BuildOS - Application Technical Specification

BuildOS Engineering Team

January 2026

Contents

1 Architecture Overview

BuildOS is built on a modern, scalable technology stack designed for performance, maintainability, and developer productivity.

1.1 Technology Stack

1.1.1 Frontend Framework

Next.js 14.2.28 (App Router)

- **Server-Side Rendering (SSR):** Improved SEO and initial page load performance
- **App Router:** Modern routing with layouts, loading states, and error boundaries
- **Server Components:** Reduced client-side JavaScript bundle size
- **API Routes:** Built-in API endpoints for backend functionality
- **Image Optimization:** Automatic image optimization and lazy loading
- **TypeScript Support:** Full type safety across the application

React 18.2.0

- **Concurrent Features:** Improved rendering performance
- **Automatic Batching:** Optimized state updates
- **Suspense:** Better loading state management
- **Hooks:** Modern state and lifecycle management
- **Context API:** Global state management for auth and notifications

1.1.2 Backend & Database

Prisma 6.7.0 ORM

- **Type-Safe Database Access:** Auto-generated TypeScript types
- **Schema Management:** Declarative schema definition
- **Migration System:** Version-controlled database changes
- **Query Builder:** Intuitive and type-safe query API
- **Connection Pooling:** Efficient database connection management
- **Middleware Support:** Request logging and error handling

PostgreSQL Database

- **Relational Database:** ACID compliance for data integrity
- **Advanced Features:** JSON support, full-text search, array types
- **Scalability:** Horizontal and vertical scaling capabilities
- **Performance:** Optimized indexes and query planning
- **Backup & Recovery:** Point-in-time recovery support
- **Extensions:** PostGIS for geospatial data (future use)

1.1.3 File Storage

AWS S3

- **Scalable Storage:** Unlimited storage capacity
- **High Availability:** 99.99% uptime SLA
- **Security:** Encryption at rest and in transit
- **Presigned URLs:** Secure, time-limited file access
- **Versioning:** File version history
- **Lifecycle Policies:** Automated archival and deletion

1.1.4 Authentication

NextAuth.js

- **Multiple Providers:** Email/password, OAuth, SSO support
- **JWT Tokens:** Stateless authentication
- **Session Management:** Secure session handling
- **CSRF Protection:** Built-in security features
- **Callbacks:** Customizable authentication flow
- **Database Sessions:** Optional persistent sessions

1.2 Architecture Patterns

1.2.1 Layered Architecture

```
   Presentation Layer (React)
 - Pages, Components, UI Logic

           ↓

  Application Layer (Next.js)
 - API Routes, Server Actions

           ↓

  Business Logic Layer (Services)
 - Domain Logic, Validation

           ↓

  Data Access Layer (Prisma)
 - Database Queries, ORM

           ↓

  Database Layer (PostgreSQL)
 - Data Storage, Relationships
```

1.2.2 Design Patterns

- **Repository Pattern:** Abstraction over data access
- **Service Layer Pattern:** Business logic encapsulation
- **Factory Pattern:** Object creation for complex entities
- **Observer Pattern:** Event-driven notifications
- **Singleton Pattern:** Database connection management
- **Middleware Pattern:** Request/response processing

1.2.3 API Design

- **RESTful Principles:** Resource-based URLs, HTTP methods
- **Consistent Response Format:** Standardized success/error responses
- **Pagination:** Cursor-based and offset-based pagination
- **Filtering & Sorting:** Query parameter-based filtering
- **Error Handling:** Consistent error codes and messages

- **Versioning:** API version support for backward compatibility

2 Database Schema (Prisma Models)

2.1 User Management

2.1.1 User Model

```
model User {
  id                String    @id @default(cuid())
  email             String    @unique
  name              String?
  password          String
  role              Role      @default(USER)
  createdAt         DateTime  @default(now())
  updatedAt         DateTime  @updatedAt

  // Relationships
  projects          Project[]
  rfis              RFI[]
  submittals        Submittal[]
  changeOrders      ChangeOrder[]
  punchItems        PunchItem[]
  dailyReports      DailyReport[]
  documents         Document[]
  timeEntries       TimeEntry[]
  notifications     Notification[]
  notificationPrefs NotificationPreference?
  activities        Activity[]
  designRequests    DesignRequest[]
}

enum Role {
  ADMIN
  PROJECT_MANAGER
  FIELD_SUPERVISOR
  SUBCONTRACTOR
  CLIENT
}
```

2.2 Project Management

2.2.1 Project Model

```
model Project {
  id              String    @id @default(cuid())
  name            String
  description     String?
  location        String?
  startDate       DateTime?
  endDate         DateTime?
  budget          Decimal?  @db.Decimal(12, 2)
  status          ProjectStatus @default(PLANNING)
  clientName      String?
  projectManager  String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  user            User      @relation(fields: [userId], references: [id])
  rfis            RFI[]
  submittals      Submittal[]
  changeOrders    ChangeOrder[]
  punchItems      PunchItem[]
  dailyReports    DailyReport[]
  documents       Document[]
  timeEntries     TimeEntry[]
  activities      Activity[]
  equipment       Equipment[]
  materials       MaterialRequisition[]
  designRequests  DesignRequest[]
  properties      Property[]
}

enum ProjectStatus {
  PLANNING
  ACTIVE
  ON_HOLD
  COMPLETED
  CANCELLED
}
```

2.2.2 Property Model (Real Estate)

```
model Property {
  id                String    @id @default(cuid())
  name              String
  address           String?
  propertyType      String?
  squareFootage     Int?
  numberOfUnits     Int?
  assetTypeId       String?
  developmentStageId String?
  projectId         String?
  createdAt         DateTime  @default(now())
  updatedAt         DateTime  @updatedAt

  // Relationships
  assetType         AssetType? @relation(fields: [assetTypeId], references: [id])
  developmentStage  DevelopmentStage? @relation(fields: [developmentStageId], references: [id])
  project           Project?   @relation(fields: [projectId], references: [id])
}

model AssetType {
  id          String     @id @default(cuid())
  name        String     @unique
  description String?
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt
  properties  Property[]
}

model DevelopmentStage {
  id          String     @id @default(cuid())
  name        String     @unique
  description String?
  order       Int        @default(0)
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt

  properties  Property[]
}
```

2.3 RFI System

2.3.1 RFI Model

```
model RFI {
  id          String    @id @default(cuid())
  projectId   String
  subject     String
  description String
  status      RFIStatus @default(DRAFT)
  priority    Priority  @default(MEDIUM)
  dueDate     DateTime?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
  userId      String

  // Relationships
  project     Project   @relation(fields: [projectId], references: [id])
  user        User      @relation(fields: [userId], references: [id])
  responses   RFIResponse[]
  comments    RFIComment[]
}

enum RFIStatus {
  DRAFT
  SUBMITTED
  UNDER_REVIEW
  ANSWERED
  CLOSED
}

enum Priority {
  LOW
  MEDIUM
  HIGH
  CRITICAL
}

model RFIResponse {
  id          String    @id @default(cuid())
  rfiId       String
  response    String
  respondedBy String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  rfi         RFI       @relation(fields: [rfiId], references: [id], onDelete: Cascade)
}

model RFIComment {
  id          String    @id @default(cuid())
  rfiId       String
  comment     String
  authorName  String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  rfi         RFI       @relation(fields: [rfiId], references: [id], onDelete: Cascade)
}
```

2.4 Submittal System

2.4.1 Submittal Model

```
model Submittal {
  id              String    @id @default(cuid())
  projectId       String
  submittalNumber String
  title           String
  description     String?
  type            SubmittalType
  status          SubmittalStatus @default(PENDING)
  submittedDate   DateTime?
  requiredDate    DateTime?
  specSection     String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
  responses       SubmittalResponse[]
  comments        SubmittalComment[]
}

enum SubmittalType {
  SHOP_DRAWINGS
  PRODUCT_DATA
  SAMPLES
  MIX_DESIGNS
  TEST_REPORTS
  OTHER
}
```

```
enum SubmittalStatus {
  PENDING
  UNDER_REVIEW
  APPROVED
  APPROVED_AS_NOTED
  REJECTED
  RESUBMIT
}

model SubmittalResponse {
  id          String    @id @default(cuid())
  submittalId String
  response    String
  reviewedBy  String
  decision    SubmittalStatus
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  submittal   Submittal @relation(fields: [submittalId], references: [id], onDelete: Cascade)
}

model SubmittalComment {
  id          String    @id @default(cuid())
  submittalId String
  comment     String
  authorName  String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  submittal   Submittal @relation(fields: [submittalId], references: [id], onDelete: Cascade)
}
```

2.5 Change Orders

2.5.1 ChangeOrder Model

```
model ChangeOrder {
  id                String    @id @default(cuid())
  projectId         String
  changeOrderNumber String
  title             String
  description       String
  reason            String?
  status            ChangeOrderStatus @default(DRAFT)
  originalCost      Decimal?  @db.Decimal(12, 2)
  proposedCost      Decimal?  @db.Decimal(12, 2)
  approvedCost      Decimal?  @db.Decimal(12, 2)
  scheduleImpact    Int?      // Days
  requestedDate     DateTime?
  approvedDate      DateTime?
  createdAt         DateTime  @default(now())
  updatedAt         DateTime  @updatedAt
  userId            String
  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
}

enum ChangeOrderStatus {
  DRAFT
  PENDING
  APPROVED
  REJECTED
  IMPLEMENTED
}
```

2.6 Punch Items

2.6.1 PunchItem Model

```
model PunchItem {
  id              String    @id @default(cuid())
  projectId       String
  itemNumber      String
  description     String
  location        String?
  assignedTo      String?
  status          PunchItemStatus @default(OPEN)
  priority        Priority  @default(MEDIUM)
  dueDate         DateTime?
  completedDate   DateTime?
  verifiedDate    DateTime?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
}

enum PunchItemStatus {
  OPEN
  IN_PROGRESS
  COMPLETED
  VERIFIED
  CLOSED
}
```

2.7 Daily Reports

2.7.1 DailyReport Model

```
model DailyReport {
  id              String    @id @default(cuid())
  projectId       String
  reportDate      DateTime
  weather         String?
  temperature     String?
  workPerformed   String
  crewSize        Int?
  equipmentUsed   String?
  materialsDelivered String?
  visitors        String?
  safetyIncidents String?
  delays          String?
  notes           String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
}
```

2.8 Document Management

2.8.1 Document Model

```
model Document {
  id              String    @id @default(cuid())
  projectId       String
  title           String
  description     String?
  fileName        String
  fileSize        Int?
  fileType        String?
  category        DocumentCategory
  uploadDate      DateTime  @default(now())
  cloudStoragePath String?
  isPublic        Boolean   @default(false)
  version         String?   @default("1.0")
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
}

enum DocumentCategory {
  DRAWINGS
  SPECIFICATIONS
  CONTRACTS
  REPORTS
  PHOTOS
  OTHER
}
```

2.9 Design Services

2.9.1 DesignRequest Model

```
model DesignRequest {
  id              String    @id @default(cuid())
  projectId       String
  title           String
  description     String
  designType      String?
  requirements    String?
  status          DesignRequestStatus @default(PENDING)
  priority        Priority  @default(MEDIUM)
  dueDate         DateTime?
  completedDate   DateTime?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  userId          String

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])
  user            User      @relation(fields: [userId], references: [id])
  tasks           DesignTask[]
}

enum DesignRequestStatus {
  PENDING
  IN_PROGRESS
  COMPLETED
  CANCELLED
}

model DesignTask {
  id              String    @id @default(cuid())
  designRequestId String
  taskType        String
  parameters      Json?
  status          DesignTaskStatus @default(PENDING)
  externalTaskId  String?
  resultUrl       String?
  errorMessage    String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt

  // Relationships
  designRequest   DesignRequest @relation(fields: [designRequestId], references: [id], onDelete: Cascad
}

enum DesignTaskStatus {
  PENDING
  SENT
  PROCESSING
  COMPLETED
  FAILED
}
```

2.10 Notifications & Activity

2.10.1 Notification Model

```
model Notification {
  id          String    @id @default(cuid())
  userId      String
  type        NotificationType
  title       String
  message     String
  category    NotificationCategory
  relatedId   String?   // ID of related entity (RFI, Submittal, etc.)
  relatedType String?   // Type of related entity
  isRead      Boolean   @default(false)
  readAt      DateTime?
  createdAt   DateTime  @default(now())

  // Relationships
  user        User      @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId, isRead])
  @@index([createdAt])
}

enum NotificationType {
  RFI_RESPONSE
  RFI_CREATED
  RFI_CLOSED
  SUBMITTAL_STATUS
  SUBMITTAL_CREATED
  SUBMITTAL_REVIEWED
  CHANGE_ORDER
  CHANGE_ORDER_APPROVED
  PUNCH_ITEM
  PUNCH_ITEM_COMPLETED
  DAILY_REPORT
  DOCUMENT_UPLOADED
  TIME_ENTRY_SUBMITTED
  EQUIPMENT_MAINTENANCE
  MATERIAL_ORDERED
  DESIGN_REQUEST_COMPLETED
  SYSTEM_ALERT
}

enum NotificationCategory {
  PROJECT
  DOCUMENT
  FINANCIAL
  SYSTEM
  COMMUNICATION
}

model NotificationPreference {
  id              String    @id @default(cuid())
  userId          String    @unique
  emailEnabled    Boolean   @default(true)
  inAppEnabled    Boolean   @default(true)
  rfiNotifications Boolean  @default(true)
  submittalNotifications Boolean @default(true)
  changeOrderNotifications Boolean @default(true)
  punchItemNotifications Boolean @default(true)
  dailyReportNotifications Boolean @default(true)
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt

  // Relationships
  user            User      @relation(fields: [userId], references: [id], onDelete: Cascade)
}
```

2.10.2 Activity Model

```
model Activity {
  id          String    @id @default(cuid())
  userId      String
  projectId   String?
  activityType ActivityType
  entityType  String?   // RFI, Submittal, ChangeOrder, etc.
  entityId    String?
  description String
  metadata    Json?
  createdAt   DateTime  @default(now())

  // Relationships
  user        User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  project     Project?  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  @@index([userId])
  @@index([projectId])
  @@index([createdAt])
}

enum ActivityType {
  PROJECT_CREATED
  PROJECT_UPDATED
  RFI_CREATED
  RFI_RESPONDED
  RFI_CLOSED
  SUBMITTAL_CREATED
  SUBMITTAL_REVIEWED
  SUBMITTAL_APPROVED
  CHANGE_ORDER_CREATED
  CHANGE_ORDER_APPROVED
  PUNCH_ITEM_CREATED
  PUNCH_ITEM_COMPLETED
  DAILY_REPORT_CREATED
  DOCUMENT_UPLOADED
  TIME_ENTRY_SUBMITTED
  EQUIPMENT_MAINTENANCE
  MATERIAL_ORDERED
  DESIGN_REQUEST_CREATED
}
```

2.11 Time Tracking & Labor

2.11.1 TimeEntry Model

```
model TimeEntry {
  id          String    @id @default(cuid())
  projectId   String
  userId      String
  date        DateTime
  regularHours Decimal  @db.Decimal(5, 2)
  overtimeHours Decimal @db.Decimal(5, 2) @default(0)
  hourlyRate  Decimal?  @db.Decimal(8, 2)
  totalCost   Decimal?  @db.Decimal(10, 2)
  description String?
  status      TimeEntryStatus @default(DRAFT)
  crewId      String?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  // Relationships
  project     Project   @relation(fields: [projectId], references: [id])
  user        User      @relation(fields: [userId], references: [id])
  crew        Crew?     @relation(fields: [crewId], references: [id])

  @@index([projectId])
  @@index([userId])
  @@index([date])
}

enum TimeEntryStatus {
  DRAFT
  SUBMITTED
  APPROVED
  REJECTED
}

model Crew {
  id          String    @id @default(cuid())
  name        String
  description String?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  // Relationships
  members     CrewMember[]
  timeEntries TimeEntry[]
}

model CrewMember {
  id          String    @id @default(cuid())
  crewId      String
  name        String
  role        String?
  hourlyRate  Decimal?  @db.Decimal(8, 2)
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  // Relationships
  crew        Crew      @relation(fields: [crewId], references: [id], onDelete: Cascade)
}
```

2.12 Equipment & Materials

2.12.1 Equipment Model

```
model Equipment {
  id               String    @id @default(cuid())
  name             String
  description      String?
  equipmentType    String?
  serialNumber     String?
  purchaseDate     DateTime?
  purchaseCost     Decimal?  @db.Decimal(12, 2)
  status           EquipmentStatus @default(AVAILABLE)
  ownershipType    OwnershipType @default(OWNED)
  location         String?
  projectId        String?
  createdAt        DateTime  @default(now())
  updatedAt        DateTime  @updatedAt

  // Relationships
  project          Project?  @relation(fields: [projectId], references: [id])
  maintenanceRecords MaintenanceRecord[]

  @@index([status])
  @@index([projectId])
}

enum EquipmentStatus {
  AVAILABLE
  IN_USE
  MAINTENANCE
  RETIRED
}

enum OwnershipType {
  OWNED
  RENTED
  LEASED
}

model MaintenanceRecord {
  id               String    @id @default(cuid())
  equipmentId      String
  maintenanceType String
  description     String?
  maintenanceDate DateTime
  nextDueDate     DateTime?
  cost            Decimal?  @db.Decimal(10, 2)
  performedBy     String?
  notes           String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt

  // Relationships
  equipment       Equipment @relation(fields: [equipmentId], references: [id], onDelete: Cascade)

  @@index([equipmentId])
  @@index([maintenanceDate])
}
```

2.12.2 MaterialRequisition Model

```
model MaterialRequisition {
  id              String    @id @default(cuid())
  projectId       String
  materialName    String
  description     String?
  quantity        Decimal   @db.Decimal(10, 2)
  unit            String?
  estimatedCost   Decimal?  @db.Decimal(12, 2)
  actualCost      Decimal?  @db.Decimal(12, 2)
  status          MaterialStatus @default(REQUESTED)
  requiredDate    DateTime?
  orderedDate     DateTime?
  deliveredDate   DateTime?
  vendor          String?
  notes           String?
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt

  // Relationships
  project         Project   @relation(fields: [projectId], references: [id])

  @@index([projectId])
  @@index([status])
}

enum MaterialStatus {
  REQUESTED
  ORDERED
  IN_TRANSIT
  DELIVERED
  CANCELLED
}
```

3 API Endpoints (53 total)

3.1 Authentication Endpoints

3.1.1 POST /api/auth/signup

Description: Create new user account

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123",
  "name": "John Doe",
  "role": "PROJECT_MANAGER"
}
```

Response:

```
{
  "success": true,
  "user": {
    "id": "clx123abc",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "PROJECT_MANAGER"
  }
}
```

3.1.2 POST /api/auth/signin

Description: Authenticate user and create session

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response:

```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": "clx123abc",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "PROJECT_MANAGER"
  }
}
```

3.2 Project Endpoints

3.2.1 GET /api/projects

Description: List all projects with optional filtering

Query Parameters: - `status` (optional): Filter by project status - `search` (optional): Search by name or description - `page` (optional): Page number for pagination - `limit` (optional): Items per page

Response:

```json
{
  "success": true,
  "projects": [
    {
      "id": "clx123abc",
      "name": "Downtown Office Building",
      "description": "15-story commercial building",
      "location": "123 Main St, City, State",
      "status": "ACTIVE",
      "budget": 5000000.00,
      "startDate": "2026-01-01T00:00:00Z",
      "endDate": "2026-12-31T00:00:00Z",
      "createdAt": "2025-12-01T00:00:00Z"
    }
  ],
  "pagination": {
    "total": 25,
    "page": 1,
    "limit": 10,
    "pages": 3
  }
}
```

3.2.2 POST /api/projects

Description: Create new project

Request Body:

```json
{
  "name": "Downtown Office Building",
  "description": "15-story commercial building",
  "location": "123 Main St, City, State",
  "budget": 5000000.00,
  "startDate": "2026-01-01",
  "endDate": "2026-12-31",
  "status": "PLANNING",
  "clientName": "ABC Corporation",
  "projectManager": "John Doe"
}
```

3.2.3 GET /api/projects/[id]

Description: Get single project details

Response:

```json
{
  "success": true,
  "project": {
    "id": "clx123abc",
    "name": "Downtown Office Building",
    "description": "15-story commercial building",
    "location": "123 Main St, City, State",
    "status": "ACTIVE",
    "budget": 5000000.00,
    "startDate": "2026-01-01T00:00:00Z",
    "endDate": "2026-12-31T00:00:00Z",
    "rfis": [],
    "submittals": [],
    "changeOrders": [],
    "punchItems": []
  }
}
```

3.2.4 PATCH /api/projects/[id]

Description: Update project details

Request Body:

```
{
  "status": "ACTIVE",
  "budget": 5200000.00
}
```

3.3 RFI Endpoints

3.3.1 GET /api/rfis

Description: List all RFIs with filtering

Query Parameters: - `projectId` (optional): Filter by project - `status` (optional): Filter by status - `priority` (optional): Filter by priority

Response:

```
{
  "success": true,
  "rfis": [
    {
      "id": "clx456def",
      "projectId": "clx123abc",
      "subject": "Clarification on structural beam size",
      "description": "Need confirmation on beam size for grid line A-5",
      "status": "SUBMITTED",
      "priority": "HIGH",
      "dueDate": "2026-01-15T00:00:00Z",
      "createdAt": "2026-01-08T00:00:00Z",
      "responses": []
    }
  ]
}
```

3.3.2 POST /api/rfis

Description: Create new RFI

Request Body:

```
{
  "projectId": "clx123abc",
  "subject": "Clarification on structural beam size",
  "description": "Need confirmation on beam size for grid line A-5",
  "priority": "HIGH",
  "dueDate": "2026-01-15"
}
```

3.3.3 GET /api/rfis/[id]

Description: Get single RFI with responses and comments

3.3.4 PATCH /api/rfis/[id]

Description: Update RFI status or details

3.3.5 POST /api/rfis/[id]/responses

Description: Add response to RFI

Request Body:

```
{
  "response": "The beam size should be W18x50 as per structural drawings sheet S-3",
  "respondedBy": "Jane Smith, Structural Engineer"
}
```

3.3.6 POST /api/rfis/[id]/comments

Description: Add comment to RFI

3.4 Submittal Endpoints

3.4.1 GET /api/submittals

Description: List all submittals with filtering

Query Parameters: - `projectId` (optional): Filter by project - `status` (optional): Filter by status - `type` (optional): Filter by submittal type

3.4.2 POST /api/submittals

Description: Create new submittal

Request Body:

```
{
  "projectId": "clx123abc",
  "submittalNumber": "S-001",
  "title": "Structural Steel Shop Drawings",
  "description": "Shop drawings for structural steel fabrication",
  "type": "SHOP_DRAWINGS",
  "specSection": "05 12 00",
  "requiredDate": "2026-02-01"
}
```

3.4.3 GET /api/submittals/[id]

Description: Get single submittal with responses

3.4.4 PATCH /api/submittals/[id]

Description: Update submittal status or details

3.4.5 POST /api/submittals/[id]/responses

Description: Add review response to submittal

Request Body:

```json
{
  "response": "Approved with minor corrections noted in red",
  "reviewedBy": "John Architect, AIA",
  "decision": "APPROVED_AS_NOTED"
}
```

3.5 Change Order Endpoints

3.5.1 GET /api/change-orders

Description: List all change orders

Query Parameters: - `projectId` (optional): Filter by project - `status` (optional): Filter by status

Response:

```json
{
  "success": true,
  "changeOrders": [
    {
      "id": "clx789ghi",
      "projectId": "clx123abc",
      "changeOrderNumber": "CO-001",
      "title": "Additional HVAC Units",
      "description": "Add two additional rooftop HVAC units",
      "status": "PENDING",
      "originalCost": 0.00,
      "proposedCost": 45000.00,
      "scheduleImpact": 7,
      "createdAt": "2026-01-08T00:00:00Z"
    }
  ]
}
```

3.5.2 POST /api/change-orders

Description: Create new change order

Request Body:

```json
{
  "projectId": "clx123abc",
  "changeOrderNumber": "CO-001",
  "title": "Additional HVAC Units",
  "description": "Add two additional rooftop HVAC units",
  "reason": "Owner requested additional cooling capacity",
  "proposedCost": 45000.00,
  "scheduleImpact": 7
}
```

3.5.3 GET /api/change-orders/[id]

Description: Get single change order details

3.5.4 PATCH /api/change-orders/[id]

Description: Update change order status or costs

3.6 Punch Item Endpoints

3.6.1 GET /api/punch-items

Description: List all punch items

Query Parameters: - `projectId` (optional): Filter by project - `status` (optional): Filter by status - `assignedTo` (optional): Filter by assignee

3.6.2 POST /api/punch-items

Description: Create new punch item

Request Body:

```
{
  "projectId": "clx123abc",
  "itemNumber": "P-001",
  "description": "Touch up paint on wall in Room 201",
  "location": "Room 201, 2nd Floor",
  "assignedTo": "ABC Painting Contractors",
  "priority": "MEDIUM",
  "dueDate": "2026-01-20"
}
```

3.6.3 GET /api/punch-items/[id]

Description: Get single punch item details

3.6.4 PATCH /api/punch-items/[id]

Description: Update punch item status or details

3.7 Daily Report Endpoints

3.7.1 GET /api/daily-reports

Description: List all daily reports

Query Parameters: - `projectId` (optional): Filter by project - `startDate` (optional): Filter by date range start - `endDate` (optional): Filter by date range end

3.7.2 POST /api/daily-reports

Description: Create new daily report

Request Body:

```
{
  "projectId": "clx123abc",
  "reportDate": "2026-01-08",
  "weather": "Partly Cloudy",
  "temperature": "45°F",
  "workPerformed": "Continued concrete pour for foundation. Installed rebar for east wall.",
  "crewSize": 12,
  "equipmentUsed": "Concrete pump, 2 excavators, 1 crane",
  "materialsDelivered": "50 cubic yards concrete, 2 tons rebar",
  "delays": "None"
}
```

3.7.3 GET /api/daily-reports/[id]

Description: Get single daily report details

3.8 Document Endpoints

3.8.1 GET /api/documents

Description: List all documents

Query Parameters: - `projectId` (optional): Filter by project - `category` (optional): Filter by document category

3.8.2 POST /api/documents

Description: Upload new document

Request Body (multipart/form-data): - `file`: File to upload - `projectId`: Project ID - `title`: Document title - `description`: Document description - `category`: Document category - `isPublic`: Public access flag

Response:

```
{
  "success": true,
  "document": {
    "id": "clx999jkl",
    "title": "Site Plan Rev 3",
    "fileName": "site-plan-rev3.pdf",
    "fileSize": 2048576,
    "fileType": "application/pdf",
    "category": "DRAWINGS",
    "cloudStoragePath": "projects/clx123abc/documents/site-plan-rev3.pdf",
    "uploadDate": "2026-01-08T00:00:00Z"
  }
}
```

3.8.3 GET /api/documents/[id]

Description: Get document details and download URL

3.8.4 DELETE /api/documents/[id]

Description: Delete document

3.9 Notification Endpoints

3.9.1 GET /api/notifications

Description: Get user notifications

Query Parameters: - `isRead` (optional): Filter by read status (true/false) - `category` (optional): Filter by notification category - `limit` (optional): Number of notifications to return

Response:

```json
{
  "success": true,
  "notifications": [
    {
      "id": "clx111mno",
      "type": "RFI_RESPONSE",
      "title": "RFI Response Received",
      "message": "Your RFI 'Clarification on structural beam size' has received a response",
      "category": "PROJECT",
      "relatedId": "clx456def",
      "relatedType": "RFI",
      "isRead": false,
      "createdAt": "2026-01-08T10:30:00Z"
    }
  ],
  "unreadCount": 5
}
```

3.9.2 POST /api/notifications/[id]/read

Description: Mark notification as read

Response:

```json
{
  "success": true,
  "notification": {
    "id": "clx111mno",
    "isRead": true,
    "readAt": "2026-01-08T11:00:00Z"
  }
}
```

3.9.3 POST /api/notifications/mark-all-read

Description: Mark all notifications as read

3.9.4 GET /api/notifications/preferences

Description: Get user notification preferences

Response:

```json
{
  "success": true,
  "preferences": {
    "emailEnabled": true,
    "inAppEnabled": true,
    "rfiNotifications": true,
    "submittalNotifications": true,
    "changeOrderNotifications": true,
    "punchItemNotifications": true,
    "dailyReportNotifications": false
  }
}
```

3.9.5 PATCH /api/notifications/preferences

Description: Update notification preferences

Request Body:

```
{
  "emailEnabled": true,
  "rfiNotifications": true,
  "submittalNotifications": false
}
```

3.10 Activity Endpoints

3.10.1 GET /api/activities

Description: Get activity feed

Query Parameters: - `projectId` (optional): Filter by project - `userId` (optional): Filter by user - `activityType` (optional): Filter by activity type - `limit` (optional): Number of activities to return

Response:

```
{
  "success": true,
  "activities": [
    {
      "id": "clx222pqr",
      "activityType": "RFI_CREATED",
      "description": "John Doe created RFI 'Clarification on structural beam size'",
      "entityType": "RFI",
      "entityId": "clx456def",
      "createdAt": "2026-01-08T09:00:00Z",
      "user": {
        "name": "John Doe"
      }
    }
  ]
}
```

3.11 Analytics Endpoints

3.11.1 GET /api/analytics/metrics

Description: Get project health and performance metrics

Query Parameters: - `projectId` (optional): Filter by specific project - `dateRange` (optional): Date range (7, 30, 90, 365 days)

Response:

```
{
  "success": true,
  "metrics": {
    "projectHealth": {
      "activeProjects": 12,
      "onTimePercentage": 85.5,
      "budgetVariance": -2.3,
      "criticalItems": 3
    },
    "rfiMetrics": {
      "totalRFIs": 45,
      "openRFIs": 8,
      "averageResponseTime": 3.2,
      "overdueRFIs": 2
    },
    "submittalMetrics": {
      "totalSubmittals": 67,
      "pendingSubmittals": 12,
      "approvalRate": 78.5,
      "averageReviewTime": 5.8
    },
    "changeOrderMetrics": {
      "totalChangeOrders": 15,
      "totalValue": 245000.00,
      "approvedValue": 180000.00,
      "pendingValue": 65000.00
    }
  }
}
```

3.11.2 GET /api/analytics/trends

Description: Get time-series trend data

Query Parameters: - `projectId` (optional): Filter by project - `metric` (required): Metric to analyze (rfis, submittals, changeOrders, etc.) - `dateRange` (optional): Date range for analysis

Response:

```
{
  "success": true,
  "trends": {
    "metric": "rfis",
    "data": [
      {
        "date": "2026-01-01",
        "created": 5,
        "closed": 3,
        "open": 12
      },
      {
        "date": "2026-01-02",
        "created": 2,
        "closed": 4,
        "open": 10
      }
    ]
  }
}
```

3.12 Time Tracking Endpoints

3.12.1 GET /api/time-tracking

Description: Get time entries

Query Parameters: - `projectId` (optional): Filter by project - `userId` (optional): Filter by user - `startDate` (optional): Filter by date range - `endDate` (optional): Filter by date range - `status` (optional): Filter by status

3.12.2 POST /api/time-tracking

Description: Create time entry

Request Body:

```
{
  "projectId": "clx123abc",
  "date": "2026-01-08",
  "regularHours": 8.0,
  "overtimeHours": 2.0,
  "hourlyRate": 45.00,
  "description": "Foundation work and rebar installation",
  "crewId": "clx333stu"
}
```

3.12.3 GET /api/time-tracking/[id]

Description: Get single time entry

3.12.4 PATCH /api/time-tracking/[id]

Description: Update time entry or change status

3.13 Equipment Endpoints

3.13.1 GET /api/equipment

Description: List all equipment

Query Parameters: - `status` (optional): Filter by equipment status - `projectId` (optional): Filter by assigned project - `ownershipType` (optional): Filter by ownership type

Response:

```
{
  "success": true,
  "equipment": [
    {
      "id": "clx444vwx",
      "name": "Excavator CAT 320",
      "equipmentType": "Heavy Equipment",
      "serialNumber": "CAT320-12345",
      "status": "IN_USE",
      "ownershipType": "OWNED",
      "projectId": "clx123abc",
      "location": "Downtown Office Building Site"
    }
  ]
}
```

3.13.2 POST /api/equipment

Description: Add new equipment

Request Body:

```
{
  "name": "Excavator CAT 320",
  "description": "320 model excavator with 1.2 cubic yard bucket",
  "equipmentType": "Heavy Equipment",
  "serialNumber": "CAT320-12345",
  "purchaseDate": "2024-06-15",
  "purchaseCost": 185000.00,
  "status": "AVAILABLE",
  "ownershipType": "OWNED"
}
```

3.13.3 GET /api/equipment/[id]

Description: Get equipment details with maintenance history

3.13.4 PATCH /api/equipment/[id]

Description: Update equipment details or status

3.13.5 POST /api/equipment/[id]/maintenance

Description: Add maintenance record

Request Body:

```
{
  "maintenanceType": "Preventive Maintenance",
  "description": "Oil change and filter replacement",
  "maintenanceDate": "2026-01-08",
  "nextDueDate": "2026-04-08",
  "cost": 450.00,
  "performedBy": "ABC Equipment Services"
}
```

3.14 Material Endpoints

3.14.1 GET /api/materials

Description: List material requisitions

Query Parameters: - `projectId` (optional): Filter by project - `status` (optional): Filter by status

Response:

```
{
  "success": true,
  "materials": [
    {
      "id": "clx555yza",
      "projectId": "clx123abc",
      "materialName": "Concrete - 4000 PSI",
      "quantity": 150.00,
      "unit": "cubic yards",
      "estimatedCost": 18000.00,
      "status": "ORDERED",
      "requiredDate": "2026-01-15T00:00:00Z",
      "vendor": "ABC Concrete Supply"
    }
  ]
}
```

3.14.2 POST /api/materials

Description: Create material requisition

Request Body:

```
{
  "projectId": "clx123abc",
  "materialName": "Concrete - 4000 PSI",
  "description": "Ready-mix concrete for foundation",
  "quantity": 150.00,
  "unit": "cubic yards",
  "estimatedCost": 18000.00,
  "requiredDate": "2026-01-15",
  "vendor": "ABC Concrete Supply"
}
```

3.14.3 GET /api/materials/[id]

Description: Get material requisition details

3.14.4 PATCH /api/materials/[id]

Description: Update material requisition status

3.15 Webhook Endpoints

3.15.1 POST /api/webhooks/change-orders

Description: Webhook endpoint for n8n change order automation

Headers: - `x-webhook-signature`: HMAC signature for verification

Request Body:

```
{
  "action": "approve",
  "changeOrderId": "clx789ghi",
  "approvedBy": "Jane Smith",
  "approvedCost": 45000.00,
  "notes": "Approved by owner"
}
```

Response:

```
{
  "success": true,
  "message": "Change order updated successfully"
}
```

3.15.2 POST /api/webhooks/design-callback

Description: Callback endpoint for design service results

Headers: - `x-design-signature`: HMAC signature for verification

Request Body:

```
{
  "taskId": "clx666bcd",
  "status": "completed",
  "resultUrls": [
    "https://i.ytimg.com/vi/OwyMOmhmBNO/hqdefault.jpg?v=6627ecca",
    "https://i.ytimg.com/vi/nvHs8Z6hraQ/sddefault.jpg"
  ],
  "metadata": {
    "resolution": "4K",
    "format": "JPEG",
    "processingTime": 180
  }
}
```

Response:

```
{
  "success": true,
  "message": "Design results processed successfully"
}
```

4 Key Libraries & Dependencies

4.1 Core Dependencies

4.1.1 Frontend Libraries

React & Next.js

```json
{
  "next": "14.2.28",
  "react": "18.2.0",
  "react-dom": "18.2.0"
}
```

TypeScript

```json
{
  "typescript": "^5.0.0",
  "@types/react": "^18.2.0",
  "@types/node": "^20.0.0"
}
```

4.1.2 UI Component Libraries

Radix UI

```json
{
  "@radix-ui/react-dialog": "^1.0.5",
  "@radix-ui/react-dropdown-menu": "^2.0.6",
  "@radix-ui/react-label": "^2.0.2",
  "@radix-ui/react-select": "^2.0.0",
  "@radix-ui/react-tabs": "^1.0.4",
  "@radix-ui/react-toast": "^1.1.5",
  "@radix-ui/react-popover": "^1.0.7"
}
```

Tailwind CSS

```json
{
  "tailwindcss": "^3.4.0",
  "tailwind-merge": "^2.2.0",
  "tailwindcss-animate": "^1.0.7"
}
```

4.1.3 Data Visualization

Chart.js

```json
{
  "chart.js": "4.4.9",
  "react-chartjs-2": "^5.2.0"
}
```

Features: - Line charts for trend analysis - Bar charts for comparative data - Doughnut charts for distributions - Responsive and interactive - Custom tooltips and legends - Animation support

4.1.4 Form Management

React Hook Form

```json
{
  "react-hook-form": "^7.50.0"
}
```

Features: - Performance-optimized form handling - Built-in validation - TypeScript support - Minimal re-renders - Easy integration with UI libraries

Zod Validation

```json
{
  "zod": "^3.22.0",
  "@hookform/resolvers": "^3.3.0"
}
```

Features: - TypeScript-first schema validation - Runtime type checking - Composable schemas - Custom error messages - Integration with React Hook Form

4.1.5 Database & ORM

Prisma

```json
{
  "prisma": "6.7.0",
  "@prisma/client": "6.7.0"
}
```

Features: - Type-safe database client - Auto-generated types - Migration management - Query optimization - Connection pooling - Middleware support

4.1.6 Authentication

NextAuth.js

```json
{
  "next-auth": "^4.24.0"
}
```

Features: - Multiple authentication providers - JWT and database sessions - Built-in CSRF protection - TypeScript support - Customizable callbacks - Role-based access control

4.1.7 File Storage

AWS SDK v3

```json
{
  "@aws-sdk/client-s3": "^3.500.0",
  "@aws-sdk/s3-request-presigner": "^3.500.0"
}
```

Features: - S3 file upload/download - Presigned URL generation - Multipart upload support - Stream handling - Error handling and retries

4.1.8 Date & Time

date-fns

```json
{
  "date-fns": "^3.0.0"
}
```

Features: - Lightweight date manipulation - Immutable operations - Tree-shakeable - TypeScript support - Timezone handling

4.1.9 Utilities

clsx

```json
{
  "clsx": "^2.1.0"
}
```

Features: - Conditional className construction - Lightweight (< 1KB) - TypeScript support

lucide-react

```json
{
  "lucide-react": "^0.344.0"
}
```

Features: - Modern icon library - Tree-shakeable - Customizable size and color - 1000+ icons

4.2 Development Dependencies

4.2.1 Code Quality

ESLint

```json
{
  "eslint": "^8.56.0",
  "eslint-config-next": "14.2.28"
}
```

Prettier

```json
{
  "prettier": "^3.2.0",
  "prettier-plugin-tailwindcss": "^0.5.0"
}
```

4.2.2 Build Tools

PostCSS

```json
{
  "postcss": "^8.4.0",
  "autoprefixer": "^10.4.0"
}
```

4.2.3 Testing (Planned)

Jest

```json
{
  "jest": "^29.7.0",
  "@testing-library/react": "^14.0.0",
  "@testing-library/jest-dom": "^6.0.0"
}
```

Playwright

```json
{
  "@playwright/test": "^1.40.0"
}
```

5 Authentication & Security

5.1 NextAuth.js Configuration

5.1.1 JWT Strategy

BuildOS uses JSON Web Tokens (JWT) for stateless authentication:

Configuration:

```
// lib/auth.ts
import NextAuth from "next-auth"
import CredentialsProvider from "next-auth/providers/credentials"
import { PrismaAdapter } from "@next-auth/prisma-adapter"
import { prisma } from "@/lib/prisma"
import bcrypt from "bcryptjs"

export const authOptions = {
  adapter: PrismaAdapter(prisma),
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "email" },
        password: { label: "Password", type: "password" }
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          return null
        }

        const user = await prisma.user.findUnique({
          where: { email: credentials.email }
        })

        if (!user) {
          return null
        }

        const isPasswordValid = await bcrypt.compare(
          credentials.password,
          user.password
        )

        if (!isPasswordValid) {
          return null
        }

        return {
          id: user.id,
          email: user.email,
          name: user.name,
          role: user.role
        }
      }
    })
  ],
  session: {
    strategy: "jwt",
    maxAge: 30 * 24 * 60 * 60, // 30 days
  },
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id
        token.role = user.role
      }
      return token
    },
    async session({ session, token }) {
      if (session.user) {
        session.user.id = token.id
        session.user.role = token.role
      }
      return session
    }
  },
  pages: {
    signIn: "/login",
    signOut: "/logout",
    error: "/auth/error",
  }
}
```

5.1.2 Password Security

Hashing: - Algorithm: bcrypt - Salt rounds: 10 - Passwords never stored in plain text - Password strength requirements enforced

Password Requirements: - Minimum 8 characters - At least one uppercase letter - At least one lowercase letter - At least one number - At least one special character

5.2 Role-Based Access Control (RBAC)

5.2.1 Middleware Protection

Route Protection:

```
// middleware.ts
import { withAuth } from "next-auth/middleware"
import { NextResponse } from "next/server"

export default withAuth(
  function middleware(req) {
    const token = req.nextauth.token
    const path = req.nextUrl.pathname

    // Admin-only routes
    if (path.startsWith("/admin") && token?.role !== "ADMIN") {
      return NextResponse.redirect(new URL("/unauthorized", req.url))
    }
    // Project manager routes
    if (path.startsWith("/projects/new") &&
        !["ADMIN", "PROJECT_MANAGER"].includes(token?.role)) {
      return NextResponse.redirect(new URL("/unauthorized", req.url))
    }

    return NextResponse.next()
  },
  {
    callbacks: {
      authorized: ({ token }) => !!token
    }
  }
)

export const config = {
  matcher: [
    "/dashboard/:path*",
    "/projects/:path*",
    "/rfis/:path*",
    "/submittals/:path*",
    "/admin/:path*"
  ]
}
```

5.2.2 API Route Protection

Authorization Helper:

```
// lib/auth-helpers.ts
import { getServerSession } from "next-auth"
import { authOptions } from "@/lib/auth"

export async function requireAuth() {
  const session = await getServerSession(authOptions)

  if (!session) {
    throw new Error("Unauthorized")
  }

  return session
}

export async function requireRole(allowedRoles: string[]) {
  const session = await requireAuth()

  if (!allowedRoles.includes(session.user.role)) {
    throw new Error("Forbidden")
  }

  return session
}
```

Usage in API Routes:

```
// app/api/projects/route.ts
import { requireRole } from "@/lib/auth-helpers"

export async function POST(request: Request) {
  try {
    // Only admins and project managers can create projects
    const session = await requireRole(["ADMIN", "PROJECT_MANAGER"])

    const body = await request.json()
    // ... create project logic

  } catch (error) {
    return NextResponse.json(
      { error: error.message },
      { status: error.message === "Unauthorized" ? 401 : 403 }
    )
  }
}
```

5.3 Webhook Signature Verification

5.3.1 HMAC Signature Validation

Change Orders Webhook:

```
// app/api/webhooks/change-orders/route.ts
import crypto from "crypto"

const WEBHOOK_SECRET = process.env.N8N_CHANGE_ORDER_WEBHOOK_SECRET!

function verifySignature(payload: string, signature: string): boolean {
  const expectedSignature = crypto
    .createHmac("sha256", WEBHOOK_SECRET)
    .update(payload)
    .digest("hex")

  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expectedSignature)
  )
}

export async function POST(request: Request) {
  const signature = request.headers.get("x-webhook-signature")

  if (!signature) {
    return NextResponse.json(
      { error: "Missing signature" },
      { status: 401 }
    )
  }

  const payload = await request.text()

  if (!verifySignature(payload, signature)) {
    return NextResponse.json(
      { error: "Invalid signature" },
      { status: 401 }
    )
  }

  // Process webhook...
}
```

Design Services Webhook:

```
// app/api/webhooks/design-callback/route.ts
const DESIGN_SECRET = process.env.DESIGN_WEBHOOK_SECRET!

function verifyDesignSignature(payload: string, signature: string): boolean {
  const expectedSignature = crypto
    .createHmac("sha256", DESIGN_SECRET)
    .update(payload)
    .digest("hex")

  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expectedSignature)
  )
}
```

5.4 Data Security

5.4.1 Input Validation

Zod Schemas:

```
// lib/validations/project.ts
import { z } from "zod"

export const createProjectSchema = z.object({
  name: z.string().min(1, "Project name is required").max(200),
  description: z.string().optional(),
  location: z.string().optional(),
  budget: z.number().positive().optional(),
  startDate: z.string().datetime().optional(),
  endDate: z.string().datetime().optional(),
  status: z.enum(["PLANNING", "ACTIVE", "ON_HOLD", "COMPLETED", "CANCELLED"]),
  clientName: z.string().optional(),
  projectManager: z.string().optional()
})

export type CreateProjectInput = z.infer<typeof createProjectSchema>
```

5.4.2 SQL Injection Prevention

- **Prisma ORM:** Parameterized queries prevent SQL injection
- **No raw SQL:** All queries use Prisma's type-safe API
- **Input sanitization:** All user inputs validated before database operations

5.4.3 XSS Prevention

- **React:** Automatic escaping of user content
- **Content Security Policy:** Restrictive CSP headers
- **Sanitization:** HTML content sanitized before rendering

5.4.4 CSRF Protection

- **NextAuth.js:** Built-in CSRF token validation
- **SameSite Cookies:** Cookies set with SameSite=Lax
- **Token Validation:** CSRF tokens validated on state-changing operations

5.5 Environment Variables Security

5.5.1 Required Environment Variables

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/buildos"

# Authentication
NEXTAUTH_SECRET="your-secret-key-here"
NEXTAUTH_URL="http://localhost:3000"

# AWS S3
AWS_BUCKET_NAME="buildos-documents"
AWS_FOLDER_PREFIX="production/"
AWS_ACCESS_KEY_ID="your-access-key"
AWS_SECRET_ACCESS_KEY="your-secret-key"
AWS_REGION="us-east-1"

# Webhooks
N8N_CHANGE_ORDER_WEBHOOK_SECRET="34d7412dfa4d1c54106d4c5129c6a312061a3d8b50966dbe6000124cbece9890"
DESIGN_WEBHOOK_URL="https://gmllorlxfsxmsejhsjpa.supabase.co/functions/v1/n8n-orders-webhook"
DESIGN_WEBHOOK_SECRET="31851db11bdbfef9a8f5e433769d75a0416d9f922253089b1c08619ad70df2f7"

# Email (Optional)
SMTP_HOST="smtp.example.com"
SMTP_PORT="587"
SMTP_USER="notifications@buildos.com"
SMTP_PASSWORD="your-smtp-password"
```

5.5.2 Environment Variable Management

- **Never commit:** .env files excluded from version control
- **Separate environments:** Different values for dev/staging/production
- **Rotation:** Regular rotation of secrets and API keys
- **Access control:** Limited access to production environment variables

6 File Storage

6.1 AWS S3 Integration

6.1.1 Configuration

S3 Client Setup:

```
// lib/s3.ts
import { S3Client } from "@aws-sdk/client-s3"

export const s3Client = new S3Client({
  region: process.env.AWS_REGION!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!
  }
})

export const BUCKET_NAME = process.env.AWS_BUCKET_NAME!
export const FOLDER_PREFIX = process.env.AWS_FOLDER_PREFIX || ""
```

6.1.2 File Upload

Presigned URL Generation:

```
// lib/s3-upload.ts
import { PutObjectCommand } from "@aws-sdk/client-s3"
import { getSignedUrl } from "@aws-sdk/s3-request-presigner"
import { s3Client, BUCKET_NAME, FOLDER_PREFIX } from "./s3"

export async function generateUploadUrl(
  fileName: string,
  fileType: string,
  projectId: string
): Promise<{ uploadUrl: string; fileKey: string }> {
  const fileKey = `${FOLDER_PREFIX}projects/${projectId}/documents/${Date.now()}-${fileName}`

  const command = new PutObjectCommand({
    Bucket: BUCKET_NAME,
    Key: fileKey,
    ContentType: fileType
  })

  const uploadUrl = await getSignedUrl(s3Client, command, {
    expiresIn: 3600 // 1 hour
  })

  return { uploadUrl, fileKey }
}
```

Client-Side Upload:

```
// Client component
async function uploadFile(file: File, projectId: string) {
  // Get presigned URL from API
  const response = await fetch("/api/documents/upload-url", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      fileName: file.name,
      fileType: file.type,
      projectId
    })
  })

  const { uploadUrl, fileKey } = await response.json()

  // Upload directly to S3
  await fetch(uploadUrl, {
    method: "PUT",
    body: file,
    headers: {
      "Content-Type": file.type
    }
  })

  // Save document record in database
  await fetch("/api/documents", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      projectId,
      title: file.name,
      fileName: file.name,
      fileSize: file.size,
      fileType: file.type,
      cloudStoragePath: fileKey
    })
  })
}
```

6.1.3 File Download

Presigned Download URL:

```
// lib/s3-download.ts
import { GetObjectCommand } from "@aws-sdk/client-s3"
import { getSignedUrl } from "@aws-sdk/s3-request-presigner"
import { s3Client, BUCKET_NAME } from "./s3"

export async function generateDownloadUrl(
  fileKey: string,
  expiresIn: number = 3600
): Promise<string> {
  const command = new GetObjectCommand({
    Bucket: BUCKET_NAME,
    Key: fileKey
  })

  return await getSignedUrl(s3Client, command, { expiresIn })
}
```

API Endpoint:

```
// app/api/documents/[id]/download/route.ts
import { generateDownloadUrl } from "@/lib/s3-download"
import { prisma } from "@/lib/prisma"
import { requireAuth } from "@/lib/auth-helpers"

export async function GET(
  request: Request,
  { params }: { params: { id: string } }
) {
  try {
    await requireAuth()

    const document = await prisma.document.findUnique({
      where: { id: params.id }
    })

    if (!document) {
      return NextResponse.json(
        { error: "Document not found" },
        { status: 404 }
      )
    }

    const downloadUrl = await generateDownloadUrl(
      document.cloudStoragePath!
    )

    return NextResponse.json({ downloadUrl })

  } catch (error) {
    return NextResponse.json(
      { error: "Failed to generate download URL" },
      { status: 500 }
    )
  }
}
```

6.1.4 Public vs Private Files

Public Files: - Accessible without authentication - Used for: Public project images, marketing materials - S3 bucket policy allows public read access to specific prefixes

Private Files: - Require authentication to access - Used for: Contracts, financial documents, internal reports - Access controlled via presigned URLs with short expiration

Implementation:

```
// lib/s3-access.ts
export function getFileAccessLevel(category: string): "public" | "private" {
  const publicCategories = ["PHOTOS"]
  return publicCategories.includes(category) ? "public" : "private"
}

export function getS3Path(
  projectId: string,
  category: string,
  fileName: string,
  isPublic: boolean
): string {
  const prefix = isPublic ? "public/" : "private/"
  return `${FOLDER_PREFIX}${prefix}projects/${projectId}/${category.toLowerCase()}/${fileName}`
}
```

6.1.5 Cloud Storage Path Management

Organized Folder Structure:

```
buildos-documents/
   production/
       public/
           projects/
               {projectId}/
                   photos/
                   drawings/
       private/
           projects/
               {projectId}/
                   contracts/
                   reports/
                   specifications/
   staging/
       ... (same structure)
```

Benefits: - Easy to locate files - Simple backup and archival - Clear separation of public/private content - Project-based organization - Environment isolation

6.1.6 File Lifecycle Management

S3 Lifecycle Policies:

```
{
  "Rules": [
    {
      "Id": "ArchiveOldDocuments",
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 90,
          "StorageClass": "STANDARD_IA"
        },
        {
          "Days": 365,
          "StorageClass": "GLACIER"
        }
      ]
    },
    {
      "Id": "DeleteTempFiles",
      "Status": "Enabled",
      "Prefix": "temp/",
      "Expiration": {
        "Days": 7
      }
    }
  ]
}
```

Benefits: - Cost optimization through storage tiering - Automatic archival of old files - Cleanup of temporary files - Compliance with retention policies

7 Notification System Architecture

7.1 System Overview

The BuildOS notification system provides real-time updates to users about important project events through multiple channels.

7.1.1 Notification Types

18 Notification Types:

1. **RFI_RESPONSE** - RFI received a response
2. **RFI_CREATED** - New RFI created
3. **RFI_CLOSED** - RFI marked as closed
4. **SUBMITTAL_STATUS** - Submittal status changed
5. **SUBMITTAL_CREATED** - New submittal submitted
6. **SUBMITTAL_REVIEWED** - Submittal review completed
7. **CHANGE_ORDER** - Change order created or updated
8. **CHANGE_ORDER_APPROVED** - Change order approved
9. **PUNCH_ITEM** - Punch item assigned or updated
10. **PUNCH_ITEM_COMPLETED** - Punch item marked complete
11. **DAILY_REPORT** - Daily report submitted
12. **DOCUMENT_UPLOADED** - New document uploaded
13. **TIME_ENTRY_SUBMITTED** - Time entry submitted for approval
14. **EQUIPMENT_MAINTENANCE** - Equipment maintenance scheduled
15. **MATERIAL_ORDERED** - Material requisition ordered
16. **DESIGN_REQUEST_COMPLETED** - Design request completed
17. **SYSTEM_ALERT** - System-level notifications
18. **COMMUNICATION** - General communication messages

7.1.2 Notification Categories

5 Categories for Organization:

- **PROJECT** - Project-related notifications
- **DOCUMENT** - Document and file notifications
- **FINANCIAL** - Budget and cost notifications
- **SYSTEM** - System and administrative notifications
- **COMMUNICATION** - Messages and updates

7.2 In-App Notifications

7.2.1 Notification Creation

Service Function:

```
// lib/notifications.ts
import { prisma } from "./prisma"
import { NotificationType, NotificationCategory } from "@prisma/client"

export async function createNotification({
  userId,
  type,
  title,
  message,
  category,
  relatedId,
  relatedType
}: {
  userId: string
  type: NotificationType
  title: string
  message: string
  category: NotificationCategory
  relatedId?: string
  relatedType?: string
}) {
  return await prisma.notification.create({
    data: {
      userId,
      type,
      title,
      message,
      category,
      relatedId,
      relatedType,
      isRead: false
    }
  })
}
```

Usage Example:

```
// When RFI receives a response
await createNotification({
  userId: rfi.userId,
  type: "RFI_RESPONSE",
  title: "RFI Response Received",
  message: `Your RFI "${rfi.subject}" has received a response`,
  category: "PROJECT",
  relatedId: rfi.id,
  relatedType: "RFI"
})
```

7.2.2 Notification Retrieval

API Endpoint:

```
// app/api/notifications/route.ts
export async function GET(request: Request) {
  const session = await requireAuth()
  const { searchParams } = new URL(request.url)

  const isRead = searchParams.get("isRead")
  const category = searchParams.get("category")
  const limit = parseInt(searchParams.get("limit") || "50")

  const notifications = await prisma.notification.findMany({
    where: {
      userId: session.user.id,
      ...(isRead !== null && { isRead: isRead === "true" }),
      ...(category && { category })
    },
    orderBy: { createdAt: "desc" },
    take: limit
  })

  const unreadCount = await prisma.notification.count({
    where: {
      userId: session.user.id,
      isRead: false
    }
  })

  return NextResponse.json({
    success: true,
    notifications,
    unreadCount
  })
}
```

7.2.3 Real-Time Polling

Client-Side Implementation:

```
// hooks/useNotifications.ts
import { useEffect, useState } from "react"

export function useNotifications() {
  const [notifications, setNotifications] = useState([])
  const [unreadCount, setUnreadCount] = useState(0)

  useEffect(() => {
    // Initial fetch
    fetchNotifications()

    // Poll every 30 seconds
    const interval = setInterval(fetchNotifications, 30000)

    return () => clearInterval(interval)
  }, [])

  async function fetchNotifications() {
    const response = await fetch("/api/notifications?limit=20")
    const data = await response.json()

    setNotifications(data.notifications)
    setUnreadCount(data.unreadCount)
  }

  async function markAsRead(notificationId: string) {
    await fetch(`/api/notifications/${notificationId}/read`, {
      method: "POST"
    })

    fetchNotifications()
  }
  async function markAllAsRead() {
    await fetch("/api/notifications/mark-all-read", {
      method: "POST"
    })

    fetchNotifications()
  }

  return {
    notifications,
    unreadCount,
    markAsRead,
    markAllAsRead,
    refresh: fetchNotifications
  }
}
```

7.2.4 Notification UI Component

Bell Icon with Badge:

```
// components/NotificationBell.tsx
import { Bell } from "lucide-react"
import { useNotifications } from "@/hooks/useNotifications"

export function NotificationBell() {
  const { notifications, unreadCount, markAsRead } = useNotifications()

  return (
    <Popover>
      <PopoverTrigger>
        <div className="relative">
          <Bell className="h-6 w-6" />
          {unreadCount > 0 && (
            <span className="absolute -top-1 -right-1 bg-red-500 text-white text-xs rounded-full h-5 w-
              {unreadCount}
            </span>
          )}
        </div>
      </PopoverTrigger>
      <PopoverContent className="w-96">
        <div className="space-y-2">
          <h3 className="font-semibold">Notifications</h3>
          {notifications.map((notification) => (
            <div
              key={notification.id}
              className={`p-3 rounded ${
                notification.isRead ? "bg-gray-50" : "bg-blue-50"
              }`}
              onClick={() => markAsRead(notification.id)}
            >
              <p className="font-medium">{notification.title}</p>
              <p className="text-sm text-gray-600">{notification.message}</p>
              <p className="text-xs text-gray-400 mt-1">
                {formatDistanceToNow(new Date(notification.createdAt))} ago
              </p>
            </div>
          ))}
        </div>
      </PopoverContent>
    </Popover>
  )
}
```

7.3 Email Notifications

7.3.1 Email Service Configuration

Nodemailer Setup:

```
// lib/email.ts
import nodemailer from "nodemailer"

const transporter = nodemailer.createTransporter({
  host: process.env.SMTP_HOST,
  port: parseInt(process.env.SMTP_PORT || "587"),
  secure: false,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASSWORD
  }
})

export async function sendEmail({
  to,
  subject,
  html
}: {
  to: string
  subject: string
  html: string
}) {
  await transporter.sendMail({
    from: process.env.SMTP_USER,
    to,
    subject,
    html
  })
}
```

7.3.2 HTML Email Templates

RFI Response Template:

```
// lib/email-templates/rfi-response.ts
export function rfiResponseTemplate({
  userName,
  rfiSubject,
  response,
  projectName,
  rfiUrl
}: {
  userName: string
  rfiSubject: string
  response: string
  projectName: string
  rfiUrl: string
}) {
  return `
    <!DOCTYPE html>
    <html>
      <head>
        <style>
          body { font-family: Arial, sans-serif; line-height: 1.6; }
          .container { max-width: 600px; margin: 0 auto; padding: 20px; }
          .header { background: #1e40af; color: white; padding: 20px; text-align: center; }
          .content { background: #f9fafb; padding: 20px; margin: 20px 0; }
          .button { background: #1e40af; color: white; padding: 12px 24px; text-decoration: none; borde
          .footer { text-align: center; color: #6b7280; font-size: 12px; margin-top: 20px; }
        </style>
      </head>
      <body>
        <div class="container">
          <div class="header">
            <h1>BuildOS</h1>
          </div>
          <div class="content">
            <h2>RFI Response Received</h2>
            <p>Hi ${userName},</p>
            <p>Your RFI has received a response:</p>
            <p><strong>Project:</strong> ${projectName}</p>
            <p><strong>RFI:</strong> ${rfiSubject}</p>
            <p><strong>Response:</strong></p>
            <p>${response}</p>
            <p>
              <a href="${rfiUrl}" class="button">View RFI</a>
            </p>
          </div>
          <div class="footer">
            <p>© 2026 BuildOS. All rights reserved.</p>
            <p>You're receiving this email because you have notifications enabled.</p>
          </div>
        </div>
      </body>
    </html>
  `
}
```

7.3.3 Email Notification Trigger

Integration with Notification Creation:

```
// lib/notifications.ts
export async function createNotificationWithEmail({
  userId,
  type,
  title,
  message,
  category,
  relatedId,
  relatedType
}: NotificationParams) {
  // Create in-app notification
  const notification = await createNotification({
    userId,
    type,
    title,
    message,
    category,
    relatedId,
    relatedType
  })

  // Check user preferences
  const preferences = await prisma.notificationPreference.findUnique({
    where: { userId }
  })

  if (preferences?.emailEnabled) {
    // Get user email
    const user = await prisma.user.findUnique({
      where: { id: userId },
      select: { email: true, name: true }
    })

    // Send email based on notification type
    const emailTemplate = getEmailTemplate(type, {
      userName: user.name,
      title,
      message,
      relatedId,
      relatedType
    })

    await sendEmail({
      to: user.email,
      subject: title,
      html: emailTemplate
    })
  }

  return notification
}
```

7.3.4 User Notification Preferences

Preference Management:

```
// app/api/notifications/preferences/route.ts
export async function PATCH(request: Request) {
  const session = await requireAuth()
  const body = await request.json()

  const preferences = await prisma.notificationPreference.upsert({
    where: { userId: session.user.id },
    update: body,
    create: {
      userId: session.user.id,
      ...body
    }
  })

  return NextResponse.json({
    success: true,
    preferences
  })
}
```

Preference UI:

```
// components/NotificationPreferences.tsx
export function NotificationPreferences() {
  const [preferences, setPreferences] = useState({
    emailEnabled: true,
    inAppEnabled: true,
    rfiNotifications: true,
    submittalNotifications: true,
    changeOrderNotifications: true,
    punchItemNotifications: true,
    dailyReportNotifications: false
  })

  async function savePreferences() {
    await fetch("/api/notifications/preferences", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(preferences)
    })
  }

  return (
    <div className="space-y-4">
      <h2>Notification Preferences</h2>

      <div>
        <label>
          <input
            type="checkbox"
            checked={preferences.emailEnabled}
            onChange={(e) => setPreferences({
              ...preferences,
              emailEnabled: e.target.checked
            })}
          />
          Email Notifications
        </label>
      </div>

      <div>
        <label>
          <input
            type="checkbox"
            checked={preferences.rfiNotifications}
            onChange={(e) => setPreferences({
              ...preferences,
              rfiNotifications: e.target.checked
            })}
          />
          RFI Notifications
        </label>
      </div>

      {/* More preference options... */}

      <button onClick={savePreferences}>
        Save Preferences
      </button>
    </div>
  )
}
```

8 Analytics System

8.1 Architecture Overview

The analytics system provides real-time insights into project performance, team productivity, and financial health through optimized database queries and client-side aggregation.

8.2 Optimized Prisma Queries

8.2.1 Project Health Metrics

Query Strategy: - Fetch all necessary data in a single query - Use Prisma's `select` to limit fields - Leverage database indexes for performance - Aggregate on the client side for flexibility

Implementation:

```
// lib/analytics/project-health.ts
export async function getProjectHealthMetrics(dateRange: number = 30) {
  const startDate = new Date()
  startDate.setDate(startDate.getDate() - dateRange)

  // Fetch all projects with related data
  const projects = await prisma.project.findMany({
    where: {
      status: "ACTIVE"
    },
    select: {
      id: true,
      name: true,
      budget: true,
      startDate: true,
      endDate: true,
      status: true,
      changeOrders: {
        where: { status: "APPROVED" },
        select: { approvedCost: true }
      },
      rfis: {
        where: { createdAt: { gte: startDate } },
        select: { id: true, status: true }
      },
      punchItems: {
        where: { status: { in: ["OPEN", "IN_PROGRESS"] } },
        select: { id: true, priority: true }
      }
    }
  })

  // Client-side aggregation
  const metrics = {
    activeProjects: projects.length,
    onTimePercentage: calculateOnTimePercentage(projects),
    budgetVariance: calculateBudgetVariance(projects),
    criticalItems: countCriticalItems(projects)
  }
  return metrics
}

function calculateOnTimePercentage(projects: any[]): number {
  const projectsWithDates = projects.filter(p => p.endDate)
  const onTimeProjects = projectsWithDates.filter(p => {
    return new Date(p.endDate) >= new Date()
  })

  return projectsWithDates.length > 0
    ? (onTimeProjects.length / projectsWithDates.length) * 100
    : 0
}

function calculateBudgetVariance(projects: any[]): number {
  let totalBudget = 0
  let totalSpent = 0

  projects.forEach(project => {
    if (project.budget) {
      totalBudget += Number(project.budget)

      const changeOrderCosts = project.changeOrders.reduce(
        (sum: number, co: any) => sum + Number(co.approvedCost || 0),
        0
      )

      totalSpent += changeOrderCosts
    }
  })

  return totalBudget > 0
    ? ((totalSpent - totalBudget) / totalBudget) * 100
    : 0
}
```

8.2.2 RFI Velocity Metrics

Query with Date Filtering:

```
// lib/analytics/rfi-metrics.ts
export async function getRFIMetrics(
  projectId?: string,
  dateRange: number = 30
) {
  const startDate = new Date()
  startDate.setDate(startDate.getDate() - dateRange)

  const rfis = await prisma.rFI.findMany({
    where: {
      ...(projectId && { projectId }),
      createdAt: { gte: startDate }
    },
    select: {
      id: true,
      status: true,
      priority: true,
      createdAt: true,
      dueDate: true,
      responses: {
        select: {
          createdAt: true
        },
        orderBy: {
          createdAt: "asc"
        },
        take: 1
      }
    }
  })

  // Calculate metrics
  const totalRFIs = rfis.length
  const openRFIs = rfis.filter(r => r.status !== "CLOSED").length
  const overdueRFIs = rfis.filter(r =>
    r.dueDate && new Date(r.dueDate) < new Date() && r.status !== "CLOSED"
  ).length

  // Calculate average response time
  const responseTimes = rfis
    .filter(r => r.responses.length > 0)
    .map(r => {
      const created = new Date(r.createdAt)
      const responded = new Date(r.responses[0].createdAt)
      return (responded.getTime() - created.getTime()) / (1000 * 60 * 60 * 24) // Days
    })

  const averageResponseTime = responseTimes.length > 0
    ? responseTimes.reduce((a, b) => a + b, 0) / responseTimes.length
    : 0

  return {
    totalRFIs,
    openRFIs,
    overdueRFIs,
    averageResponseTime: Math.round(averageResponseTime * 10) / 10
  }
}
```

8.3 Client-Side Aggregation

8.3.1 Trend Data Processing

Time-Series Aggregation:

```
// lib/analytics/trends.ts
export function aggregateTrendData(
  data: any[],
  dateField: string,
  groupBy: "day" | "week" | "month"
): TrendData[] {
  // Group data by time period
  const grouped = data.reduce((acc, item) => {
    const date = new Date(item[dateField])
    const key = formatDateKey(date, groupBy)

    if (!acc[key]) {
      acc[key] = {
        date: key,
        created: 0,
        closed: 0,
        open: 0
      }
    }

    acc[key].created++
    if (item.status === "CLOSED") {
      acc[key].closed++
    }

    return acc
  }, {} as Record<string, any>)

  // Convert to array and calculate running totals
  const trends = Object.values(grouped).sort((a, b) =>
    new Date(a.date).getTime() - new Date(b.date).getTime()
  )

  let runningOpen = 0
  trends.forEach(trend => {
    runningOpen += trend.created - trend.closed
    trend.open = runningOpen
  })

  return trends
}

function formatDateKey(date: Date, groupBy: string): string {
  switch (groupBy) {
    case "day":
      return date.toISOString().split("T")[0]
    case "week":
      const weekStart = new Date(date)
      weekStart.setDate(date.getDate() - date.getDay())
      return weekStart.toISOString().split("T")[0]
    case "month":
      return `${date.getFullYear()}-${String(date.getMonth() + 1).padStart(2, "0")}`
    default:
      return date.toISOString().split("T")[0]
  }
}
```

8.3.2 Chart Data Preparation

Chart.js Data Formatting:

```
// lib/analytics/chart-data.ts
export function prepareLineChartData(
  trends: TrendData[],
  labels: string[]
): ChartData {
  return {
    labels: trends.map(t => formatDate(t.date)),
    datasets: [
      {
        label: "Created",
        data: trends.map(t => t.created),
        borderColor: "rgb(59, 130, 246)",
        backgroundColor: "rgba(59, 130, 246, 0.1)",
        tension: 0.4
      },
      {
        label: "Closed",
        data: trends.map(t => t.closed),
        borderColor: "rgb(34, 197, 94)",
        backgroundColor: "rgba(34, 197, 94, 0.1)",
        tension: 0.4
      },
      {
        label: "Open",
        data: trends.map(t => t.open),
        borderColor: "rgb(234, 179, 8)",
        backgroundColor: "rgba(234, 179, 8, 0.1)",
        tension: 0.4
      }
    ]
  }
}

export function prepareDoughnutChartData(
  data: Record<string, number>
): ChartData {
  return {
    labels: Object.keys(data),
    datasets: [
      {
        data: Object.values(data),
        backgroundColor: [
          "rgba(59, 130, 246, 0.8)",
          "rgba(34, 197, 94, 0.8)",
          "rgba(234, 179, 8, 0.8)",
          "rgba(239, 68, 68, 0.8)",
          "rgba(168, 85, 247, 0.8)"
        ],
        borderWidth: 2,
        borderColor: "#ffffff"
      }
    ]
  }
}
```

8.4 Date Range Filtering

8.4.1 Filter Component

Date Range Selector:

```
// components/DateRangeFilter.tsx
export function DateRangeFilter({
  value,
  onChange
}: {
  value: number
  onChange: (days: number) => void
}) {
  const options = [
    { label: "Last 7 days", value: 7 },
    { label: "Last 30 days", value: 30 },
    { label: "Last 90 days", value: 90 },
    { label: "Last 365 days", value: 365 }
  ]

  return (
    <Select value={value.toString()} onValueChange={(v) => onChange(parseInt(v))}>
      <SelectTrigger className="w-[180px]">
        <SelectValue />
      </SelectTrigger>
      <SelectContent>
        {options.map(option => (
          <SelectItem key={option.value} value={option.value.toString()}>
            {option.label}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
  )
}
```

8.4.2 Dynamic Query Updates

React Hook for Analytics:

```
// hooks/useAnalytics.ts
export function useAnalytics(projectId?: string) {
  const [dateRange, setDateRange] = useState(30)
  const [metrics, setMetrics] = useState(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    fetchMetrics()
  }, [dateRange, projectId])

  async function fetchMetrics() {
    setLoading(true)

    const params = new URLSearchParams({
      dateRange: dateRange.toString(),
      ...(projectId && { projectId })
    })

    const response = await fetch(`/api/analytics/metrics?${params}`)
    const data = await response.json()

    setMetrics(data.metrics)
    setLoading(false)
  }

  return {
    metrics,
    loading,
    dateRange,
    setDateRange,
    refresh: fetchMetrics
  }
}
```

8.5 Role-Based Data Access

8.5.1 Data Filtering by Role

Access Control in Queries:

```
// lib/analytics/access-control.ts
export function getProjectFilter(session: Session): Prisma.ProjectWhereInput {
  switch (session.user.role) {
    case "ADMIN":
      // Admins see all projects
      return {}

    case "PROJECT_MANAGER":
      // Project managers see their assigned projects
      return {
        OR: [
          { userId: session.user.id },
          { projectManager: session.user.name }
        ]
      }

    case "FIELD_SUPERVISOR":
    case "SUBCONTRACTOR":
      // Limited access to assigned projects only
      return {
        userId: session.user.id
      }

    case "CLIENT":
      // Clients see projects where they are the client
      return {
        clientName: session.user.name
      }

    default:
      return { id: "none" } // No access
  }
}
```

Usage in Analytics Queries:

```
// app/api/analytics/metrics/route.ts
export async function GET(request: Request) {
  const session = await requireAuth()
  const { searchParams } = new URL(request.url)

  const dateRange = parseInt(searchParams.get("dateRange") || "30")
  const projectId = searchParams.get("projectId")

  // Apply role-based filtering
  const projectFilter = getProjectFilter(session)

  const projects = await prisma.project.findMany({
    where: {
      ...projectFilter,
      ...(projectId && { id: projectId }),
      status: "ACTIVE"
    },
    // ... rest of query
  })

  // ... calculate and return metrics
}
```

9 Performance Optimizations

9.1 Singleton Prisma Client

9.1.1 Problem

Creating multiple Prisma Client instances can exhaust database connections and degrade performance.

9.1.2 Solution

Singleton Pattern Implementation:

```
// lib/prisma.ts
import { PrismaClient } from "@prisma/client"

const globalForPrisma = global as unknown as { prisma: PrismaClient }

export const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    log: process.env.NODE_ENV === "development" ? ["query", "error", "warn"] : ["error"]
  })

if (process.env.NODE_ENV !== "production") globalForPrisma.prisma = prisma
```

Benefits: - Single database connection pool - Prevents connection exhaustion - Improved performance in development (hot reload) - Consistent logging configuration

9.2 Connection Pooling

9.2.1 Configuration

Database URL with Connection Pooling:

```
# .env
DATABASE_URL="postgresql://user:password@localhost:5432/buildos?connection_limit=10&pool_timeout=20"
```

Prisma Configuration:

```
// prisma/schema.prisma
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider        = "prisma-client-js"
  previewFeatures = ["fullTextSearch", "fullTextIndex"]
}
```

Connection Pool Settings: - **connection_limit:** Maximum number of connections (default: 10) - **pool_timeout:** Timeout for acquiring connection (seconds) - **connect_timeout:** Timeout for establishing connection

9.2.2 Benefits

- Reuse database connections
- Reduce connection overhead

- Handle concurrent requests efficiently
- Prevent connection exhaustion

9.3 Bulk Data Fetching

9.3.1 Strategy

Instead of multiple individual queries, fetch related data in bulk using Prisma's `include` and `select`.

Inefficient Approach (N+1 Problem):

```
// DON'T DO THIS
const projects = await prisma.project.findMany()

for (const project of projects) {
  const rfis = await prisma.rFI.findMany({
    where: { projectId: project.id }
  })
  // Process rfis...
}
```

Optimized Approach:

```
// DO THIS
const projects = await prisma.project.findMany({
  include: {
    rfis: {
      where: { status: { not: "CLOSED" } },
      select: {
        id: true,
        subject: true,
        status: true,
        priority: true
      }
    },
    submittals: {
      where: { status: "PENDING" },
      select: {
        id: true,
        title: true,
        status: true
      }
    }
  }
})

// All data fetched in a single query
```

9.3.2 Client-Side Aggregation

Fetch Once, Aggregate Multiple Times:

```
// Fetch all data once
const data = await prisma.rFI.findMany({
  where: {
    createdAt: { gte: startDate }
  },
  select: {
    id: true,
    status: true,
    priority: true,
    createdAt: true,
    projectId: true
  }
})

// Perform multiple aggregations on client
const totalRFIs = data.length
const openRFIs = data.filter(r => r.status !== "CLOSED").length
const highPriorityRFIs = data.filter(r => r.priority === "HIGH").length
const rfisByProject = groupBy(data, "projectId")
const rfisByStatus = groupBy(data, "status")
```

Benefits: - Single database query - Reduced network latency - Flexible aggregation logic - Easier to test and debug

9.4 Database Indexes

9.4.1 Index Strategy

Prisma Schema with Indexes:

```
model Notification {
  id          String    @id @default(cuid())
  userId      String
  type        NotificationType
  isRead      Boolean   @default(false)
  createdAt   DateTime  @default(now())

  user        User      @relation(fields: [userId], references: [id])

  @@index([userId, isRead])  // Composite index for common query
  @@index([createdAt])       // Index for sorting
}

model Activity {
  id          String    @id @default(cuid())
  userId      String
  projectId   String?
  activityType ActivityType
  createdAt   DateTime  @default(now())

  user        User      @relation(fields: [userId], references: [id])
  project     Project?  @relation(fields: [projectId], references: [id])

  @@index([userId])      // Index for user queries
  @@index([projectId])   // Index for project queries
  @@index([createdAt])   // Index for time-based queries
}

model TimeEntry {
  id          String    @id @default(cuid())
  projectId   String
  userId      String
  date        DateTime
  status      TimeEntryStatus

  project     Project   @relation(fields: [projectId], references: [id])
  user        User      @relation(fields: [userId], references: [id])

  @@index([projectId])   // Index for project queries
  @@index([userId])      // Index for user queries
  @@index([date])        // Index for date range queries
}
```

Index Guidelines: - Index foreign keys used in joins - Index fields used in WHERE clauses - Index fields used for sorting (ORDER BY) - Use composite indexes for common query patterns - Avoid over-indexing (impacts write performance)

9.5 Server-Side Rendering

9.5.1 Next.js App Router

Server Components (Default):

```
// app/projects/page.tsx
import { prisma } from "@/lib/prisma"
import { getServerSession } from "next-auth"

// This is a Server Component by default
export default async function ProjectsPage() {
  const session = await getServerSession()

  // Data fetching happens on the server
  const projects = await prisma.project.findMany({
    where: { userId: session.user.id },
    orderBy: { createdAt: "desc" }
  })

  return (
    <div>
      <h1>Projects</h1>
      <ProjectList projects={projects} />
    </div>
  )
}
```

Benefits: - Faster initial page load - Better SEO - Reduced client-side JavaScript - Direct database access (no API route needed) - Automatic code splitting

9.5.2 Client Components (When Needed)

Use Client Components for: - Interactive features (forms, buttons) - Browser APIs (localStorage, geolocation) - State management (useState, useContext) - Event handlers (onClick, onChange)

```
// components/ProjectForm.tsx
"use client"

import { useState } from "react"
import { useRouter } from "next/navigation"

export function ProjectForm() {
  const [formData, setFormData] = useState({})
  const router = useRouter()

  async function handleSubmit(e: FormEvent) {
    e.preventDefault()

    await fetch("/api/projects", {
      method: "POST",
      body: JSON.stringify(formData)
    })

    router.push("/projects")
    router.refresh() // Revalidate server components
  }

  return <form onSubmit={handleSubmit}>...</form>
}
```

9.6 Caching Strategies

9.6.1 Next.js Caching

Route Segment Config:

```
// app/projects/page.tsx
export const revalidate = 60 // Revalidate every 60 seconds

export default async function ProjectsPage() {
  // This page will be cached and revalidated every 60 seconds
  const projects = await getProjects()
  return <ProjectList projects={projects} />
}
```

On-Demand Revalidation:

```
// app/api/projects/route.ts
import { revalidatePath } from "next/cache"

export async function POST(request: Request) {
  const body = await request.json()

  // Create project
  await prisma.project.create({ data: body })

  // Revalidate the projects page
  revalidatePath("/projects")

  return NextResponse.json({ success: true })
}
```

9.6.2 React Query (Optional Enhancement)

Client-Side Caching:

```
// hooks/useProjects.ts
import { useQuery } from "@tanstack/react-query"

export function useProjects() {
  return useQuery({
    queryKey: ["projects"],
    queryFn: async () => {
      const response = await fetch("/api/projects")
      return response.json()
    },
    staleTime: 5 * 60 * 1000, // 5 minutes
    cacheTime: 10 * 60 * 1000 // 10 minutes
  })
}
```

Benefits: - Automatic background refetching - Optimistic updates - Request deduplication - Pagination and infinite scroll support

9.7 Image Optimization

9.7.1 Next.js Image Component

Automatic Optimization:

```
import Image from "next/image"

export function ProjectImage({ src, alt }: { src: string; alt: string }) {
  return (
    <Image
      src={src}
      alt={alt}
      width={800}
      height={600}
      quality={85}
      placeholder="blur"
      blurDataURL="/placeholder.jpg"
    />
  )
}
```

Benefits: - Automatic format conversion (WebP, AVIF) - Responsive images - Lazy loading - Blur placeholder - Prevents layout shift

9.7.2 S3 Image Optimization

CloudFront CDN (Recommended): - Cache images at edge locations - Automatic compression - Custom cache policies - HTTPS by default

---

Document Version: 1.0
Last Updated: January 2026
Status: Production Release