FRAMING, ELECTRICAL, PLUMBING, HVAC, PAINTING, FLOORING, ROOFING, CONCRETE, DRYWALL, FINISH_CARPENTRY, OTHER)
   - foreman (FK to User, nullable)
   - members (M2M to User)
   - hourly_rate (DecimalField — crew loaded rate)

4. Equipment (extends TenantModel):
   - name, description
   - equipment_type (CharField)
   - serial_number (CharField)

- status (CharField: AVAILABLE, IN_USE, MAINTENANCE, RETIRED)
- current_project (FK to Project, nullable)
- purchase_date (DateField, nullable)
- purchase_cost (DecimalField, nullable)
- depreciation_method (CharField: STRAIGHT_LINE, DECLINING_BALANCE)
- useful_life_years (IntegerField)
- salvage_value (DecimalField)
- current_book_value (DecimalField)
- daily_rental_rate (DecimalField — for internal costing)
- next_maintenance (DateField, nullable)

SERVICES:
- CriticalPathService: Calculate critical path using forward/backward pass algorithm. Identify float for each task. Highlight critical path tasks.
- ScheduleConflictService: Detect crew over-allocation across projects, equipment double-booking, and resource capacity issues.
- GanttDataService: Return structured Gantt chart data with tasks, dependencies, milestones, baseline comparisons, and crew allocation heat map.

SERIALIZERS & VIEWS: Full CRUD for all models. GanttChartView returns complete Gantt data structure. CrewAvailabilityView shows crew allocation across projects by date range. EquipmentViewSet includes depreciation calculation endpoint.

TASKS:
- recalculate_critical_paths(): Hourly — update critical path flags for all active projects
- check_schedule_conflicts(): Daily — scan for resource conflicts, create action items
- calculate_equipment_depreciation(): Monthly — update book values

Write tests for critical path calculation, dependency chain validation, and conflict detection.

Section 11: Financial Management Suite

Claude Code Prompt

Build the Financial Management Suite in apps/financials/. Requires HasModuleAccess('FINANCIALS'). This is the most data-intensive module — covers job costing, accounting, invoicing, change orders, purchase orders, and QuickBooks sync.

MODELS:

1. Budget (extends TenantModel):
   - project (FK to Project)
   - cost_code (FK to CostCode)
   - description (CharField)
   - budgeted_amount (DecimalField)
   - committed_amount (DecimalField, default=0 — from POs/subcontracts)
   - actual_amount (DecimalField, default=0)
   - variance (DecimalField — budgeted - actual, auto-calculated)
   - variance_percentage (DecimalField)

2. Expense (extends TenantModel):
   - project (FK to Project)
   - budget (FK to Budget, nullable)
   - cost_code (FK to CostCode, nullable)
   - expense_type (CharField: MATERIAL, LABOR, EQUIPMENT, SUBCONTRACTOR, OVERHEAD, OTHER)
   - description (CharField)
   - amount (DecimalField)
   - vendor (FK to 'crm.Company', nullable)
   - receipt_file_key (CharField, nullable — S3)
   - date (DateField)
   - entered_by (FK to User)
   - approved_by (FK to User, nullable)
   - status (CharField: PENDING, APPROVED, REJECTED, PAID)
   - payment_method (CharField: CHECK, CREDIT_CARD, ACH, CASH)

3. Invoice (extends TenantModel):
   - project (FK to Project)
   - client (FK to 'crm.Contact')
   - invoice_number (CharField, unique per org)
   - invoice_type (CharField: STANDARD, PROGRESS, AIA_G702, DEPOSIT, RETAINAGE_RELEASE, CHANGE_ORDER)
   - status (CharField: DRAFT, SENT, VIEWED, PARTIALLY_PAID, PAID, OVERDUE, VOID)
   - subtotal, tax_amount, retainage_amount, total, amount_paid, balance_due (DecimalField)
   - issue_date (DateField)
   - due_date (DateField)
   - paid_date (DateField, nullable)
   - payment_terms (CharField: DUE_ON_RECEIPT, NET_15, NET_30, NET_45)
   - notes, internal_notes (TextField)

- public_token (UUIDField — for client payment link)
- stripe_payment_intent_id (CharField, nullable)
- sent_at (DateTimeField, nullable)
- viewed_at (DateTimeField, nullable)

4. InvoiceLineItem:
- invoice (FK to Invoice)
- description, quantity, unit, unit_price, total (standard line item fields)
- cost_code (FK to CostCode, nullable)

5. Payment (extends TenantModel):
- invoice (FK to Invoice)
- amount (DecimalField)
- payment_method (CharField: CREDIT_CARD, ACH, CHECK, CASH, WIRE)
- reference_number (CharField)
- stripe_payment_id (CharField, nullable)
- received_date (DateField)
- notes (TextField)

6. ChangeOrder (extends TenantModel):
- project (FK to Project)
- co_number (IntegerField, per project)
- title, description
- reason (CharField: CLIENT_REQUEST, UNFORESEEN_CONDITIONS, DESIGN_CHANGE, CODE_REQUIREMENT, VALUE_ENGINEERING)
- status (CharField: DRAFT, SUBMITTED, APPROVED, REJECTED, VOID)
- cost_impact (DecimalField — positive = increase)
- schedule_impact_days (IntegerField)
- line_items (reverse FK from ChangeOrderLineItem)
- requested_by (FK to User)
- approved_by (FK to User, nullable)
- client_approval (FK to 'clients.ClientApproval', nullable)

7. PurchaseOrder (extends TenantModel):
- project (FK to Project)
- po_number (CharField)
- vendor (FK to 'crm.Company')
- status (CharField: DRAFT, SENT, ACKNOWLEDGED, PARTIALLY_RECEIVED, RECEIVED, CANCELED)
- subtotal, tax, total (DecimalField)
- delivery_date (DateField, nullable)
- line_items (reverse FK)
- notes (TextField)

8. PurchaseOrderLineItem:

- purchase_order (FK to PO)
- description, quantity, unit, unit_price, total
- received_quantity (DecimalField, default=0)
- cost_code (FK to CostCode, nullable)

SERVICES:
- JobCostingService: Calculate real-time cost status for a project — group by cost code, compare budget vs committed vs actual, calculate variance and earned value metrics.
- InvoicingService: Generate invoices from milestones or manual entry, send to client with payment link, track payments, handle retainage, calculate aging.
- ChangeOrderService: Create CO, calculate cost + schedule impact, route for internal and client approval, update project budget and schedule on approval.
- PurchaseOrderService: Create POs from estimate line items, track delivery status, three-way match (PO + receipt + vendor invoice).
- QuickBooksSyncService (stub for Phase 4): Define interface for bidirectional sync — customers, invoices, payments, bills, chart of accounts.

SERIALIZERS & VIEWS: Full CRUD for all models with appropriate permission levels (ACCOUNTANT role can access all financial views). Include:
- BudgetViewSet with @action variance_report
- ExpenseViewSet with receipt upload support
- InvoiceViewSet with @action send, @action record_payment, @action generate_pdf, @action client_pay (Stripe checkout)
- ChangeOrderViewSet with @action submit, @action approve, @action generate_client_approval
- PurchaseOrderViewSet with @action send_to_vendor, @action receive_items, @action three_way_match
- JobCostReportView: Comprehensive job cost report for a project
- CashFlowForecastView: 30/60/90 day cash flow projection

TASKS:
- check_overdue_invoices(): Daily — mark overdue, send reminders, create action items
- calculate_budget_variances(): Hourly — update variance calculations
- generate_aging_report(): Weekly — compile A/R aging summary

Write tests for job cost calculations, invoice payment flow, change order budget impact, and three-way match validation.

Section 12: Field Operations Hub

Claude Code Prompt

Build the Field Operations Hub in apps/field_ops/. Requires HasModuleAccess('FIELD_OPS'). Covers daily logs, time tracking, expense capture, and mobile time clock.

MODELS:

1. DailyLog (extends TenantModel):
   - project (FK to Project)
   - log_date (DateField)
   - submitted_by (FK to User)
   - status (CharField: DRAFT, SUBMITTED, APPROVED)
   - weather_conditions (JSONField: {temp_high, temp_low, conditions, wind, precipitation})
   - work_performed (TextField)
   - issues_encountered (TextField, nullable)
   - delays (TextField, nullable)
   - delay_reason (CharField: WEATHER, MATERIAL, LABOR, INSPECTION, CLIENT, PERMIT, EQUIPMENT, NONE)
   - visitors (JSONField, default=list — [{name, company, time_in, time_out, purpose}])
   - material_deliveries (JSONField, default=list — [{description, vendor, quantity, condition}])
   - safety_incidents (BooleanField, default=False)
   - photos (M2M to 'documents.Photo')
   - approved_by (FK to User, nullable)
   - approved_at (DateTimeField, nullable)
   - unique_together = [project, log_date] — one log per project per day

2. DailyLogCrewEntry:
   - daily_log (FK to DailyLog)
   - crew_or_trade (CharField)
   - worker_count (IntegerField)
   - hours_worked (DecimalField)
   - work_description (TextField)

3. TimeEntry (extends TenantModel):
   - user (FK to User)
   - project (FK to Project)
   - cost_code (FK to CostCode, nullable)
   - date (DateField)
   - clock_in (DateTimeField)
   - clock_out (DateTimeField, nullable)
   - hours (DecimalField — calculated or manual)
   - entry_type (CharField: CLOCK, MANUAL)
   - status (CharField: PENDING, APPROVED, REJECTED)
   - gps_clock_in (JSONField, nullable — {lat, lng, accuracy})
   - gps_clock_out (JSONField, nullable)

- is_within_geofence (BooleanField, nullable)
- overtime_hours (DecimalField, default=0)
- notes (TextField, nullable)
- approved_by (FK to User, nullable)

4. ExpenseEntry (extends TenantModel):
- user (FK to User)
- project (FK to Project)
- cost_code (FK to CostCode, nullable)
- date (DateField)
- category (CharField: MATERIAL, FUEL, MEALS, TOOLS, EQUIPMENT_RENTAL, SUPPLIES, MILEAGE, OTHER)
- description (CharField)
- amount (DecimalField)
- receipt_file_key (CharField, nullable)
- status (CharField: PENDING, APPROVED, REJECTED, REIMBURSED)
- mileage (DecimalField, nullable — if category is MILEAGE)
- approved_by (FK to User, nullable)

SERVICES:
- TimeClockService: Handle clock in/out with GPS validation. Calculate hours, detect overtime (configurable rules: weekly 40hr, daily 8hr, California-style). Validate against geofence if configured.
- DailyLogService: Create/update logs, auto-populate weather data (stub), link photos and time entries for the day. Approval workflow.
- BulkApprovalService: Batch approve time entries and expenses for a project/date range by supervisor.

SERIALIZERS & VIEWS:
- DailyLogViewSet: CRUD plus @action submit, @action approve, @action add_crew_entry, @action attach_photos
- TimeEntryViewSet: CRUD plus @action clock_in, @action clock_out, @action bulk_approve. Filter by user, project, date range, status.
- ExpenseEntryViewSet: CRUD with receipt upload plus @action approve, @action bulk_approve
- TimesheetSummaryView: Aggregate time entries by user/project/week with overtime calculations
- DailyLogCalendarView: Return log existence/status by date for a project (calendar view)

TASKS:
- auto_clock_out(): Nightly — auto clock-out entries over 14 hours with flag for review
- reminder_daily_log(): Every day at 4pm — remind superintendents of projects without daily logs
- calculate_overtime(): Nightly — recalculate overtime based on weekly totals

Write tests for overtime calculation, clock in/out flow, GPS geofence validation, and bulk approval.

Section 13: Quality & Safety Compliance

Claude Code Prompt

Build Quality & Safety Compliance in apps/quality_safety/. Requires HasModuleAccess('QUALITY_SAFETY').

MODELS:

1. InspectionChecklist (extends TenantModel — template):
   - name (CharField, e.g., "Rough Framing Inspection")
   - checklist_type (CharField: QUALITY, SAFETY, REGULATORY)
   - category (CharField: FRAMING, ELECTRICAL, PLUMBING, HVAC, ROOFING, CONCRETE, FINAL, SAFETY_DAILY, SAFETY_WEEKLY, OSHA)
   - items (reverse FK from ChecklistItem)
   - is_template (BooleanField, default=True)

2. ChecklistItem:
   - checklist (FK to InspectionChecklist)
   - description (CharField)
   - is_required (BooleanField, default=True)
   - sort_order (IntegerField)

3. Inspection (extends TenantModel):
   - project (FK to Project)
   - checklist (FK to InspectionChecklist)
   - inspector (FK to User)
   - inspection_date (DateField)
   - status (CharField: SCHEDULED, IN_PROGRESS, PASSED, FAILED, CONDITIONAL)
   - overall_score (IntegerField, 0-100, nullable)
   - notes (TextField)
   - photos (M2M to Photo)
   - results (reverse FK from InspectionResult)

4. InspectionResult:
   - inspection (FK to Inspection)
   - checklist_item (FK to ChecklistItem)
   - status (CharField: PASS, FAIL, NA, NOT_INSPECTED)
   - notes (TextField)
   - photo (FK to Photo, nullable)

5. Deficiency (extends TenantModel):
   - project (FK to Project)
   - inspection (FK to Inspection, nullable)
   - title, description
   - severity (CharField: CRITICAL, MAJOR, MINOR, COSMETIC)
   - status (CharField: OPEN, IN_PROGRESS, RESOLVED, VERIFIED)
   - assigned_to (FK to User)

- due_date (DateField)
- resolved_date (DateField, nullable)
- photos (M2M to Photo)
- resolution_notes (TextField)

6. SafetyIncident (extends TenantModel):
- project (FK to Project)
- incident_date (DateTimeField)
- incident_type (CharField: INJURY, NEAR_MISS, PROPERTY_DAMAGE, ENVIRONMENTAL, FIRE, FALL, STRUCK_BY, CAUGHT_IN, ELECTRICAL)
- severity (CharField: FIRST_AID, MEDICAL, LOST_TIME, FATALITY)
- description (TextField)
- reported_by (FK to User)
- witnesses (JSONField, default=list)
- injured_person_name (CharField, nullable)
- root_cause (TextField, nullable)
- corrective_actions (TextField, nullable)
- osha_reportable (BooleanField, default=False)
- photos (M2M to Photo)
- status (CharField: REPORTED, INVESTIGATING, CORRECTIVE_ACTION, CLOSED)

7. ToolboxTalk (extends TenantModel):
- project (FK to Project)
- topic (CharField)
- content (TextField)
- presented_by (FK to User)
- presented_date (DateField)
- attendees (M2M to User)
- sign_in_sheet (ImageField, nullable)

SERIALIZERS & VIEWS: Full CRUD for all models. InspectionViewSet with @action record_results (bulk update results), @action generate_report. DeficiencyViewSet with @action resolve, @action verify. SafetyIncidentViewSet with full workflow. Analytics endpoints for quality scores over time, safety incident trends, and deficiency resolution rates.

Seed common inspection checklists (framing, electrical, plumbing, HVAC, roofing, final, daily safety, weekly safety) as templates on org creation.

Write tests for inspection scoring, deficiency lifecycle, and safety incident reporting flow.

PHASE 4 — ENTERPRISE & SPECIALTY (Sections 14–18)

Section 14: Payroll & Workforce Management

Claude Code Prompt

Build Payroll & Workforce Management in apps/payroll/. Requires HasModuleAccess('PAYROLL') — Enterprise tier only.

MODELS:

1. Employee (extends TenantModel):
   - user (OneToOneField to User, nullable — may not have app login)
   - employee_id (CharField, unique per org)
   - employment_type (CharField: W2_FULL_TIME, W2_PART_TIME, 1099_CONTRACTOR)
   - trade (CharField — same trade choices as Crew)
   - hire_date (DateField)
   - termination_date (DateField, nullable)
   - base_hourly_rate (DecimalField)
   - overtime_rate_multiplier (DecimalField, default=1.5)
   - burden_rate (DecimalField — benefits, insurance, taxes as % of base)
   - ssn_last_four (CharField, max 4 — for display only, never store full SSN)
   - direct_deposit_accounts (JSONField, encrypted — routing/account info)
   - tax_filing_status (CharField)
   - federal_allowances, state_allowances (IntegerField)
   - certifications (JSONField — [{name, number, expiry, issuing_body}])
   - emergency_contact (JSONField)

2. PayrollRun (extends TenantModel):
   - pay_period_start, pay_period_end (DateField)
   - run_date (DateField)
   - status (CharField: DRAFT, PROCESSING, APPROVED, PAID, VOID)
   - total_gross, total_taxes, total_deductions, total_net (DecimalField)
   - check_date (DateField)
   - approved_by (FK to User, nullable)

3. PayrollEntry:
   - payroll_run (FK to PayrollRun)
   - employee (FK to Employee)
   - regular_hours, overtime_hours, double_time_hours (DecimalField)
   - gross_pay, federal_tax, state_tax, fica, medicare (DecimalField)
   - deductions (JSONField — [{type, amount}])
   - net_pay (DecimalField)
   - job_cost_allocations (JSONField — [{project_id, cost_code, hours, amount}])

4. CertifiedPayrollReport (extends TenantModel):
   - project (FK to Project)
   - payroll_run (FK to PayrollRun)
   - report_type (CharField: WH_347, STATE_SPECIFIC)

- week_ending (DateField)
- status (CharField: DRAFT, SUBMITTED, ACCEPTED)
- generated_file_key (CharField, nullable — S3)

5. PrevailingWageRate (extends TenantModel):
- project (FK to Project)
- trade (CharField)
- base_rate (DecimalField)
- fringe_rate (DecimalField)
- total_rate (DecimalField)
- effective_date (DateField)

SERVICES:
- PayrollCalculationService: Calculate gross pay from time entries with multi-rate support, apply tax calculations (use stub for actual tax tables — note this would integrate with tax service in production), calculate deductions, allocate labor costs to projects/cost codes from time entries.
- CertifiedPayrollService: Generate WH-347 reports from payroll data. Validate prevailing wage compliance. Flag underpayments.
- WorkforceService: Track certifications with expiry alerts, onboarding checklists, skills inventory.

SERIALIZERS & VIEWS: Full CRUD. PayrollRunViewSet with @action calculate (run calculations), @action approve, @action generate_checks, @action export_ach. CertifiedPayrollViewSet with @action generate_wh347. EmployeeViewSet with @action certifications (manage certs with expiry tracking). ComplianceDashboardView showing certification status, prevailing wage compliance, apprentice ratios.

TASKS:
- check_certification_expirations(): Daily — alert on certs expiring within 30 days
- prevailing_wage_compliance_check(): Weekly — verify all prevailing wage projects are compliant

Write tests for payroll calculation accuracy, overtime rules, and certified payroll generation.

Section 15: Service & Warranty Management

Claude Code Prompt

Build Service & Warranty Management in apps/service/. Requires HasModuleAccess('SERVICE_WARRANTY') — Enterprise tier only.

MODELS:

1. ServiceTicket (extends TenantModel):
   - ticket_number (CharField, auto-generated per org)
   - project (FK to Project, nullable — may be standalone)
   - client (FK to 'crm.Contact')
   - title, description
   - priority (CharField: EMERGENCY, HIGH, NORMAL, LOW)
   - status (CharField: NEW, ASSIGNED, IN_PROGRESS, ON_HOLD, COMPLETED, CLOSED)
   - ticket_type (CharField: WARRANTY, SERVICE_CALL, MAINTENANCE, CALLBACK, EMERGENCY)
   - assigned_to (FK to User, nullable)
   - scheduled_date (DateTimeField, nullable)
   - completed_date (DateTimeField, nullable)
   - resolution (TextField)
   - billable (BooleanField, default=True)
   - billing_type (CharField: TIME_AND_MATERIAL, FLAT_RATE, WARRANTY_NO_CHARGE)
   - labor_hours (DecimalField, default=0)
   - parts_cost (DecimalField, default=0)
   - total_cost (DecimalField, default=0)
   - invoice (FK to 'financials.Invoice', nullable)

2. Warranty (extends TenantModel):
   - project (FK to Project)
   - warranty_type (CharField: WORKMANSHIP, MANUFACTURER, EXTENDED)
   - description, coverage_details (TextField)
   - start_date, end_date (DateField)
   - manufacturer (CharField, nullable)
   - product_info (JSONField, nullable)
   - status (CharField: ACTIVE, EXPIRED, CLAIMED)

3. WarrantyClaim (extends TenantModel):
   - warranty (FK to Warranty)
   - service_ticket (FK to ServiceTicket, nullable)
   - description (TextField)
   - status (CharField: FILED, IN_REVIEW, APPROVED, DENIED, RESOLVED)
   - cost (DecimalField, nullable)
   - resolution (TextField, nullable)

4. ServiceAgreement (extends TenantModel):
   - client (FK to Contact)

- name (CharField)
- agreement_type (CharField: MAINTENANCE, INSPECTION, FULL_SERVICE)
- start_date, end_date (DateField)
- billing_frequency (CharField: MONTHLY, QUARTERLY, ANNUAL)
- billing_amount (DecimalField)
- visits_per_year (IntegerField)
- visits_completed (IntegerField, default=0)
- auto_renew (BooleanField, default=True)
- status (CharField: ACTIVE, EXPIRED, CANCELED)

SERIALIZERS & VIEWS: Full CRUD for all models. ServiceTicketViewSet with @action assign, @action complete, @action generate_invoice. WarrantyViewSet with @action file_claim. ServiceAgreementViewSet with @action schedule_visits, @action renew. DispatchBoardView for technician scheduling.

TASKS:
- check_expiring_warranties(): Daily — alert clients of warranties expiring within 60 days
- generate_recurring_invoices(): Monthly — invoice for active service agreements
- schedule_maintenance_visits(): Monthly — auto-create tickets for upcoming maintenance visits

Write tests for warranty claim flow, service agreement billing cycle, and ticket lifecycle.

Section 16: Analytics & Reporting Engine

Claude Code Prompt

Build the Analytics & Reporting Engine in apps/analytics/. This is a Core module — always active for all plans.

MODELS:

1. SavedReport (extends TenantModel):
   - name (CharField)
   - report_type (CharField: FINANCIAL_PL, JOB_COST, CASH_FLOW, AGING, WIP, LEAD_CONVERSION, SALES_PIPELINE, PROJECT_STATUS, RESOURCE_UTILIZATION, SAFETY_SUMMARY, PAYROLL_LABOR, SERVICE_PERFORMANCE, CUSTOM)
   - parameters (JSONField — date ranges, filters, grouping)
   - created_by (FK to User)
   - is_shared (BooleanField)
   - schedule (JSONField, nullable — {frequency, recipients, next_run})

2. DashboardWidget (extends TenantModel):
   - name (CharField)
   - widget_type (CharField: CHART_BAR, CHART_LINE, CHART_PIE, CHART_DONUT, KPI_CARD, TABLE, GAUGE, HEATMAP)
   - data_source (CharField — which report type/query feeds this)
   - config (JSONField — chart options, colors, dimensions)
   - refresh_interval_minutes (IntegerField, default=60)

3. ReportExecution (audit log):
   - saved_report (FK to SavedReport)
   - executed_at (DateTimeField)
   - parameters_used (JSONField)
   - row_count (IntegerField)
   - execution_time_ms (IntegerField)
   - file_key (CharField, nullable — cached export)

SERVICES:
- ReportEngine: Centralized query builder that accepts report_type + parameters and returns structured data. Handles: P&L by project/client/period, job cost with earned value, cash flow projections, A/R aging, WIP reports, lead funnel analytics, resource utilization heatmaps, safety trends, payroll summaries, service metrics.
- ExportService: Export any report to Excel (.xlsx with formatting), PDF (with org branding), or CSV.
- ScheduledReportService: Generate reports on schedule and email to configured recipients.

VIEWS:
- ReportViewSet: CRUD for saved reports plus @action execute (run report, return data), @action export (Excel/PDF/CSV download), @action schedule (configure automatic execution)
- DashboardWidgetViewSet: CRUD for custom dashboard widgets
- QuickReportView: Execute ad-hoc reports without saving — pass report_type + parameters, get data back
- KPIDashboardView: Return pre-calculated KPI cards — active projects, revenue MTD/YTD, margin %, overdue

invoices, lead conversion rate, safety incident count

TASKS:
- run_scheduled_reports(): Hourly — check for reports due, execute and email
- precalculate_kpis(organization_id): Every 30 min — update cached KPI values
- cleanup_old_exports(): Weekly — delete report exports older than 30 days

Write tests for report accuracy across different date ranges and filters. Test Excel export formatting.

Section 17: Integration Ecosystem & Open API

Claude Code Prompt

Build the Integration Ecosystem in a new apps/integrations/ app. This handles QuickBooks sync, Stripe webhooks (extend billing app), weather API, Google/Microsoft calendar sync, and the public REST API.

MODELS:

1. IntegrationConnection (extends TenantModel):
   - integration_type (CharField: QUICKBOOKS_ONLINE, QUICKBOOKS_DESKTOP, XERO, GOOGLE_WORKSPACE, MICROSOFT_365, STRIPE_CONNECT)
   - status (CharField: CONNECTED, DISCONNECTED, ERROR, SYNCING)
   - access_token_encrypted (TextField)
   - refresh_token_encrypted (TextField)
   - token_expires_at (DateTimeField, nullable)
   - last_sync_at (DateTimeField, nullable)
   - last_error (TextField, nullable)
   - sync_config (JSONField — what to sync, direction, frequency)
   - connected_by (FK to User)

2. SyncLog (extends TenantModel):
   - connection (FK to IntegrationConnection)
   - sync_type (CharField: FULL, INCREMENTAL, MANUAL)
   - status (CharField: STARTED, COMPLETED, FAILED, PARTIAL)
   - records_synced, records_failed (IntegerField)
   - started_at, completed_at (DateTimeField)
   - error_details (JSONField, nullable)

3. WebhookEndpoint (extends TenantModel):
   - url (URLField)
   - events (JSONField — list of event types to send)
   - secret (CharField — for HMAC signature)
   - is_active (BooleanField)
   - last_triggered_at (DateTimeField, nullable)
   - failure_count (IntegerField, default=0)

4. APIKey (extends TenantModel):
   - name (CharField)
   - key_prefix (CharField, max 8 — visible prefix for identification)
   - key_hash (CharField — hashed API key, never store plaintext)
   - scopes (JSONField — list of allowed scopes: read:projects, write:projects, etc.)
   - rate_limit_per_hour (IntegerField, default=1000)
   - last_used_at (DateTimeField, nullable)
   - expires_at (DateTimeField, nullable)
   - created_by (FK to User)

SERVICES:
- QuickBooksSyncService: OAuth 2.0 connect flow. Bidirectional sync: customers <-> contacts, invoices, payments, bills <-> expenses, chart of accounts <-> cost codes. Conflict resolution (last-write-wins with manual review for conflicts). Runs every 4 hours via Celery Beat.
- WeatherService: Fetch 5-day forecast for project job sites using OpenWeatherMap or WeatherAPI. Cache results for 3 hours. Auto-create weather alerts when conditions impact outdoor work.
- WebhookDispatchService: On key events (project status change, invoice paid, etc.), send signed webhook payloads to registered endpoints. Retry with exponential backoff.
- APIKeyAuthBackend: Custom DRF authentication that validates hashed API keys, checks scopes, enforces rate limits.

VIEWS:
- IntegrationConnectionViewSet: List available integrations, initiate OAuth connect, disconnect, force sync, view sync logs
- QuickBooksOAuthView: Handle OAuth callback and token storage
- WebhookEndpointViewSet: CRUD for webhook registrations (Enterprise only)
- APIKeyViewSet: Create (returns key once), list (prefix only), revoke
- Public API Views: Expose all major resources (projects, contacts, estimates, invoices) under /api/v1/public/ with API key auth and scope checking. Full OpenAPI docs via drf-spectacular.

TASKS:
- sync_quickbooks(organization_id): Scheduled every 4 hours — run incremental sync
- refresh_oauth_tokens(): Every 30 min — refresh tokens expiring within 1 hour
- fetch_weather_forecasts(): Every 3 hours — update forecasts for all active job sites
- dispatch_webhooks(event_type, payload, organization_id): Async — fan out to all registered endpoints

Write tests for OAuth flow, sync conflict resolution, API key authentication, rate limiting, and webhook signature verification.

Section 18: Mobile / PWA Experience

Claude Code Prompt

Build Progressive Web App (PWA) support for Builders Stream Pro's React frontend. This makes the app installable on mobile devices with offline capability for critical field operations.

This is a FRONTEND-FOCUSED section. Create the following in the frontend/ directory:

1. PWA Configuration:
   - service-worker.js: Cache critical assets (app shell, fonts, icons). Implement stale-while-revalidate for API responses. Queue failed POST/PATCH requests for retry when online (background sync).
   - manifest.json: App name "Builders Stream Pro", theme colors matching brand, icons at 72/96/128/192/384/512px, display: standalone, start_url: /dashboard, orientation: portrait.

2. Offline Data Layer:
   - Create an IndexedDB wrapper (use idb library) that caches:
     - Active projects list
     - Current day's daily log draft
     - Time entry draft (clock in/out)
     - Expense entry drafts with receipt photos (as blobs)
     - Recent photos queue for upload
   - SyncManager service: On connectivity restore, sync queued items to API in order. Handle conflicts (server data newer). Show sync status indicator in UI.

3. Mobile-Optimized Components:
   - MobileNavigation: Bottom tab bar with 5 quick actions — Dashboard, Clock In/Out, Daily Log, Camera (photo capture), Messages
   - QuickClockInOut: Large touch-friendly button with GPS capture. Works offline — stores entry locally.
   - MobileDailyLog: Simplified form optimized for one-thumb use. Voice-to-text for notes (Web Speech API). Photo attachment from camera.
   - MobilePhotoCapture: Camera integration, auto-tag with current project, queue for upload.
   - MobileExpenseCapture: Photo receipt capture, amount entry, project/cost code selection.
   - PushNotificationManager: Request permission, register service worker for push. Handle notification types: approval_needed, schedule_change, message_received, payment_received.

4. Responsive Layout System:
   - useBreakpoint() hook: Detect mobile/tablet/desktop
   - MobileLayout wrapper: Bottom nav, no sidebar, stacked cards
   - TabletLayout wrapper: Collapsible sidebar, two-column where appropriate
   - DesktopLayout wrapper: Full sidebar, multi-column dashboard

5. Geofencing:
   - GeofenceService: Use Geolocation API to check if user is within configured radius of job site when clocking in. Cache job site coordinates locally.

IMPORTANT: All mobile components should work as part of the same React app — NOT a separate app. Use responsive

design and feature detection to adapt the single codebase for mobile use. The PWA approach avoids the need for separate iOS/Android development.

Create a comprehensive testing plan for offline scenarios: clock in while offline -> go online -> verify sync, daily log with photos while offline -> verify queued upload, expense with receipt while offline -> verify sync order.

APPENDIX A: Frontend Shell Setup Prompt

> Run this after Section 1 to set up the React frontend base.

Set up the React 18 frontend for Builders Stream Pro in the frontend/ directory.

Tech: Vite + React 18 + TypeScript + TailwindCSS + React Router v6 + TanStack Query (React Query) + Zustand (state management) + Axios

Structure:

```
frontend/
├── public/
├── src/
│   ├── api/           # Axios instance, API service functions per module
│   ├── components/    # Shared UI components (Button, Modal, Table, Form, etc.)
│   ├── features/      # Feature modules matching Django apps
│   │   ├── auth/      # Login, Register, ForgotPassword pages + hooks
│   │   ├── dashboard/ # Main dashboard, widgets, layout manager
│   │   ├── projects/  # Project list, detail, lifecycle, team
│   │   ├── crm/       # Contacts, leads, pipeline board (kanban)
│   │   ├── estimating/# Estimates, proposals, cost database
│   │   ├── scheduling/# Gantt chart, crew management
│   │   ├── financials/# Invoices, expenses, budgets, reports
│   │   ├── clients/   # Client portal (separate route tree)
│   │   ├── documents/ # File manager, RFIs, photos
│   │   ├── field_ops/ # Daily logs, time clock, expenses
│   │   ├── quality/   # Inspections, safety, checklists
│   │   ├── payroll/   # Payroll runs, employees
│   │   ├── service/   # Service tickets, warranties
│   │   ├── analytics/ # Reports, custom dashboards
│   │   ├── settings/  # Org settings, billing, integrations, users
│   │   └── admin/     # Super admin (if applicable)
│   ├── hooks/         # Custom hooks (useAuth, useTenant, useModuleAccess, etc.)
│   ├── layouts/       # AppLayout, AuthLayout, ClientPortalLayout, MobileLayout
│   ├── stores/        # Zustand stores (auth, tenant, ui preferences)
│   ├── types/         # TypeScript interfaces matching Django serializers
│   ├── utils/         # Formatters, validators, constants
│   ├── App.tsx
│   ├── main.tsx
│   └── router.tsx     # All route definitions with lazy loading
├── tailwind.config.js
├── vite.config.ts
├── tsconfig.json
└── package.json
```

Key setup:
- Axios interceptor: Attach JWT access token + X-Organization-ID header to all requests. Auto-refresh expired tokens.

Redirect to /login on 401.
- TanStack Query: Default staleTime 60s, refetchOnWindowFocus for dashboard data.
- Route guards: ProtectedRoute component checks auth. ModuleRoute component checks module access. RoleRoute checks minimum role.
- Zustand auth store: user, tokens, organization, switchOrganization(), login(), logout()
- useModuleAccess(moduleKey) hook: Returns boolean, gates UI sections
- Lazy load all feature modules with React.lazy + Suspense for code splitting
- TailwindCSS with a custom theme: construction-appropriate colors (navy, orange/amber accents, slate grays), large touch targets for field use (min 44px), clear typography hierarchy

Create the shell with login page, authenticated layout with sidebar navigation, and a placeholder dashboard page. The sidebar should show/hide menu items based on active modules.

APPENDIX B: Development Workflow Reminders

Before starting each section:

1. Ensure previous sections' migrations are applied: `python manage.py migrate`
2. Create the Django app if not exists: `python manage.py startapp <app_name>` then move to `apps/`
3. Register the app in `INSTALLED_APPS` as `apps.<app_name>`
4. Add the app's URLs to `config/urls.py`

After completing each section:

1. Run `python manage.py makemigrations` and `python manage.py migrate`
2. Run tests: `python manage.py test apps.<app_name>`
3. Verify API endpoints in DRF Browsable API or Swagger at `/api/docs/`
4. Commit with descriptive message referencing the section number

Environment Variables (.env):

```
# Database
DATABASE_URL=postgresql://bsp_user:password@localhost:5432/builderstream

# Redis
REDIS_URL=redis://localhost:6379/0

# Django
DJANGO_SECRET_KEY=your-secret-key-here
DJANGO_DEBUG=True
DJANGO_ALLOWED_HOSTS=localhost,127.0.0.1

# AWS S3
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_STORAGE_BUCKET_NAME=builderstream-files
AWS_S3_REGION_NAME=us-east-1

# Stripe
STRIPE_SECRET_KEY=sk_test_...
STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...

# Email
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=smtp.sendgrid.net
EMAIL_PORT=587
SENDGRID_API_KEY=

# OAuth
GOOGLE_OAUTH_CLIENT_ID=
GOOGLE_OAUTH_CLIENT_SECRET=
GITHUB_OAUTH_CLIENT_ID=
GITHUB_OAUTH_CLIENT_SECRET=

# Frontend
FRONTEND_URL=http://localhost:5173
```

End of Builders Stream Pro Development Prompt Guide 18 sections covering 12 modules + infrastructure across 4 development phases Estimated total development: 14–18 months with iterative delivery