

MindFlow Construction Platform - Comprehensive Deep-Dive Analysis

Project: MindFlow Construction Platform

Location: C:\GitHub\ConstructionPlatform

Purpose: Exhaustive analysis of every component, integration point, and potential issue

Budget: \$500 Claude Code credits - USE THEM ALL

Date: November 4, 2025

ANALYSIS OBJECTIVES

This is a **COMPREHENSIVE, NO-HOLDS-BARRED ANALYSIS**. Your mission:

1. **Read EVERY file** in backend/src/, frontend/src/, backend/prisma/, shared/types/
2. **Test EVERY integration point** between frontend and backend
3. **Validate EVERY authentication flow** and security mechanism
4. **Analyze EVERY database model** and relationship
5. **Identify EVERY bug, gap, inconsistency, or potential issue**
6. **Provide SPECIFIC line numbers, file paths, and code snippets**
7. **Give actionable recommendations with time estimates**

DO NOT skimp on details. DO NOT summarize. BE THOROUGH.

PART 1: BACKEND DEEP DIVE (150+ Questions)

SECTION 1.1: Project Structure & Architecture (20 questions)

Q1.1.1: List EVERY file in backend/src/ with its purpose

- Include full file tree with line counts
- Categorize by: routes, controllers, middleware, services, utilities, types
- Identify any orphaned or unused files
- Flag any missing standard files (e.g., no error handler?)

Q1.1.2: What is the exact folder structure convention?

- Is it following a consistent pattern?
- MVC? Service-oriented? Domain-driven?
- Are files organized logically?
- Any violations of the chosen pattern?

Q1.1.3: Analyze backend/package.json dependencies

- List ALL dependencies with versions
- Are any packages outdated or deprecated?
- Any security vulnerabilities (check npm audit)?
- Any unused dependencies that should be removed?
- Any missing dependencies for claimed features?

Q1.1.4: Review backend/tsconfig.json configuration

- Is strict mode enabled?
- What compiler options are set?
- Are paths/aliases configured?
- Any issues with the TypeScript setup?
- Compare to best practices for Express + TypeScript

Q1.1.5: Analyze the entry point (backend/src/index.ts or server.ts)

- How is the Express app initialized?
- What middleware is registered and in what order?
- Is there proper error handling?
- How are routes mounted?
- Any startup checks (database connection, etc.)?
- Graceful shutdown handling?

Q1.1.6: What environment variables are required?

- List EVERY variable from .env.example
- Which are optional vs. required?
- Any missing variables that code references?
- Any hardcoded values that should be env vars?
- Validation of env vars at startup?

Q1.1.7: Is there a health check endpoint?

- What does it check (database, external services)?
- What's the response format?
- Does it actually work?

Q1.1.8: How is logging implemented?

- What logging library (if any)?
- What log levels are used?
- Are logs structured (JSON)?
- Is sensitive data filtered from logs?
- Any log aggregation planned?

Q1.1.9: How is error handling implemented?

- Is there a global error handler?
- Custom error classes?
- Error response format standardized?
- Stack traces in production (should NOT be)?
- HTTP status codes used correctly?

Q1.1.10: Is there API versioning?

- Routes prefixed with /v1, /api, etc.?
- Strategy for handling breaking changes?

Q1.1.11: What's the CORS configuration?

- Which origins are allowed?
- Is it too permissive for production?
- OPTIONS preflight handled?

Q1.1.12: Rate limiting implemented?

- Which endpoints are rate limited?
- What are the limits?
- Storage mechanism (memory, Redis)?

Q1.1.13: Request validation strategy?

- Zod mentioned in package.json - where's it used?
- Which endpoints validate input?
- Is validation consistent across all routes?

Q1.1.14: Response format standardization?

- Is there a consistent response wrapper?
- Success: { success: true, data: ... }?
- Error: { success: false, error: ... }?
- Or different format?

Q1.1.15: Database connection management?

- How is Prisma Client instantiated?
- Connection pooling configured?
- Disconnect handling on shutdown?

Q1.1.16: Are there any background jobs or scheduled tasks?

- Cron jobs?
- Task queues?
- If so, what do they do?

Q1.1.17: File upload handling?

- Any multer or similar middleware?
- Where are files stored?
- File size limits?
- File type validation?

Q1.1.18: API documentation?

- Swagger/OpenAPI spec?
- JSDoc comments?
- Postman collection?
- README with endpoint list?

Q1.1.19: Code quality tooling?

- ESLint configuration?
- Prettier configuration?
- Pre-commit hooks?
- Any linting errors currently?

Q1.1.20: Testing infrastructure?

- Jest, Mocha, or other test framework?
- Test files present?
- Test coverage measurement?

- Integration test setup?
-

SECTION 1.2: Authentication System (30 questions)

Q1.2.1: Locate ALL authentication-related files

- List every file involved in auth
- Line counts for each
- Purpose of each file

Q1.2.2: JWT implementation analysis

- Which JWT library is used?
- How are tokens generated?
- What claims are included in the token payload?
- Token expiration times (access vs. refresh)?
- Secret management (environment variable)?

Q1.2.3: Access token deep dive

- Exact token payload structure?
- Include code snippet of token generation
- How is token signed?
- Any custom claims beyond user ID?

Q1.2.4: Refresh token deep dive

- Are refresh tokens implemented?
- Where are they stored?
- How do they differ from access tokens?
- Refresh token rotation implemented?
- Revocation mechanism?

Q1.2.5: Password hashing analysis

- Bcrypt as indicated in package.json?
- Salt rounds configured?
- Where is hashing done (registration, password change)?
- Include code snippets

Q1.2.6: Registration endpoint analysis

- Exact route path?
- Request body validation?
- Email uniqueness check?
- Password strength requirements?
- Default user role assigned?
- Welcome email sent?
- Any post-registration hooks?
- Full code review with line numbers

Q1.2.7: Login endpoint analysis

- Exact route path?
- Email + password validation?
- Account status check (isActive)?

- Failed login attempt tracking?
- Account lockout after X failures?
- Response includes what (token, user object)?
- Full code review with line numbers

Q1.2.8: Logout endpoint analysis

- Is there one?
- What does it do (token revocation)?
- Or just client-side (remove token)?

Q1.2.9: Token refresh endpoint analysis

- Is there one?
- How does it validate the refresh token?
- New tokens issued?
- Full code review

Q1.2.10: Password change endpoint analysis

- Is there one?
- Requires current password?
- Re-authentication required?
- All active sessions invalidated?

Q1.2.11: Password reset flow analysis

- Forgot password endpoint?
- Reset token generation and storage?
- Reset email sending?
- Reset token expiration?
- Reset password endpoint?

Q1.2.12: Email verification?

- Verification email sent on registration?
- Verification token system?
- Unverified users restricted?

Q1.2.13: Authentication middleware analysis

- What's it called? Where is it?
- How does it extract the token?
- Bearer scheme required?
- Token validation process?
- Where is user data attached to request?
- Full code review with line numbers

Q1.2.14: Role-based access control (RBAC)

- What roles exist (list from schema)?
- Is there authorization middleware?
- How is it implemented?
- Example: `requireRole(['ADMIN', 'PROJECT_MANAGER'])`?
- Full code review

Q1.2.15: Permission system?

- Beyond roles, are there granular permissions?
- E.g., "canEditCustomer", "canDeleteProject"?

Q1.2.16: User context in requests

- How is current user accessed in controllers?
- req.user attached by middleware?
- What properties available?

Q1.2.17: Multi-tenancy considerations?

- Single tenant or multi-tenant architecture?
- Data isolation between customers/builders?
- Row-level security?

Q1.2.18: Session management?

- Server-side sessions or stateless JWT?
- Session storage if applicable?

Q1.2.19: Account management endpoints

- Get current user profile?
- Update profile?
- Upload avatar?
- Change email?
- Delete account?

Q1.2.20: Admin user management?

- List all users (paginated)?
- Create user as admin?
- Update user role?
- Deactivate user?
- Delete user?

Q1.2.21: Security headers implemented?

- Helmet.js used?
- CSP, HSTS, X-Frame-Options, etc.?

Q1.2.22: XSS protection?

- Input sanitization?
- Output encoding?

Q1.2.23: CSRF protection?

- Relevant for cookie-based auth?
- Implemented if needed?

Q1.2.24: SQL injection protection?

- Prisma parameterized queries (should be safe)?
- Any raw SQL queries?

Q1.2.25: Secrets management?

- JWT_SECRET properly secured?
- Database password not in code?
- Any hardcoded secrets?

Q1.2.26: Authentication error messages?

- Do they leak information?
- "User not found" vs. "Invalid credentials"?

Q1.2.27: Account lockout mechanism?

- After N failed attempts?
- How long locked?
- Unlock mechanism?

Q1.2.28: Two-factor authentication (2FA)?

- Is it implemented or planned?
- Any TODOs in code?

Q1.2.29: OAuth integration?

- Google, GitHub, etc.?
- Or just email/password?

Q1.2.30: Authentication testing coverage?

- Review test files for auth
- What's tested?
- What's NOT tested?
- Test coverage percentage?

SECTION 1.3: Database & Prisma (40 questions)

Q1.3.1: Review backend/prisma/schema.prisma COMPLETELY

- Paste the ENTIRE schema file
- How many models?
- How many fields total across all models?
- How many relations?

Q1.3.2: User model deep dive

- Every field listed with type and constraints
- Relations to other models?
- Indexes defined?
- Any missing fields (e.g., lastLoginAt)?
- Default values?
- Are audit fields present (createdAt, updatedAt)?

Q1.3.3: Customer model analysis

- Every field with purpose
- Relation to User (who manages the customer)?
- Pricing tier structure?
- External system mappings (Richmond, Holt)?

- Soft delete implemented?

Q1.3.4: Plan model analysis

- Every field and purpose
- Plan variants (elevations)?
- Option system structure?
- Version control?

Q1.3.5: Material model analysis

- Every field
- Vendor relationships?
- Pricing structure?
- Commodity pricing integration?

Q1.3.6: Job model analysis

- Every field
- Relationship to plans?
- Relationship to customers?
- Status workflow?

Q1.3.7: Takeoff model analysis

- Every field
- Line items structure?
- Variance tracking fields?

Q1.3.8: TakeoffLineItem model analysis

- Every field
- quantityEstimated vs quantityActual?
- Variance calculation?
- Reason tracking?

Q1.3.9: VariancePattern model analysis

- Every field
- Detection logic?
- Confidence scoring?
- Approval workflow?

Q1.3.10: PurchaseOrder model analysis

- Every field
- Status workflow?
- Relationship to jobs?

Q1.3.11: Check ALL relationships/foreign keys

- Are they properly defined?
- onDelete cascades correct?
- Referential integrity?

Q1.3.12: Review ALL indexes

- Which fields are indexed?
- Composite indexes?
- Any missing indexes for common queries?

Q1.3.13: Check for N+1 query problems

- Review controller code
- Are relations eagerly loaded when needed?
- Use of `include` and `select`?

Q1.3.14: Prisma Client generation status

- Is `.prisma/client` generated?
- Generated files size?
- Any generation errors?

Q1.3.15: Review migration history

- List all migrations
- Any failed migrations?
- Any rolled back migrations?
- Migration naming convention?

Q1.3.16: Is there a seed file?

- `backend/prisma/seed.ts`?
- What data does it create?
- Does it run successfully?
- Paste the seed file contents

Q1.3.17: Data validation at database level

- Check constraints?
- Unique constraints?
- NOT NULL constraints?
- Default values?

Q1.3.18: Enum types analysis

- What enums are defined?
- Are they used consistently?
- Any magic strings that should be enums?

Q1.3.19: DateTime handling

- Timezone consideration?
- Created/updated timestamps automatic?

Q1.3.20: JSON fields usage

- Any `Json` or `JsonArray` fields?
- What data is stored?
- Is it the right approach?

Q1.3.21: Soft deletes implementation?

- `deletedAt` field pattern?

- Scoped queries to exclude deleted?
- Or hard deletes?

Q1.3.22: Audit trail

- AuditLog model exists?
- What actions are logged?
- Who/when/what changed?

Q1.3.23: Database connection pooling

- Pool size configured?
- Connection timeout?

Q1.3.24: Transaction support

- Are transactions used where needed?
- `prisma.$transaction()`?
- Examples of multi-step operations?

Q1.3.25: Database performance

- Query execution times measured?
- Slow query log?
- Database size considerations?

Q1.3.26: Backup strategy

- Automated backups?
- Backup scripts?
- Restore tested?

Q1.3.27: Data migration strategy

- For schema changes?
- Data transformation scripts?

Q1.3.28: Multi-tenancy at DB level?

- Shared schema or separate schemas?
- Row-level security?

Q1.3.29: Database constraints

- Foreign keys enforced?
- Check constraints?
- Unique constraints?

Q1.3.30: Cascading deletes configured?

- What happens when a customer is deleted?
- Orphaned records prevented?

Q1.3.31: Database naming conventions

- Consistent naming?
- CamelCase or snake_case?
- Plural or singular table names?

Q1.3.32: Database documentation

- Comments in schema?
- ER diagram available?

Q1.3.33: Prisma Studio usage

- Can it connect?
- Any errors?
- All models visible?

Q1.3.34: Database reset procedure

- Can database be reset?
- Scripts available?

Q1.3.35: Production database plan

- Where will it be hosted?
- Connection string management?
- SSL required?

Q1.3.36: Database migrations in production

- Automated or manual?
- Downtime required?
- Rollback plan?

Q1.3.37: Data integrity checks

- Any scripts to validate data?
- Consistency checks?

Q1.3.38: Full-text search

- Implemented?
- Which fields?
- Which database features used?

Q1.3.39: Database version

- PostgreSQL version?
- Any version-specific features used?

Q1.3.40: Database permissions

- User permissions configured?
- Principle of least privilege?

SECTION 1.4: API Endpoints Inventory (25 questions)

Q1.4.1: List ALL routes in the application

- Group by resource (auth, customers, plans, etc.)
- Include HTTP method and path for each
- Handler function for each

- Middleware chain for each

Q1.4.2: Authentication routes complete analysis

- POST /auth/register - full review
- POST /auth/login - full review
- POST /auth/logout - full review
- POST /auth/refresh - full review
- GET /auth/me - full review
- POST /auth/change-password - full review
- Any others?

Q1.4.3: Customer routes analysis

- GET /customers - list
- GET /customers/:id - get one
- POST /customers - create
- PUT /customers/:id - update
- DELETE /customers/:id - delete
- For each: request validation? auth required? response format? error handling?

Q1.4.4: Plan routes analysis

- List all plan-related routes
- Complete analysis of each

Q1.4.5: Material routes analysis

- List all material-related routes
- Complete analysis of each

Q1.4.6: Job routes analysis

- List all job-related routes
- Complete analysis of each

Q1.4.7: Takeoff routes analysis

- List all takeoff-related routes
- Complete analysis of each

Q1.4.8: Purchase Order routes analysis

- List all PO-related routes
- Complete analysis of each

Q1.4.9: Variance Pattern routes analysis

- List all variance-related routes
- Complete analysis of each

Q1.4.10: User management routes (admin)

- List all user admin routes
- Complete analysis

Q1.4.11: For EVERY endpoint: Does it validate input?

- Create a table of all endpoints with validation status

Q1.4.12: For EVERY endpoint: Does it require authentication?

- Create a table of all endpoints with auth requirement

Q1.4.13: For EVERY endpoint: What roles can access it?

- Create a table of all endpoints with RBAC

Q1.4.14: For EVERY endpoint: What's the response format?

- Consistent across all?

Q1.4.15: For EVERY endpoint: Error handling implemented?

- Try/catch blocks?
- Proper status codes?

Q1.4.16: Pagination implemented?

- Which list endpoints support it?
- Query params (page, limit, offset)?
- Response includes total count?

Q1.4.17: Filtering implemented?

- Which list endpoints support filters?
- What filters available?

Q1.4.18: Sorting implemented?

- Which list endpoints support sorting?
- Which fields can be sorted?

Q1.4.19: Search implemented?

- Which endpoints support search?
- Full-text or simple LIKE?

Q1.4.20: Bulk operations?

- Bulk create, update, delete?
- Transaction wrapped?

Q1.4.21: File upload endpoints?

- Which endpoints accept files?
- File validation?
- Storage location?

Q1.4.22: Export endpoints?

- CSV, PDF, Excel export?
- For which resources?

Q1.4.23: Import endpoints?

- Bulk import from file?
- Validation of imported data?

Q1.4.24: Webhook endpoints?

- Any webhooks for external integrations?

Q1.4.25: Health/status endpoints?

- /health, /status, /ready, /live?
 - What do they check?
-

SECTION 1.5: Business Logic & Services (25 questions)

Q1.5.1: Is there a service layer?

- Separate from controllers?
- What services exist?

Q1.5.2: CustomerService analysis (if exists)

- What business logic?
- Pricing tier calculations?
- External system sync?

Q1.5.3: PlanService analysis

- Plan validation logic?
- Template management?
- Version control?

Q1.5.4: MaterialService analysis

- Pricing calculations?
- Vendor management?
- Commodity price updates?

Q1.5.5: TakeoffService analysis

- BOM generation from plan template?
- Quantity calculations?
- Variance analysis?

Q1.5.6: PricingService analysis

- Pricing pipeline logic?
- Multi-step calculations?
- Transparent pricing breakdown?

Q1.5.7: VarianceService analysis

- Pattern detection algorithms?
- Confidence scoring?
- Automatic template updates?

Q1.5.8: NotificationService analysis

- Email sending?
- In-app notifications?
- Push notifications?

Q1.5.9: External integrations

- Random Lengths API?
- Any other external APIs?
- Error handling for external calls?

Q1.5.10: Business rule validation

- Where are business rules enforced?
- Examples of rules?

Q1.5.11: Complex workflows

- Multi-step processes?
- State machines?
- Event-driven?

Q1.5.12: Calculation accuracy

- Financial calculations precise?
- Rounding handled correctly?
- Decimal vs. float types?

Q1.5.13: Data transformation

- DTOs used?
- Entity to DTO mapping?

Q1.5.14: Caching strategy

- Any caching implemented?
- Redis? In-memory?
- Cache invalidation?

Q1.5.15: Queue system

- Background jobs?
- Task queue?
- Job retry logic?

Q1.5.16: Event system

- Event emitters used?
- What events fired?
- Event subscribers?

Q1.5.17: Email service

- Which email provider?
- Template system?
- Transactional emails?

Q1.5.18: File storage service

- Local filesystem?
- S3 or similar?
- CDN integration?

Q1.5.19: PDF generation

- Any PDF generation?
- Which library?

Q1.5.20: Excel handling

- Reading Excel files?
- Writing Excel files?
- For BAT import/export?

Q1.5.21: Report generation

- Built-in reports?
- Which reports?
- Data aggregation logic?

Q1.5.22: Analytics tracking

- User activity tracking?
- Business metrics?

Q1.5.23: Scheduled tasks

- Cron jobs?
- What do they do?
- When do they run?

Q1.5.24: Data cleanup jobs

- Old data archival?
- Soft delete cleanup?

Q1.5.25: Testing of business logic

- Unit tests for services?
- Test coverage?
- Edge cases tested?

SECTION 1.6: Error Handling & Validation (10 questions)

Q1.6.1: Global error handler implementation

- Full code review
- All error types handled?

Q1.6.2: Custom error classes

- What custom errors exist?
- Inheritance hierarchy?
- Include code

Q1.6.3: Validation error handling

- Zod errors formatted nicely?
- Field-level errors returned?

Q1.6.4: Database error handling

- Prisma errors caught?
- Unique constraint violations?
- Foreign key errors?

Q1.6.5: Authentication errors

- Consistent messaging?
- Proper status codes?

Q1.6.6: Not found errors

- 404 handling?
- Resource-specific messages?

Q1.6.7: Forbidden errors

- 403 handling?
- When permissions insufficient?

Q1.6.8: Validation schema completeness

- Every POST/PUT endpoint validated?
- List all Zod schemas
- Review each schema

Q1.6.9: Error logging

- All errors logged?
- Stack traces captured?
- Error monitoring (Sentry, etc.)?

Q1.6.10: User-friendly error messages

- Technical jargon avoided?
- Actionable error messages?

PART 2: FRONTEND DEEP DIVE (100+ Questions)

SECTION 2.1: Project Structure & Setup (20 questions)

Q2.1.1: List EVERY file in frontend/src/

- Full file tree with line counts
- Categorize: components, pages, services, hooks, utils, types
- Any orphaned files?

Q2.1.2: Folder structure convention?

- Feature-based? Component-based?
- Consistent organization?
- Violations of pattern?

Q2.1.3: Analyze `frontend/package.json`

- All dependencies with versions
- Outdated packages?
- Security vulnerabilities?
- Unused dependencies?

Q2.1.4: Review Vite configuration

- Build settings?
- Dev server config?
- Environment variable handling?
- Proxy to backend configured?

Q2.1.5: TypeScript configuration

- Strict mode?
- Path aliases?
- Any type errors currently?

Q2.1.6: ESLint configuration

- Rules enabled?
- Any linting errors?
- Auto-fix on save?

Q2.1.7: Tailwind CSS configuration

- Custom theme?
- Plugins used?
- Purge configured for production?

Q2.1.8: Entry point analysis (`main.tsx`)

- How is React initialized?
- Providers wrapped?
- Router setup?
- Global styles imported?

Q2.1.9: Root component analysis (`App.tsx`)

- Layout structure?
- Router configuration?
- Protected route handling?
- Loading states?

Q2.1.10: Environment variable management

- `.env.example` present?
- What variables needed?
- `VITE_` prefix used?
- API URL configurable?

Q2.1.11: Build output analysis

- Production build works?
- Bundle size?
- Code splitting configured?
- Lazy loading implemented?

Q2.1.12: Asset management

- Images, fonts, icons?
- Where stored?
- Optimized?

Q2.1.13: PWA features?

- Service worker?
- Manifest file?
- Offline support?

Q2.1.14: Accessibility setup

- Semantic HTML?
- ARIA labels?
- Keyboard navigation?
- Screen reader tested?

Q2.1.15: Internationalization (i18n)?

- Multi-language support?
- Translation files?

Q2.1.16: Dark mode support?

- Theme switching?
- Persisted preference?

Q2.1.17: Browser compatibility

- Targets which browsers?
- Polyfills needed?

Q2.1.18: Performance monitoring

- Lighthouse scores?
- Core Web Vitals?

Q2.1.19: Error boundary implementation

- Error boundaries present?
- Fallback UI?
- Error reporting?

Q2.1.20: Code quality

- Prettier configured?
- Consistent code style?
- Component naming conventions?

SECTION 2.2: Routing & Navigation (15 questions)

Q2.2.1: Router library used?

- React Router DOM v7 as per package.json?
- Configuration correct?

Q2.2.2: List ALL routes

- Path, component, auth required
- Nested routes?
- Dynamic routes?

Q2.2.3: Protected routes implementation

- How are they protected?
- Redirect to login if not authenticated?
- Code review of protection mechanism

Q2.2.4: Route guards for roles?

- Can ESTIMATOR access admin routes?
- Authorization checks?

Q2.2.5: 404 Not Found page?

- Catch-all route?
- User-friendly message?

Q2.2.6: Navigation component(s)?

- Navbar, sidebar, etc.?
- Active route highlighting?
- Responsive?

Q2.2.7: Breadcrumbs?

- Implemented?
- Auto-generated or manual?

Q2.2.8: Back button handling?

- Browser back button works?
- In-app back buttons?

Q2.2.9: Deep linking?

- Can you link directly to specific resources?
- e.g., /customers/123/plans/456

Q2.2.10: Query parameters usage?

- Filters, sorting in URL?
- State persisted in URL?

Q2.2.11: Route transitions?

- Loading states between routes?
- Smooth transitions?

Q2.2.12: Redirect logic?

- After login?
- After logout?
- After actions?

Q2.2.13: Lazy loading routes?

- Code splitting by route?
- Suspense boundaries?

Q2.2.14: Route metadata?

- Page titles set?
- Meta tags for SEO?

Q2.2.15: Navigation menu state?

- Current section highlighted?
- Collapsible menu?
- Mobile responsive?

SECTION 2.3: Authentication Integration (20 questions)

Q2.3.1: Authentication context/provider?

- Location and code review
- What state managed?
- What methods provided?

Q2.3.2: Token storage?

- localStorage? sessionStorage? cookies?
- Security considerations?

Q2.3.3: Login page/component analysis

- Full code review
- Form validation?
- Error handling?
- Loading states?
- Remember me?

Q2.3.4: Registration page analysis

- Full code review
- Form fields?
- Validation?
- Password strength indicator?
- Terms acceptance?

Q2.3.5: Logout functionality

- How triggered?
- Token removal?
- State cleanup?
- Redirect?

Q2.3.6: Token refresh mechanism

- Automatic refresh before expiry?
- Interceptor for 401 responses?
- Code review

Q2.3.7: Authenticated user display

- User info shown in UI?
- Avatar?
- Dropdown menu?

Q2.3.8: Profile page

- View/edit profile?
- Change password?
- Upload avatar?

Q2.3.9: Role-based UI elements

- Components hidden based on role?
- Admin-only sections?

Q2.3.10: Password change flow

- Component exists?
- Current password required?
- Validation?

Q2.3.11: Password reset flow

- Forgot password link?
- Reset request page?
- Reset password page?

Q2.3.12: Session timeout handling

- Inactive session timeout?
- Warning before timeout?
- Auto-logout?

Q2.3.13: Multi-tab synchronization

- Login in one tab reflects in others?
- Logout synced?

Q2.3.14: Authentication errors displayed

- User-friendly messages?
- Form-level vs. field-level?

Q2.3.15: Loading states during auth

- Login button disabled while loading?
- Spinner shown?

Q2.3.16: Persisted login

- User stays logged in after page refresh?
- Token validation on app start?

Q2.3.17: Social login buttons

- UI present?
- Functional or placeholder?

Q2.3.18: Email verification prompt

- If unverified, prompt to verify?
- Resend verification email?

Q2.3.19: First-time user onboarding

- After registration, guide user?
- Tutorial or welcome wizard?

Q2.3.20: Security best practices followed?

- No passwords in URLs or logs?
- HTTPS enforced?
- Auth state cleared on logout?

SECTION 2.4: API Integration & Data Fetching (20 questions)

Q2.4.1: HTTP client setup

- Axios, fetch, or other?
- Base URL configured?
- Interceptors?

Q2.4.2: API service files

- Centralized API calls?
- E.g., authService.ts, customerService.ts?
- List all service files

Q2.4.3: TanStack Query usage

- QueryClient configured?
- Provider wrapping app?
- devtools enabled?

Q2.4.4: Query hooks analysis

- useQuery for GET requests?
- useMutation for POST/PUT/DELETE?
- List all query keys

Q2.4.5: Request interceptor

- Auth token attached automatically?
- Code review

Q2.4.6: Response interceptor

- Error handling centralized?
- Token refresh on 401?
- Code review

Q2.4.7: Loading states

- Global loading indicator?
- Component-level spinners?
- Skeleton screens?

Q2.4.8: Error handling

- Error boundaries?
- Toast notifications?
- Inline error messages?

Q2.4.9: Success feedback

- Toast notifications on success?
- Optimistic updates?

Q2.4.10: Caching strategy

- TanStack Query cache time?
- Stale time configured?
- Manual cache invalidation?

Q2.4.11: Polling/refetching

- Auto-refetch on interval?
- Refetch on window focus?
- Refetch on reconnect?

Q2.4.12: Infinite scroll / pagination

- Implemented for lists?
- useInfiniteQuery used?
- Load more button?

Q2.4.13: Optimistic updates

- UI updates before API response?
- Rollback on error?

Q2.4.14: Request debouncing/throttling

- Search input debounced?
- Form submission throttled?

Q2.4.15: API error messages displayed

- Backend error messages shown?
- Fallback messages?

Q2.4.16: Network error handling

- Offline detection?
- Retry mechanism?

Q2.4.17: API versioning on frontend

- API endpoints versioned?
- Configurable version?

Q2.4.18: Mock API for development?

- MSW or similar?
- Can frontend run without backend?

Q2.4.19: API request/response logging

- Console logging in dev?
- Disabled in production?

Q2.4.20: Type safety for API calls

- TypeScript types for requests/responses?
- Shared types from backend?

SECTION 2.5: Component Analysis (25 questions)

Q2.5.1: Component inventory

- List ALL components with file paths
- Categorize: pages, layouts, common, domain-specific

Q2.5.2: Component patterns used

- Functional components?
- Hooks?
- Class components (legacy)?

Q2.5.3: Component composition

- Atomic design?
- Atoms, molecules, organisms?
- Reusable components?

Q2.5.4: Common/shared components

- Button, Input, Modal, Card, etc.?
- Consistent styling?
- Props interface well-defined?

Q2.5.5: Form components

- Custom form inputs?
- Form validation library (React Hook Form, Formik)?
- Reusable form components?

Q2.5.6: Table/DataGrid component

- List views use a table component?
- Sorting, filtering, pagination?
- Responsive?

Q2.5.7: Modal/Dialog components

- Reusable modal component?
- Confirmation dialogs?
- Accessibility (focus trap, ESC to close)?

Q2.5.8: Layout components

- Header, Footer, Sidebar?
- Responsive layout?
- Consistent across pages?

Q2.5.9: Dashboard/home page

- What does it show?
- Widgets, stats, charts?
- Responsive?

Q2.5.10: Customer pages

- List customers page?
- Customer detail page?
- Create/edit customer forms?

Q2.5.11: Plan pages

- List plans page?
- Plan detail page?
- Create/edit plan forms?

Q2.5.12: Material pages

- Similar analysis

Q2.5.13: Job pages

- Similar analysis

Q2.5.14: Takeoff pages

- Similar analysis

Q2.5.15: Variance analysis pages

- UI for reviewing variance patterns?
- Approval workflow UI?

Q2.5.16: PO pages

- Similar analysis

Q2.5.17: User management pages (admin)

- List users?
- User detail/edit?

Q2.5.18: Reports pages

- Any report generation UI?

Q2.5.19: Settings pages

- User settings?
- Application settings (admin)?

Q2.5.20: Loading skeletons

- Skeleton components?
- Used across app?

Q2.5.21: Empty states

- When no data, friendly message?
- Call-to-action?

Q2.5.22: Error states

- Error components?
- Retry buttons?

Q2.5.23: Notification system

- Toast/snackbar component?
- Notification center?

Q2.5.24: Component props validation

- PropTypes or TypeScript interfaces?
- Required vs optional props clear?

Q2.5.25: Component documentation

- JSDoc comments?
- Storybook or similar?

SECTION 2.6: State Management (10 questions)

Q2.6.1: Global state management

- React Context?
- Redux, Zustand, Jotai?
- TanStack Query for server state?

Q2.6.2: Auth state management

- Where stored?
- How accessed across app?

Q2.6.3: UI state management

- Sidebar open/closed?
- Modal visibility?
- Theme preference?

Q2.6.4: Form state management

- React Hook Form?
- Formik?
- Controlled vs. uncontrolled?

Q2.6.5: Local storage usage

- What persisted?
- Serialization/deserialization?

Q2.6.6: Session storage usage

- Temporary data stored?

Q2.6.7: URL as state

- Filters, search in URL?
- Share-able links?

Q2.6.8: Derived state

- Computed values?
- useMemo for expensive calculations?

Q2.6.9: State synchronization

- Across components?
- Across tabs?

Q2.6.10: State debugging

- Redux DevTools?
- React DevTools?

SECTION 2.7: Forms & Validation (10 questions)

Q2.7.1: Form library used

- React Hook Form, Formik, or custom?

Q2.7.2: Validation library

- Zod, Yup, or built-in?

Q2.7.3: Client-side validation

- All forms validated before submission?
- Real-time vs. on-blur?

Q2.7.4: Validation error display

- Inline errors?
- Error summary?
- Accessible?

Q2.7.5: Form submission handling

- Disabled button while submitting?
- Success feedback?
- Error handling?

Q2.7.6: Multi-step forms

- Any wizards?
- Step navigation?
- State persisted between steps?

Q2.7.7: File upload forms

- Drag-and-drop?
- File type validation?
- File size validation?
- Preview before upload?

Q2.7.8: Date/time pickers

- Library used?
- Timezone handling?

Q2.7.9: Rich text editors

- Any rich text inputs?
- Which library?

Q2.7.10: Form auto-save

- Draft saving?
- Restore on return?

SECTION 2.8: UI/UX & Styling (10 questions)

Q2.8.1: Design system

- Consistent color palette?
- Typography defined?
- Spacing system?

Q2.8.2: Component styling approach

- Tailwind utility classes?
- CSS modules?
- Styled components?
- Inline styles?

Q2.8.3: Responsive design

- Mobile-first?

- Breakpoints defined?
- All pages responsive?

Q2.8.4: Icons

- Icon library (Lucide, Heroicons)?
- Consistent icon usage?

Q2.8.5: Loading indicators

- Spinners, progress bars?
- Consistent style?

Q2.8.6: Animations/transitions

- CSS transitions?
- Animation library?
- Smooth interactions?

Q2.8.7: Accessibility

- Semantic HTML used?
- ARIA labels?
- Keyboard navigation works?
- Focus management?
- Color contrast sufficient?

Q2.8.8: User feedback

- Hover states?
- Active states?
- Disabled states clear?

Q2.8.9: Consistent spacing

- Margins and paddings consistent?
- Layout gaps?

Q2.8.10: Typography

- Font family?
- Font sizes consistent?
- Line heights appropriate?

SECTION 2.9: Performance & Optimization (5 questions)

Q2.9.1: Code splitting

- Lazy loading implemented?
- Route-based splitting?
- Component-based splitting?

Q2.9.2: Memoization

- useMemo, useCallback used?
- React.memo for components?

- Over-optimization avoided?

Q2.9.3: Image optimization

- Images lazy loaded?
- Responsive images?
- WebP format?

Q2.9.4: Bundle analysis

- Bundle size measured?
- Large dependencies identified?
- Tree shaking working?

Q2.9.5: Rendering performance

- Unnecessary re-renders avoided?
 - Virtualization for long lists?
-

SECTION 2.10: Testing (5 questions)

Q2.10.1: Testing framework

- Jest, Vitest, or other?
- Configured?

Q2.10.2: Unit tests

- Components tested?
- Utilities tested?
- Hooks tested?

Q2.10.3: Integration tests

- API mocking?
- User flows tested?

Q2.10.4: E2E tests

- Playwright, Cypress?
- Critical paths covered?

Q2.10.5: Test coverage

- Coverage reports?
 - What percentage?
 - Coverage goals?
-

PART 3: FRONTEND-BACKEND INTEGRATION (30 Questions)

SECTION 3.1: Integration Points (15 questions)

Q3.1.1: For EACH API endpoint, is there a corresponding UI?

- Create a comprehensive table:
 - API Endpoint
 - HTTP Method
 - UI Component/Page
 - Integration Status (Complete, Partial, Missing)

Q3.1.2: Authentication flow end-to-end test

- Can you register? (trace from UI to backend to database)
- Can you login? (complete flow)
- Can you logout?
- Token refresh working?

Q3.1.3: Customer CRUD end-to-end

- Create customer via UI → API → DB
- Read/list customers
- Update customer
- Delete customer
- Any broken links?

Q3.1.4: Plan CRUD end-to-end

- Similar analysis

Q3.1.5: Material CRUD end-to-end

- Similar analysis

Q3.1.6: Job CRUD end-to-end

- Similar analysis

Q3.1.7: Data consistency

- Frontend and backend data models aligned?
- Any mismatches in field names, types?

Q3.1.8: Error message propagation

- Backend errors reach frontend?
- Displayed to user appropriately?

Q3.1.9: Loading states consistency

- All API calls have loading indicators?

Q3.1.10: CORS issues?

- Backend allows frontend origin?
- OPTIONS requests handled?

Q3.1.11: Request/response data transformation

- Any DTOs on frontend?
- Data mapping needed?
- Consistent date formats?

Q3.1.12: File upload flow

- Frontend uploads to backend?
- Progress indication?
- Error handling?

Q3.1.13: Real-time features

- WebSockets or polling?
- Live updates working?

Q3.1.14: Search functionality

- Frontend search sends to backend?
- Debouncing working?
- Results display correctly?

Q3.1.15: Pagination integration

- Frontend pagination controls?
- Backend pagination working?
- Total count displayed?

SECTION 3.2: Data Flow Analysis (10 questions)

Q3.2.1: Create operation data flow

- User fills form → Validation → API call → Backend validation → Database insert → Response → UI update
- Trace for one entity (e.g., Customer)

Q3.2.2: Read operation data flow

- Similar trace

Q3.2.3: Update operation data flow

- Optimistic update?
- Rollback on error?

Q3.2.4: Delete operation data flow

- Confirmation dialog?
- Soft delete vs. hard delete?

Q3.2.5: Login data flow

- Credentials → API → Auth → Token generation → Storage → UI update

Q3.2.6: Authenticated request flow

- Request → Interceptor adds token → Backend verifies → Response

Q3.2.7: Error flow

- Backend error → Frontend error handler → User notification

Q3.2.8: File upload flow

- File selection → Preview → Upload → Progress → Success/Error

Q3.2.9: Complex workflow (e.g., Job creation)

- Multiple steps?
- Multiple API calls?
- Transaction handling?

Q3.2.10: Report generation flow

- Request report → Backend generates → Frontend downloads/displays
-

SECTION 3.3: Type Safety Across Stack (5 questions)

Q3.3.1: Shared types directory

- shared/types/ used?
- What's in it?
- Both frontend and backend import?

Q3.3.2: API request types

- Request body types defined?
- Shared or duplicated?

Q3.3.3: API response types

- Response types defined?
- Consistent with backend?

Q3.3.4: Enum consistency

- Same enums on frontend and backend?
- E.g., UserRole, JobStatus

Q3.3.5: Type generation

- Prisma types exported?
 - Code generation for types?
-

PART 4: COMPREHENSIVE ISSUES & BUGS ANALYSIS (50 Questions)

SECTION 4.1: Critical Issues Search (20 questions)

Q4.1.1: Search for TODO comments

- List EVERY TODO in the codebase
- File, line number, context
- Priority of each

Q4.1.2: Search for FIXME comments

- List EVERY FIXME
- What needs fixing?

Q4.1.3: Search for HACK comments

- Any hacky workarounds?

Q4.1.4: Search for console.log statements

- Should be removed for production
- List all instances

Q4.1.5: Search for commented-out code

- Dead code to remove?

Q4.1.6: Search for any syntax errors

- Run TypeScript compiler
- Any errors?

Q4.1.7: Search for eslint errors

- Run eslint
- Any errors or warnings?

Q4.1.8: Search for Prisma query errors

- Any improper queries?
- N+1 problems?

Q4.1.9: Search for hardcoded values

- Should be env variables?
- Should be constants?

Q4.1.10: Search for magic numbers

- Unexplained numeric literals?

Q4.1.11: Search for security issues

- Any SQL injection risks?
- XSS vulnerabilities?
- Exposed secrets?

Q4.1.12: Search for performance issues

- Inefficient loops?
- Unnecessary re-renders?
- Large dependencies?

Q4.1.13: Search for unused variables

- Dead code?

Q4.1.14: Search for unused imports

- Clean up needed?

Q4.1.15: Search for missing error handling

- Try-catch blocks missing?
- Unhandled promise rejections?

Q4.1.16: Search for missing validation

- Endpoints without validation?

Q4.1.17: Search for missing authentication

- Routes that should be protected?

Q4.1.18: Search for deprecated code

- Using deprecated APIs?

Q4.1.19: Search for type 'any' usage

- TypeScript any overused?
- Should be typed?

Q4.1.20: Search for incomplete implementations

- Placeholder functions?
- Not implemented yet?

SECTION 4.2: Integration Issues (10 questions)

Q4.2.1: Missing API endpoint implementations

- Frontend calls endpoint that doesn't exist?

Q4.2.2: Missing UI for backend features

- Backend feature without UI?

Q4.2.3: Data model mismatches

- Frontend expects field that backend doesn't return?

Q4.2.4: Broken links/navigation

- Links to non-existent pages?

Q4.2.5: Inconsistent error handling

- Some places handle errors, others don't?

Q4.2.6: Missing loading states

- API calls without spinners?

Q4.2.7: CORS configuration issues

- Would it work in production?

Q4.2.8: Authentication token issues

- Token expiry not handled?
- Refresh not working?

Q4.2.9: File upload issues

- File size limits consistent?
- File type validation?

Q4.2.10: WebSocket/real-time issues

- If implemented, working correctly?
-

SECTION 4.3: Database Issues (10 questions)

Q4.3.1: Migration issues

- Any failed migrations?
- Migration order correct?

Q4.3.2: Missing indexes

- Slow queries due to missing indexes?

Q4.3.3: Missing foreign key constraints

- Referential integrity issues?

Q4.3.4: Orphaned records possible?

- Cascade deletes configured?

Q4.3.5: Data type mismatches

- String when should be Int?
- DateTime timezone issues?

Q4.3.6: Missing required fields

- Fields that should be required but aren't?

Q4.3.7: Missing default values

- Fields that should have defaults?

Q4.3.8: Enum value mismatches

- Frontend and backend enums different?

Q4.3.9: Seed data issues

- Seed script errors?
- Seed data realistic?

Q4.3.10: Database connection issues

- Connection pool exhaustion possible?
 - Timeout issues?
-

SECTION 4.4: UI/UX Issues (10 questions)

Q4.4.1: Responsive design issues

- Pages broken on mobile?
- Elements overflow?

Q4.4.2: Accessibility issues

- Missing labels?
- Poor keyboard navigation?
- Color contrast issues?

Q4.4.3: Form validation issues

- Inconsistent validation?
- Poor error messages?

Q4.4.4: Loading state issues

- Infinite spinners?
- No feedback?

Q4.4.5: Error display issues

- Errors not shown?
- Confusing error messages?

Q4.4.6: Navigation issues

- Can't get back?
- Unexpected redirects?

Q4.4.7: Empty state issues

- Ugly empty states?
- No guidance?

Q4.4.8: Button/link functionality

- Buttons that don't work?
- Links that 404?

Q4.4.9: Modal/dialog issues

- Can't close modal?
- Modal accessibility?

Q4.4.10: Performance issues

- Slow page loads?

- Laggy interactions?
-

PART 5: PRODUCTION READINESS (40 Questions)

SECTION 5.1: Deployment Preparation (15 questions)

Q5.1.1: Environment configurations complete?

- All env vars documented?
- Production .env.example?

Q5.1.2: Secrets management

- How will secrets be managed in prod?
- Encrypted? Vault?

Q5.1.3: Database migration strategy

- How to run migrations in prod?
- Downtime required?
- Rollback plan?

Q5.1.4: Build process

- Production build works?
- Any build errors?
- Build artifacts optimized?

Q5.1.5: Docker setup (if using)

- Dockerfile present?
- Multi-stage build?
- docker-compose for prod?

Q5.1.6: CI/CD pipeline

- GitHub Actions, CircleCI, etc.?
- Automated tests?
- Automated deployment?

Q5.1.7: Hosting platform decided?

- Railway, Render, AWS?
- Database hosting?
- File storage hosting?

Q5.1.8: Domain and SSL

- Domain configured?
- SSL certificate?
- HTTPS enforced?

Q5.1.9: CDN setup

- For static assets?

- Caching configured?

Q5.1.10: Load balancing

- Needed?
- Configured?

Q5.1.11: Database backups

- Automated backups configured?
- Backup schedule?
- Restore tested?

Q5.1.12: Monitoring setup

- Application monitoring (Sentry, etc.)?
- Server monitoring?
- Database monitoring?
- Alerts configured?

Q5.1.13: Logging aggregation

- Logs centralized?
- Log retention policy?

Q5.1.14: Performance monitoring

- APM tool (New Relic, DataDog)?
- Metrics tracked?

Q5.1.15: Uptime monitoring

- External uptime monitor?
- Health check endpoint?

SECTION 5.2: Security Hardening (15 questions)

Q5.2.1: HTTPS enforced?

- HTTP redirects to HTTPS?
- HSTS header?

Q5.2.2: Security headers

- Helmet.js in production?
- CSP configured?
- X-Frame-Options, X-Content-Type-Options?

Q5.2.3: CORS properly configured

- Production origins only?
- Not allowing *?

Q5.2.4: Rate limiting

- Prevents brute force?

- DDoS protection?

Q5.2.5: Input sanitization

- XSS prevention?
- SQL injection prevention (Prisma handles)?

Q5.2.6: SQL injection tests

- Any raw SQL?
- Parameterized queries?

Q5.2.7: Authentication security

- JWT secret strong?
- Token expiry reasonable?
- Refresh token rotation?

Q5.2.8: Password security

- Bcrypt salt rounds sufficient?
- Password strength requirements?

Q5.2.9: Session security

- Secure cookies?
- HttpOnly cookies?
- SameSite attribute?

Q5.2.10: File upload security

- File type validation?
- File size limits?
- Virus scanning?
- Stored outside webroot?

Q5.2.11: Dependency vulnerabilities

- npm audit clean?
- Outdated packages updated?

Q5.2.12: Secrets in code?

- No hardcoded secrets?
- .env not committed?

Q5.2.13: Error messages

- Don't leak sensitive info?
- Stack traces hidden in prod?

Q5.2.14: Admin endpoints

- Extra protection?
- IP whitelisting needed?

Q5.2.15: Penetration testing

- Has it been pen-tested?
 - Security audit?
-

SECTION 5.3: Performance & Scalability (10 questions)

Q5.3.1: Database query optimization

- Indexes on foreign keys?
- Complex queries optimized?
- Query execution plans reviewed?

Q5.3.2: Caching strategy

- Redis for session storage?
- API response caching?
- Database query caching?

Q5.3.3: Connection pooling

- Database connection pool sized?
- Connection leaks prevented?

Q5.3.4: Frontend bundle size

- Bundle analyzed?
- Size acceptable (<500KB)?
- Code splitting effective?

Q5.3.5: Image optimization

- Images compressed?
- Lazy loading?
- Responsive images?

Q5.3.6: API response times

- Endpoints respond <500ms?
- Slow endpoints identified?

Q5.3.7: Database size projections

- Growth rate estimated?
- Archival strategy?

Q5.3.8: Horizontal scaling

- Can backend scale horizontally?
- Stateless design?

Q5.3.9: Load testing

- Has it been load tested?
- What's the capacity?

Q5.3.10: Resource usage

- Memory usage reasonable?
 - CPU usage reasonable?
-

PART 6: SPECIFIC BUSINESS REQUIREMENTS (30 Questions)

SECTION 6.1: Richmond BAT Conversion Requirements (15 questions)

Q6.1.1: Plan import functionality

- Can plans be imported from Excel?
- What format expected?
- Validation of imported data?

Q6.1.2: Material library setup

- Are Richmond-specific materials in system?
- Can materials be bulk imported?

Q6.1.3: Pricing structure

- Richmond American pricing tiers supported?
- Customer-specific pricing overrides?

Q6.1.4: Plan templates (BOMs)

- Can plan templates be created?
- Template = list of materials with quantities?

Q6.1.5: Option system

- How are plan options handled?
- Can options modify the BOM?

Q6.1.6: Elevation variants

- How are plan elevations handled?
- Elevation-specific material changes?

Q6.1.7: Job setup from plan

- Can a job be created from a plan?
- Does it generate initial takeoff?

Q6.1.8: Takeoff generation

- Automatic BOM from plan template?
- Editable quantities?

Q6.1.9: Variance tracking readiness

- Can record quantityEstimated vs. quantityActual?
- Variance reason capture?

Q6.1.10: Pricing pipeline

- Transparent pricing calculations?

- Multi-step pricing?
- Auditable price breakdown?

Q6.1.11: External system codes

- Richmond's Sales 1440 codes supported?
- Mapping to internal IDs?

Q6.1.12: Reports for Richmond

- Job costing reports?
- Variance reports?
- Pricing reports?

Q6.1.13: Excel export for BAT

- Can export back to Excel format?
- Compatible with current BAT?

Q6.1.14: Data migration path

- How to migrate 40 plans?
- Bulk import tool?
- One-by-one?

Q6.1.15: Training materials

- Documentation for using system?
- User guide?
- Video tutorials?

SECTION 6.2: Core Business Logic (15 questions)

Q6.2.1: Customer management completeness

- All customer fields captured?
- Pricing tier logic working?
- Multiple contacts per customer?

Q6.2.2: Plan management completeness

- Plan metadata (name, builder, sqft)?
- Elevation management?
- Option management?
- Template (BOM) management?

Q6.2.3: Material management completeness

- Material catalog?
- Vendor association?
- Pricing info?
- Commodity pricing integration?

Q6.2.4: Job management completeness

- Job creation and tracking?

- Status workflow?
- Association to customer, plan, lot?

Q6.2.5: Takeoff management completeness

- Generate takeoff from plan?
- Edit line items?
- Approve/lock takeoff?

Q6.2.6: Purchase Order functionality

- Create PO from takeoff?
- PO status workflow?
- PO approval?

Q6.2.7: Variance analysis

- Capture actual quantities used?
- Calculate variance?
- Categorize variance reasons?
- Pattern detection?

Q6.2.8: Pricing calculations

- Unit prices calculated correctly?
- Margins applied?
- Customer discounts applied?
- Commodity adjustments?

Q6.2.9: Notifications

- Email notifications?
- In-app notifications?
- What triggers them?

Q6.2.10: User roles and permissions

- ADMIN can do everything?
- ESTIMATOR can create jobs?
- PROJECT_MANAGER can approve?
- VIEWER read-only?

Q6.2.11: Audit trail

- Actions logged?
- Who did what, when?
- Viewable by admin?

Q6.2.12: Data validation

- Business rules enforced?
- Data integrity checks?

Q6.2.13: Calculations accuracy

- Financial calculations precise?
- No floating point errors?

Q6.2.14: External integrations

- Random Lengths API working?
- Any other integrations?

Q6.2.15: Reporting capabilities

- Built-in reports?
- Custom report builder?
- Export capabilities?

PART 7: TESTING & QUALITY ASSURANCE (25 Questions)

SECTION 7.1: Test Coverage (10 questions)

Q7.1.1: Backend unit tests

- Which files have tests?
- Coverage percentage?
- Key functions tested?

Q7.1.2: Frontend unit tests

- Which components tested?
- Coverage percentage?

Q7.1.3: Integration tests

- API endpoint tests?
- Full request-response cycle?

Q7.1.4: E2E tests

- Critical user journeys covered?
- Registration and login flow?
- Job creation flow?

Q7.1.5: Test data management

- Test database setup?
- Seed data for tests?
- Test data cleanup?

Q7.1.6: Test documentation

- How to run tests?
- What's tested?
- Known test failures?

Q7.1.7: Continuous testing

- Tests run on commit?
- CI pipeline includes tests?

Q7.1.8: Test environment

- Separate test environment?
- Matches production?

Q7.1.9: Load testing

- Performance under load?
- Concurrent users tested?

Q7.1.10: Security testing

- Penetration testing?
 - Vulnerability scanning?
-

SECTION 7.2: Manual Testing Checklist (15 questions)

Q7.2.1: User registration works?

- Fill form, submit, account created?
- Email verification sent?

Q7.2.2: User login works?

- Correct credentials → dashboard?
- Incorrect credentials → error?

Q7.2.3: Token refresh works?

- Stay logged in beyond access token expiry?

Q7.2.4: User logout works?

- Token cleared?
- Redirect to login?

Q7.2.5: Create customer works?

- Form validation?
- Success feedback?
- Appears in list?

Q7.2.6: View customer works?

- Detail page loads?
- Data displayed correctly?

Q7.2.7: Update customer works?

- Edit form pre-filled?
- Changes saved?

Q7.2.8: Delete customer works?

- Confirmation dialog?
- Removed from list?

Q7.2.9: Create plan works?

- Similar to customer

Q7.2.10: Create material works?

- Similar to customer

Q7.2.11: Create job works?

- Similar to customer

Q7.2.12: Generate takeoff works?

- From plan template?
- Line items correct?

Q7.2.13: Record variance works?

- Can enter actual quantities?
- Variance calculated?

Q7.2.14: Create PO works?

- From takeoff?
- Data transferred correctly?

Q7.2.15: Access control works?

- ESTIMATOR can't access admin routes?
- Permissions enforced?

PART 8: DOCUMENTATION & KNOWLEDGE TRANSFER (20 Questions)

SECTION 8.1: Documentation Quality (10 questions)

Q8.1.1: README.md completeness

- Project overview clear?
- Setup instructions accurate?
- Up to date?

Q8.1.2: API documentation

- Endpoints documented?
- Request/response examples?
- Error codes explained?

Q8.1.3: Database documentation

- ER diagram?
- Model descriptions?
- Relationship explanations?

Q8.1.4: Component documentation

- PropTypes/interfaces documented?
- Usage examples?

Q8.1.5: Business logic documentation

- Complex calculations explained?
- Workflow diagrams?

Q8.1.6: Setup guides

- SETUP.md accurate?
- LAUNCH_GUIDE.md accurate?
- Windows and Mac covered?

Q8.1.7: Deployment documentation

- Deployment steps?
- Environment setup?
- Troubleshooting?

Q8.1.8: User documentation

- User manual?
- Feature guides?
- FAQs?

Q8.1.9: Developer documentation

- Contribution guide?
- Code style guide?
- Architecture overview?

Q8.1.10: Changelog

- Version history?
- Release notes?

SECTION 8.2: Code Quality (10 questions)

Q8.2.1: Code consistency

- Naming conventions followed?
- Formatting consistent?

Q8.2.2: Code comments

- Complex logic commented?
- Why, not just what?

Q8.2.3: Function complexity

- Functions too long?
- Should be refactored?

Q8.2.4: DRY principle

- Code duplication?
- Reusable functions?

Q8.2.5: SOLID principles

- Single responsibility?
- Open/closed?
- Liskov substitution?
- Interface segregation?
- Dependency inversion?

Q8.2.6: Error handling consistency

- Consistent error patterns?

Q8.2.7: Logging consistency

- Consistent log messages?
- Appropriate log levels?

Q8.2.8: Type safety

- Strong typing?
- Minimal 'any'?

Q8.2.9: Test quality

- Tests meaningful?
- Good coverage?
- Not brittle?

Q8.2.10: Technical debt

- Known debt documented?
- Refactoring planned?

PART 9: FINAL COMPREHENSIVE ANALYSIS

SECTION 9.1: Executive Summary (Required Output)

After answering ALL questions above, provide:

A. Overall Project Status



Completion Percentage: X%

Backend: X%
Frontend: X%
Integration: X%
Testing: X%
Production: X%

B. Critical Blockers (Top 10)

List the TOP 10 issues that MUST be resolved before launch:

1. [Issue Title]
 - o **Category:** Backend / Frontend / Integration / Database / Security
 - o **Severity:** ● Critical / ● High / ● Medium
 - o **Description:** [What's wrong]
 - o **Impact:** [What it blocks or breaks]
 - o **Fix Required:** [What needs to be done]
 - o **Estimated Time:** X hours
 - o **Files Affected:** [file paths]
 - o **Line Numbers:** [if applicable]
2. [Repeat for all 10]

C. Feature Completeness Matrix

Feature Category	Status	Completion	Critical	Gaps
Authentication	✓ ⚠ ✗	X%		[list gaps]
Customer Mgmt	✓ ⚠ ✗	X%		[list gaps]
Plan Mgmt	✓ ⚠ ✗	X%		[list gaps]
Material Mgmt	✓ ⚠ ✗	X%		[list gaps]
Job Mgmt	✓ ⚠ ✗	X%		[list gaps]
Takeoff	✓ ⚠ ✗	X%		[list gaps]
Variance	✓ ⚠ ✗	X%		[list gaps]
PO Mgmt	✓ ⚠ ✗	X%		[list gaps]
Pricing	✓ ⚠ ✗	X%		[list gaps]
Reports	✓ ⚠ ✗	X%		[list gaps]
Admin	✓ ⚠ ✗	X%		[list gaps]

D. Quick Wins (Can Complete in <2 hours each)

List 10-15 quick wins:

1. [Task] - X min - [File] - [What to do]
2. [Continue...]

E. Long Poles (Will Take >10 hours each)

List 5-10 major undertakings:

1. [Feature/Fix] - X hours - [Why it's complex] - [Dependencies]

2. [Continue...]

F. Development Roadmap to Production

Phase 1: Critical Fixes (Week 1)

- Fix [blocker 1] - X hours
- Fix [blocker 2] - X hours
- Total: X hours

Phase 2: Feature Completion (Week 2-3)

- Complete [feature 1] - X hours
- Complete [feature 2] - X hours
- Total: X hours

Phase 3: Integration & Testing (Week 4)

- Test all integrations - X hours
- Fix integration bugs - X hours
- Total: X hours

Phase 4: Production Prep (Week 5)

- Security hardening - X hours
- Performance optimization - X hours
- Deployment setup - X hours
- Total: X hours

Total Estimated Time to Production-Ready: X hours **At 60-90 min/day:** X-X weeks

G. Risk Assessment

Risk	Likelihood	Impact	Mitigation
[Risk 1]	High/Med/Low	High/Med/Low	[Strategy]
[Continue for top 10 risks]			

H. Recommendations

Should development continue on this codebase?

- YES - X% complete, clearly production-bound
- YES WITH CAUTION - Significant work remains
- NO - Consider alternative approach

Reasoning: [Your analysis]

Recommended Next Steps:

1. [Priority 1 action]
2. [Priority 2 action]
3. [Priority 3 action]

I. Richmond BAT Conversion Readiness

Can the platform support the 40-plan conversion today?

- YES - Ready to start conversion
- PARTIAL - Some features needed first
- NO - Significant work required

Missing for BAT conversion:

1. [Feature/capability 1]
2. [Feature/capability 2]

J. Budget Assessment

Development effort remaining: X hours

At developer rate of \$100/hour: \$X,XXX

Compared to building from scratch: [Analysis]

Conclusion: [Recommendation]

FINAL NOTES FOR CLAUDE CODE

Analysis Approach:

1. Be EXHAUSTIVE - Read every file, check every integration
2. Be SPECIFIC - Always include file paths and line numbers
3. Be HONEST - Don't sugarcoat problems
4. Be PRACTICAL - Give actionable recommendations with time estimates
5. Be THOROUGH - Use all \$500 of compute credit if needed

Output Format:

- Use markdown tables for clarity
- Use checkboxes for completion status
- Use emoji indicators:
- Include code snippets where helpful (max 20 lines)
- Link related findings together

Priority:

Focus on finding **REAL ISSUES** that would block production or break functionality. Don't focus on minor style issues unless they indicate larger problems.

End Goal:

Provide Corey with enough information to make an informed decision:

- Continue with MindFlow?
- How much work remains?
- What are the risks?
- What's the timeline?
- Is it production-worthy?

Thank you for the comprehensive analysis!