

Contents

| | |
|--|-----------|
| Welcome to Sync Task Management App | 4 |
| 1 Capitolul 1: Introducere | 5 |
| 1.1 SYNC - REZUMAT TEHNIC | 5 |
| 1.1.1 Ce este această aplicație? | 5 |
| 1.1.2 Ce funcționalități oferă? | 5 |
| 1.2 Prezentare generală a arhitecturii sistemului | 6 |
| 1.3 Ce face „SYNC” special | 6 |
| 1.4 Core Features Overview | 7 |
| 1.4.1 Kanban Board Management | 7 |
| 1.4.2 TailwindCSS Design System | 7 |
| 1.4.3 CRUD Operations | 8 |
| 1.4.4 Database Transaction Architecture | 8 |
| 1.4.5 Role-Based Access Control (RBAC) | 8 |
| 1.4.6 File Management | 8 |
| 1.4.7 8. Security Layer Architecture | 9 |
| 1.5 Application Architecture | 10 |
| 1.5.1 Frontend Technologies | 10 |
| 1.5.2 Backend Technologies | 10 |
| 1.5.3 Security Technologies | 10 |
| 1.5.4 9 Task Status Flow & Pending Status Management | 11 |
| 1.5.5 10. Database Query Architecture & Optimization | 11 |
| 1.5.6 11. Algorithm & Sorting Implementation | 12 |
| 1.5.7 12. Dynamic Button & Form Components | 13 |
| 1.6 Chapter Summary: Technical Coverage | 13 |
| 1.6.1 What We’ve Covered | 13 |
| 1.6.2 Key Technical Achievements | 13 |
| 1.6.3 What’s Coming Next | 13 |
| 2 Project Setup & Folder Structure | 14 |
| 2.1 Development Example Setup | 14 |
| 2.1.1 Example Folder Structure | 14 |
| 2.2 Environment Configuration | 17 |
| 2.2.1 Client package.json | 17 |
| 2.2.2 Server package.json | 17 |
| 2.3 Development Setup Instructions | 18 |
| 3 Backend Development | 20 |
| 3.1 Express Server Architecture | 20 |
| 3.2 Express Server | 20 |
| 3.3 Middleware Stack | 20 |
| 3.4 Route Handlers | 20 |

| | | |
|----------|--|-----------|
| 3.5 | Service Layer | 20 |
| 3.6 | Prisma ORM | 21 |
| 3.7 | PostgreSQL Database | 21 |
| 3.7.1 | Database Architecture & Query Flow | 21 |
| 3.7.2 | API Endpoint Architecture Foundation | 21 |
| 3.7.3 | Server Implementation | 22 |
| 3.7.4 | Authentication Middleware | 23 |
| 3.8 | Users Table | 24 |
| 3.9 | Tasks Table | 24 |
| 3.10 | Attachments Table | 24 |
| 3.11 | Projects Table | 25 |
| 3.12 | Database Indexing Strategy | 25 |
| 3.12.1 | Prisma Client Configuration | 26 |
| 3.12.2 | Task Controller Implementation | 26 |
| 3.12.3 | Route Implementation | 27 |
| 4 | Authentication & Security | 29 |
| 4.0.1 | Multi-Layer Security Implementation | 29 |
| 4.0.2 | Authentication Service Implementation | 30 |
| 4.0.3 | CSRF Protection Middleware | 32 |
| 5 | Frontend Architecture | 34 |
| 5.0.1 | Main Application Component | 34 |
| 5.1 | Design System & UI Architecture | 35 |
| 5.1.1 | Component Architecture | 35 |
| 5.1.2 | State Management with Zustand | 35 |
| 5.1.3 | Form Components | 37 |
| 5.1.4 | Custom Hooks | 39 |
| 5.1.5 | Kanban Board Components | 41 |
| 5.2 | Performance Optimization | 42 |
| 5.2.1 | Code Splitting and Lazy Loading | 42 |
| 5.2.2 | Responsive Design Implementation | 42 |
| 6 | Task Management & Kanban | 43 |
| 6.0.1 | Kanban Board Architecture Overview | 43 |
| 6.0.2 | Task Status Flow & Workflow Management | 43 |
| 6.0.3 | Task Assignment & User Management | 44 |
| 6.0.4 | Priority Management & Sorting Algorithms | 44 |
| 6.0.5 | Real-time Collaboration & Updates | 45 |
| 7 | File Handling & Storage | 48 |
| 7.1 | Overview | 48 |
| 7.2 | File Management Architecture | 48 |
| 7.2.1 | System Overview | 48 |
| 7.3 | Secure File Uploads | 49 |
| 7.3.1 | File Type Validation | 49 |
| 7.3.2 | File Size Validation | 50 |
| 7.4 | • Advanced Security & Validation Diagrams | 51 |
| 7.4.1 | Multi-Layer Security Architecture | 51 |
| 7.4.2 | File Validation & Sanitization Flow | 51 |
| 7.4.3 | Google Cloud Storage Integration Architecture | 52 |
| 7.4.4 | File Processing & Optimization Pipeline | 53 |
| 7.4.5 | File Access Control & Permission System | 53 |
| 7.4.6 | File Search & Discovery System | 54 |

| | | |
|----------|--|-----------|
| 7.4.7 | File Analytics & Reporting System | 55 |
| 7.4.8 | File Backup & Disaster Recovery | 56 |
| 7.4.9 | File Performance & Optimization | 57 |
| 7.4.10 | File Integration & API System | 57 |
| 7.4.11 | File Compliance & Governance | 58 |
| 7.4.12 | File Workflow & Automation | 59 |
| 7.4.13 | File Monitoring & Alerting | 60 |
| 7.4.14 | File Scalability & Future Architecture | 61 |
| 7.4.15 | File Integration & Ecosystem | 62 |
| 7.4.16 | File User Experience & Interface | 63 |
| 7.4.17 | Content Security | 63 |
| 8 | Deployment & Scaling | 65 |
| 8.1 | Overview | 65 |
| 8.2 | Containerization with Docker | 65 |
| 8.2.1 | Development Dockerfile | 65 |
| 8.2.2 | Docker Compose for Production | 65 |
| 8.3 | Scaling Strategies | 67 |
| 8.3.1 | Vertical Scaling | 67 |
| 8.3.2 | Horizontal Scaling | 68 |
| 9 | Bibliography | 69 |
| 9.1 | Primary Technologies & Frameworks | 69 |
| 9.1.1 | Backend Technologies | 69 |
| 9.1.2 | Frontend Technologies | 69 |
| 9.1.3 | State Management | 69 |
| 9.1.4 | Form Management | 69 |
| 9.1.5 | Authentication & Security | 70 |
| 9.1.6 | File Handling & Storage | 70 |
| 9.1.7 | Development Tools | 70 |
| 9.1.8 | Deployment & DevOps | 70 |
| 9.1.9 | Security Standards & Best Practices | 70 |
| 9.1.10 | Database Design & Migration | 70 |

Welcome to Sync Task Management App

Full-Stack: PostgreSQL + Prisma + Express + Node.js + Zustand + TailwindCSS + JWT + Google API

Version: 1.0

Document Type: Technical Guide

GitHub: github.com/naeioana

Website: testApp.Sync.ro

Date: 2025 **Author:** Nae Ioana { .unnumbered .frontmatter }

Chapter 1

Capitolul 1: Introducere

1.1 SYNC - REZUMAT TEHNIC

1.1.1 Ce este această aplicație?

Sync reprezintă o abordare nouă în modul în care organizațiile gestionează proiectele și colaborează în echipă. Nu este doar o aplicație obișnuită de urmărire a task-urilor; este o soluție completă, construită la standarde enterprise, care integrează bune practici de arhitectură software modernă și răspunde provocărilor reale ale managementului de proiect în mediul rapid de astăzi.

Aplicația demonstrează cum poate fi creat un sistem funcțional, sigur și scalabil, care să reziste cerințelor reale ale utilizatorilor. Este un punct de referință pentru dezvoltatori, lideri tehnici și organizații care vor să înțeleagă cum se construiesc aplicații gata de producție, cu arhitectură solidă și performanță constantă.

Ceea ce diferențiază Sync de alte unelte de management este abordarea sa holistică. În loc să se concentreze doar pe task-uri, aplicația gestionează întregul ecosistem de colaborare, inclusiv securitatea datelor, cerințele de scalabilitate și integrarea fără probleme cu procesele existente ale organizației. În același timp, servește ca resursă educațională, exemplificând bune practici în dezvoltarea full-stack, implementarea securității și design-ul sistemelor.

1.1.2 Ce funcționalități oferă?

1.1.2.1 Funcționalitate de bază

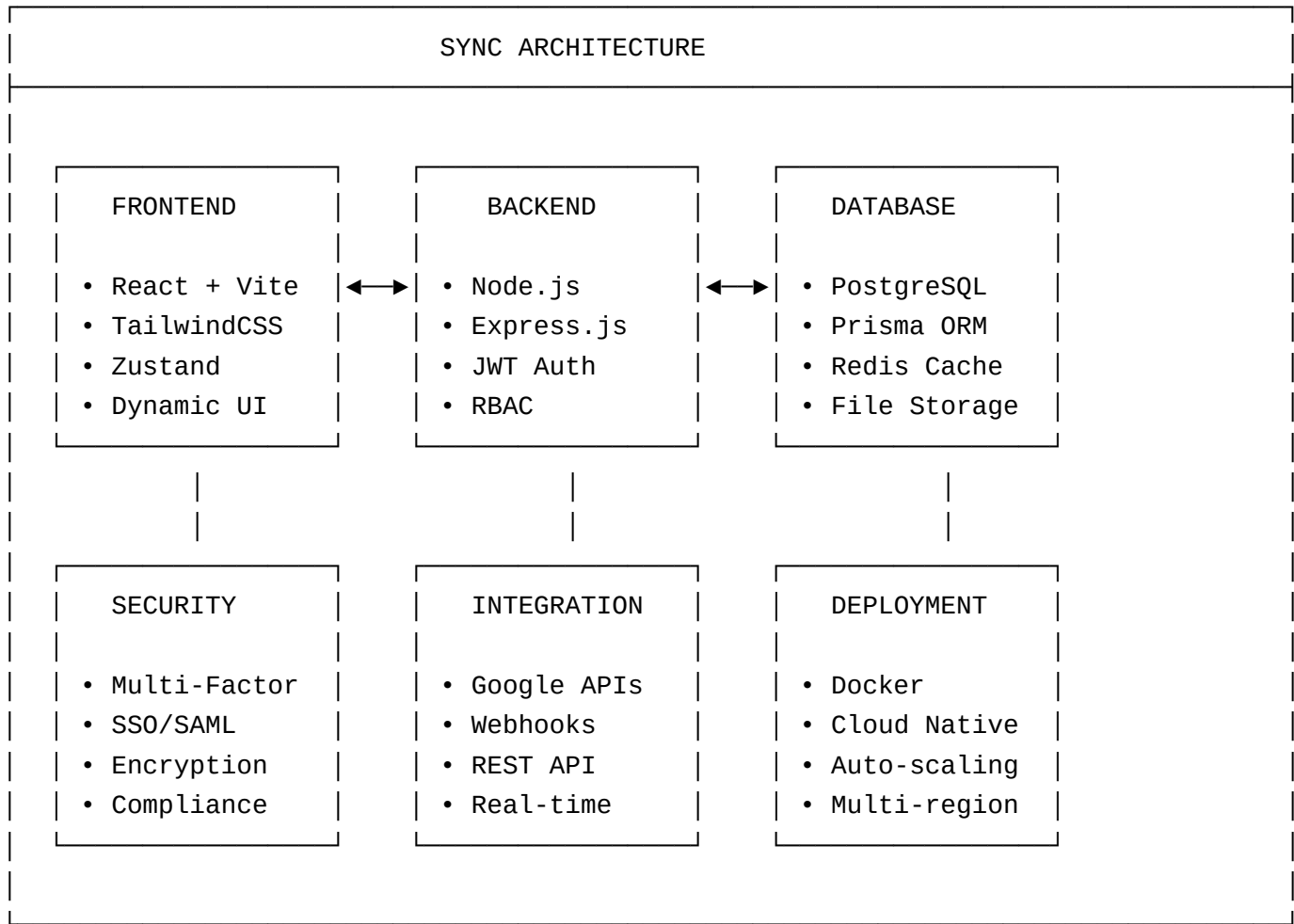
Scopul principal al aplicației este să transforme modul în care echipele planifică, organizează și realizează munca. La baza acestei transformări se află panoul Kanban, care oferă o reprezentare vizuală a fluxului de lucru, mult mai clară decât simplele liste de task-uri. Interfața drag-and-drop permite mutarea sarcinilor între stadii – de la idee, la în desfășurare și finalizare – fără pierderi de timp sau confuzie. Această abordare vizuală reduce povara administrativă și permite echipei să se concentreze pe execuție.

Gestionarea task-urilor depășește operațiile CRUD de bază. Fiecare task devine un container complet de informații: descrieri detaliate, priorități ajustabile, termene realiste cu notificări automate și atribuirea clară a responsabilităților. Sistemul urmărește nu doar starea curentă a sarcinilor, ci și istoricul complet al modificărilor: cine a făcut schimbări, când și ce modificări au fost aplicate. Această trasabilitate oferă transparența necesară pentru o guvernare eficientă a proiectelor.

Un alt element esențial este controlul bazat pe roluri. Aplicația implementează o structură de permisiuni sofisticată, care reflectă ierarhiile reale din organizații, păstrând în același timp flexibilitatea necesară colaborării eficiente. Administratorii au acces complet la configurarea aplicației, gestionarea utilizatorilor și analiza datelor,

În timp ce utilizatorii obișnuiți pot vedea și gestiona doar task-urile și informațiile relevante pentru ei. Acest nivel granular de permisiuni asigură protecția datelor sensibile și permite, în același timp, colaborarea fluentă necesară succesului proiectelor.

1.2 Prezentare generală a arhitecturii sistemului



1.3 Ce face „SYNC” special

În mediul rapid de dezvoltare de astăzi, echipele au nevoie de mai mult decât urmărirea simplă a task-urilor. Ele necesită:

- **Arhitectură dinamică a componentelor:** UI bazat pe React, cu TailwindCSS pentru interfețe responsive și plăcute vizual
- **Arhitectură scalabilă:** Fundament care crește odată cu nevoile afacerii
- **Arhitectură cloud-native:** Integrare Google Cloud Storage cu CDN global și edge computing
- **Securitate pe mai multe niveluri:** Securitate enterprise cu JWT, RBAC și funcționalități de conformitate
- **Design avansat al bazei de date:** PostgreSQL cu Prisma ORM pentru operațiuni sigure și performante
- **Colaborare în timp real:** Actualizări live și colaborare de echipă bazate pe WebSocket
- **Funcționalități bazate pe AI:** Machine learning pentru analiză de conținut și automatizare inteligentă
- **Analiză completă:** Informații în timp real și monitorizare a performanței
- **Pregătită pentru DevOps:** Integrare Docker, Kubernetes și pipeline CI/CD
- **Management centralizat al proiectelor:** Sursă unică de adevăr pentru toate activitățile

- **Control bazat pe roluri:** Colaborare sigură cu diferite niveluri de permisiuni
- **Urmărirea progresului în timp real:** Vizualizarea stadiului proiectului prin panouri Kanban
- **Gestionarea integrată a fișierelor:** Manipulare ușoară a documentelor asociate task-urilor
- **Securitate enterprise:** Securitate pe mai multe niveluri, cu capabilități de audit și conformitate
- **Analize și automatizare AI:** Analitică inteligentă și automatizare
- **Design Mobile-First:** Interfețe responsive pentru toate dispozitivele
- **Integrare AI:** Machine learning pentru analiză de conținut

1.4 Core Features Overview

1.4.1 Kanban Board Management

The heart of the application is a sophisticated Kanban board system that provides:

Advanced Kanban Features:

- **Drag & Drop:** Intuitive task movement between columns
- **Priority Colors:** Visual priority indicators (Red=High, Yellow=Medium, Green=Low)
- **Progress Tracking:** Real-time completion percentages
- **Filtering:** Sort by assignee, priority, due date, or tags
- **Search:** Quick task discovery across all boards
- **Real-Time Updates:** Live collaboration with team members
- **Mobile Responsive:** Optimized for all device sizes
- **Custom Workflows:** Configurable board layouts and statuses
- **Navigation:** Responsive navigation with user menu and auth
- **Dashboard:** Real-time interactive charts and metrics
- **Kanban Board:** Drag & drop with live updates and filters
- **Task Forms:** Validation, auto-save, and rich text editor
- **File Upload:** Drag & drop with progress bar and preview
- **User Profile:** Settings, preferences, and activity tracking

| TODO | IN PROGRESS | REVIEW | DONE |
|--|---|---|---|
| <ul style="list-style-type: none">• Task 1• Task 2• Task 3 | <ul style="list-style-type: none">• Task 4• Task 5 | <ul style="list-style-type: none">• Task 6• Task 7 | <ul style="list-style-type: none">• Task 8• Task 9• Task 10 |

1.4.2 TailwindCSS Design System

TailwindCSS system for consistent, responsive, and beautiful interfaces:

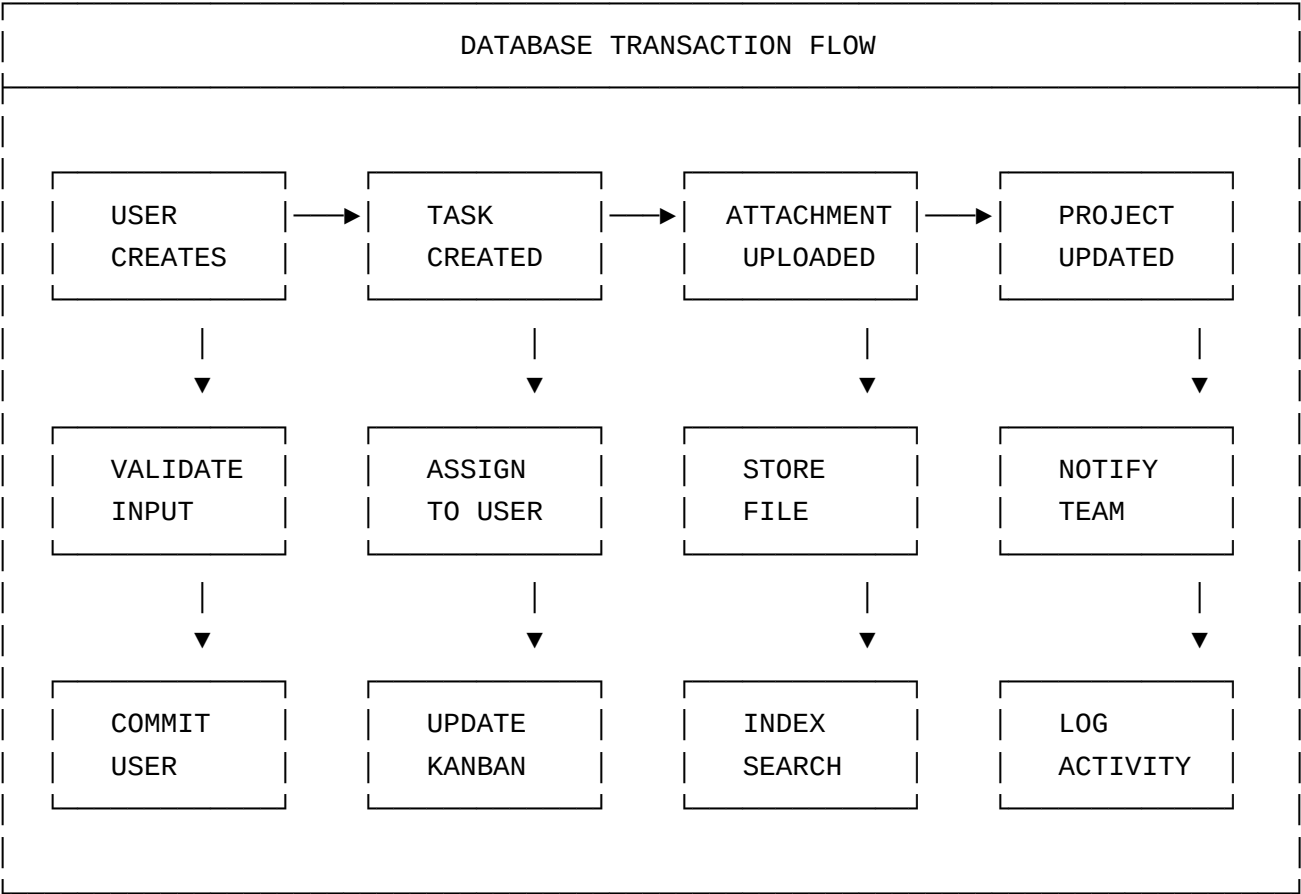
- **Color Scheme:** Primary, secondary, accent, success, warning, error themes
- **Typography:** Headings, body text, monospace, responsive design
- **Components:** Buttons, cards, forms, modals, and alerts
- **Responsive:** Mobile-first with adaptive breakpoints and grid system
- **Animations:** Smooth transitions, hover effects, and loading states
- **Utilities:** Comprehensive spacing, flexbox, and grid utilities

1.4.3 CRUD Operations

Full Create, Read, Update, Delete functionality for:

- **Tasks:** Title, description, status, priority, due dates
- **Projects:** Organization and categorization
- **Users:** Profile management and role assignment
- **Attachments:** File uploads and management

1.4.4 Database Transaction Architecture



1.4.5 Role-Based Access Control (RBAC)

Permisiuni:

| Acțiune | Super Admin | Company Admin | Project Manager | Team Member |
|------------------|-------------|---------------|-----------------|-------------|
| User Management | x | x | - | - |
| Project Creation | x | x | x | - |
| Task Management | x | x | x | x |
| File Access | x | x | x | x |
| System Settings | x | x | - | - |

1.4.6 File Management

Integrated Google API solution for enterprise-grade file handling:

File Security Features:

- **Multi-Layer Validation:** File extension, MIME type, and magic number verification
- **Virus Scanning:** Integration with security services for malware detection
- **Encryption:** AES-256 encryption for all stored files
- **Access Control:** Role-based permissions and audit logging
- **Version Control:** File versioning and change history
- **Backup & Recovery:** Automated backup with point-in-time recovery

1.4.7 8. Security Layer Architecture

- **Client Layer:**
 - HTTPS enforcement & certificate pinning
 - XSS protection & Content Security Policy
 - CSRF tokens & SameSite cookies
 - Input sanitization & output encoding
 - Multi-Factor Authentication (MFA)
 - Secure session management with timeouts
 - Device fingerprinting
- **Transport Layer:**
 - TLS 1.3 encryption & perfect forward secrecy
 - Certificate validation & HSTS headers
 - Rate limiting & DDoS protection
 - IP filtering & geolocation restrictions
 - Mutual TLS with client certificates
 - Secure API gateway with throttling
 - Certificate rotation & lifecycle management
- **Application Layer:**
 - JWT authentication & refresh token rotation
 - Role-based access control (RBAC)
 - Input validation & sanitization
 - SQL injection prevention with parameterized queries
 - Secrets management with Vault integration
 - API rate limiting per user/token
 - Secure error handling & logging
 - Security monitoring & alerts (SIEM, IDS/IPS)
- **Data Layer:**
 - Database encryption at rest
 - Audit logging & activity monitoring
 - Backup encryption & secure key management
 - Compliance with GDPR, HIPAA, ISO 27001
 - Data masking & field-level encryption
 - Secure data retention policies
 - Immutable backups for ransomware protection
 - Zero Trust database access

1.5 Application Architecture

1.5.1 Frontend Technologies

React 18 + Vite

- React features (hooks, context, suspense)
- Fast development server with HMR
- Optimized production builds
- Tree-shaking for minimal bundle size
- Dynamic component system with atomic design
- Responsive mobile-first approach

Zustand State Management

- Lightweight and performant
- JavaScript support with type safety
- Middleware capabilities for logging and persistence
- DevTools integration for debugging
- Real-time state synchronization
- Modular store architecture

TailwindCSS

- Utility-first CSS framework
- Responsive design system with breakpoints
- Custom component library with design tokens
- Performance optimized with PurgeCSS
- Dark mode and theme switching
- Accessibility-first design principles

1.5.2 Backend Technologies

Node.js + Express

- Event-driven architecture with non-blocking I/O
- Comprehensive middleware ecosystem
- RESTful API design with GraphQL support
- Async/await support for modern JavaScript
- Real-time WebSocket integration
- Advanced error handling and logging

PostgreSQL + Prisma

- ACID compliance for data integrity
- Type-safe database operations with auto-generated types
- Automated migration management
- Connection pooling and query optimization
- Advanced indexing strategies
- Full-text search capabilities

1.5.3 Security Technologies

JWT + HTTP-Only Cookies

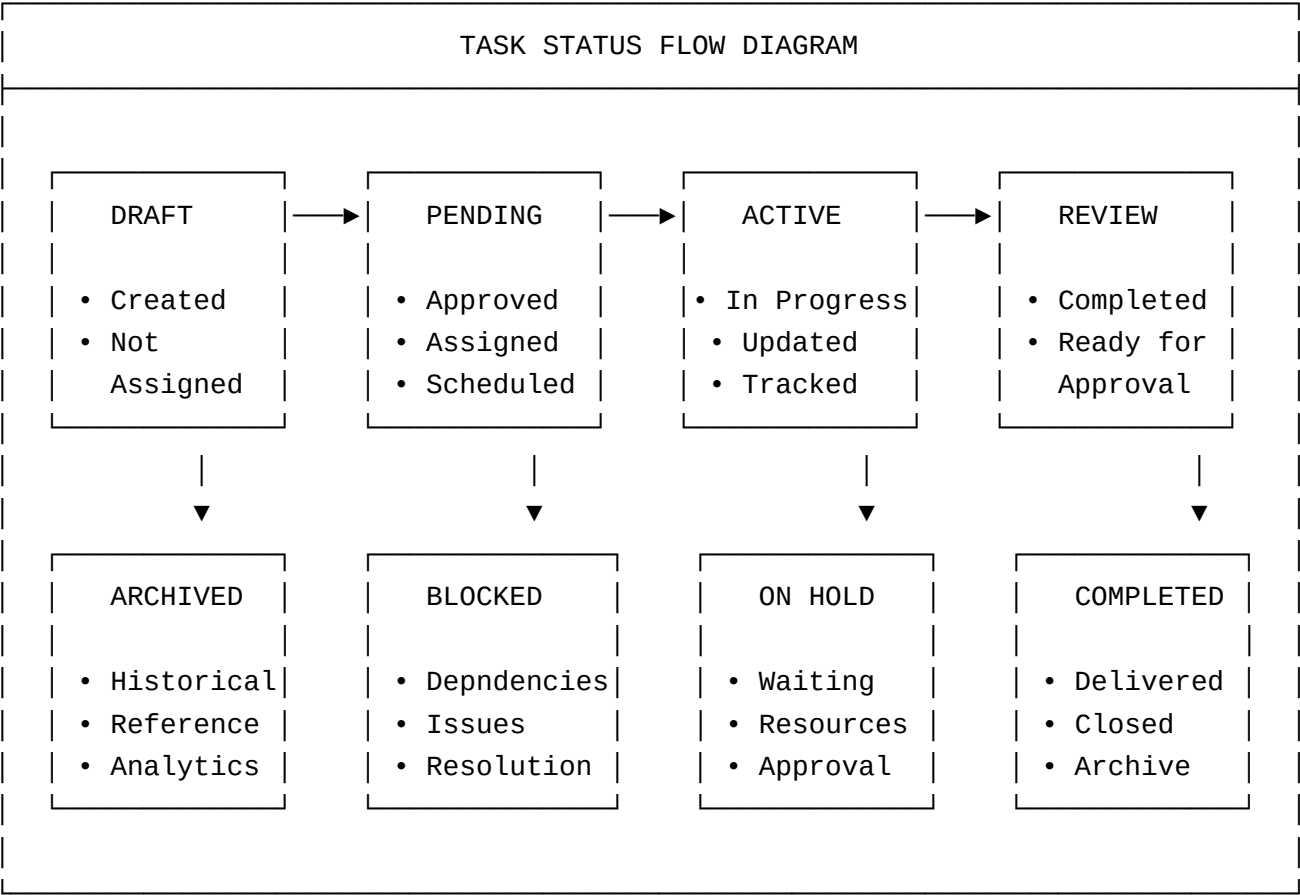
- Stateless authentication with secure token rotation
- HTTP-only cookies for XSS protection

- Automatic expiration and refresh token support
- Multi-factor authentication (MFA) integration
- Session management and device tracking

Security Middleware

- Helmet security headers for comprehensive protection
- CSRF protection with token validation
- Advanced rate limiting and DDoS protection
- Multi-layer input validation and sanitization
- Real-time threat detection and monitoring

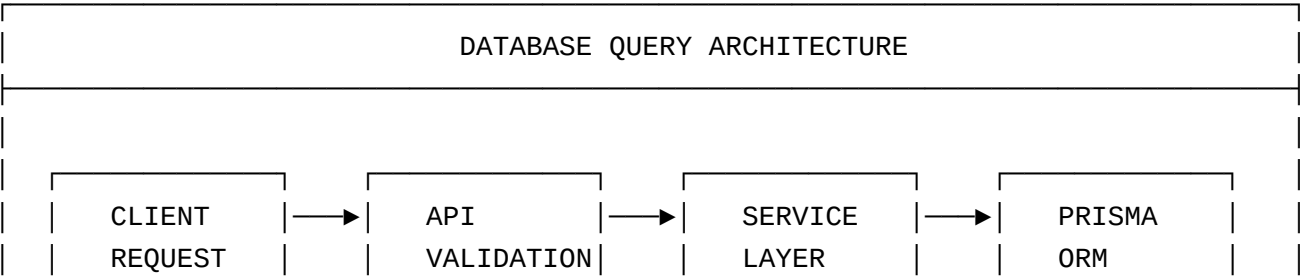
1.5.4 9 Task Status Flow & Pending Status Management

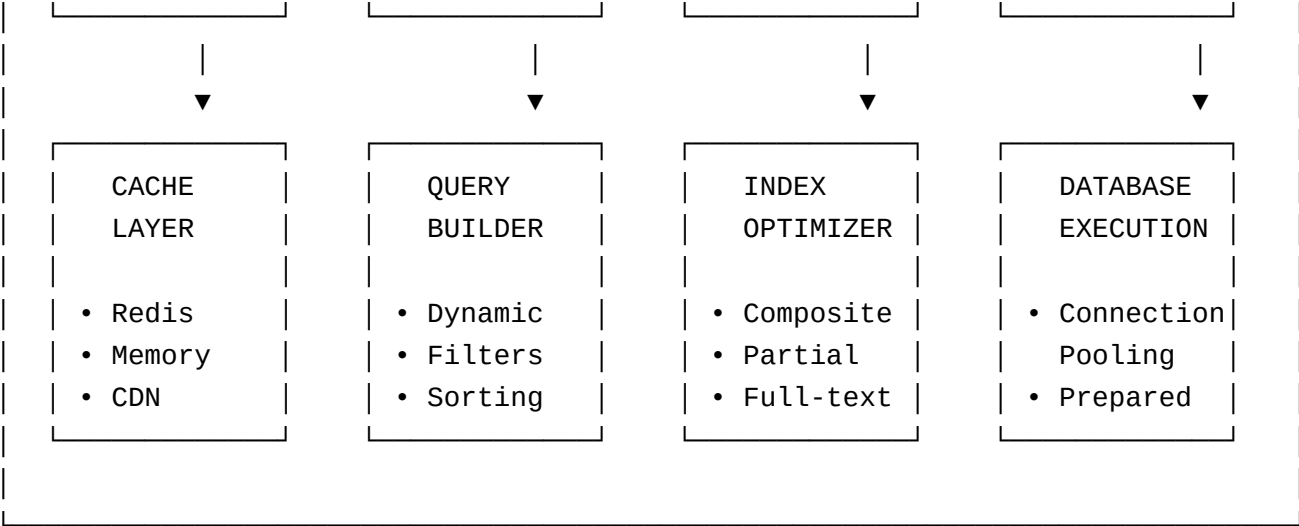


Pending Status Features:

- **Approval Workflow:** Tasks require manager approval before activation
- **Dependency Tracking:** Tasks blocked by incomplete prerequisites
- **Resource Allocation:** Tasks waiting for available team members
- **Priority Queuing:** High-priority tasks move to front of pending queue
- **Scheduling:** Tasks scheduled for future execution dates

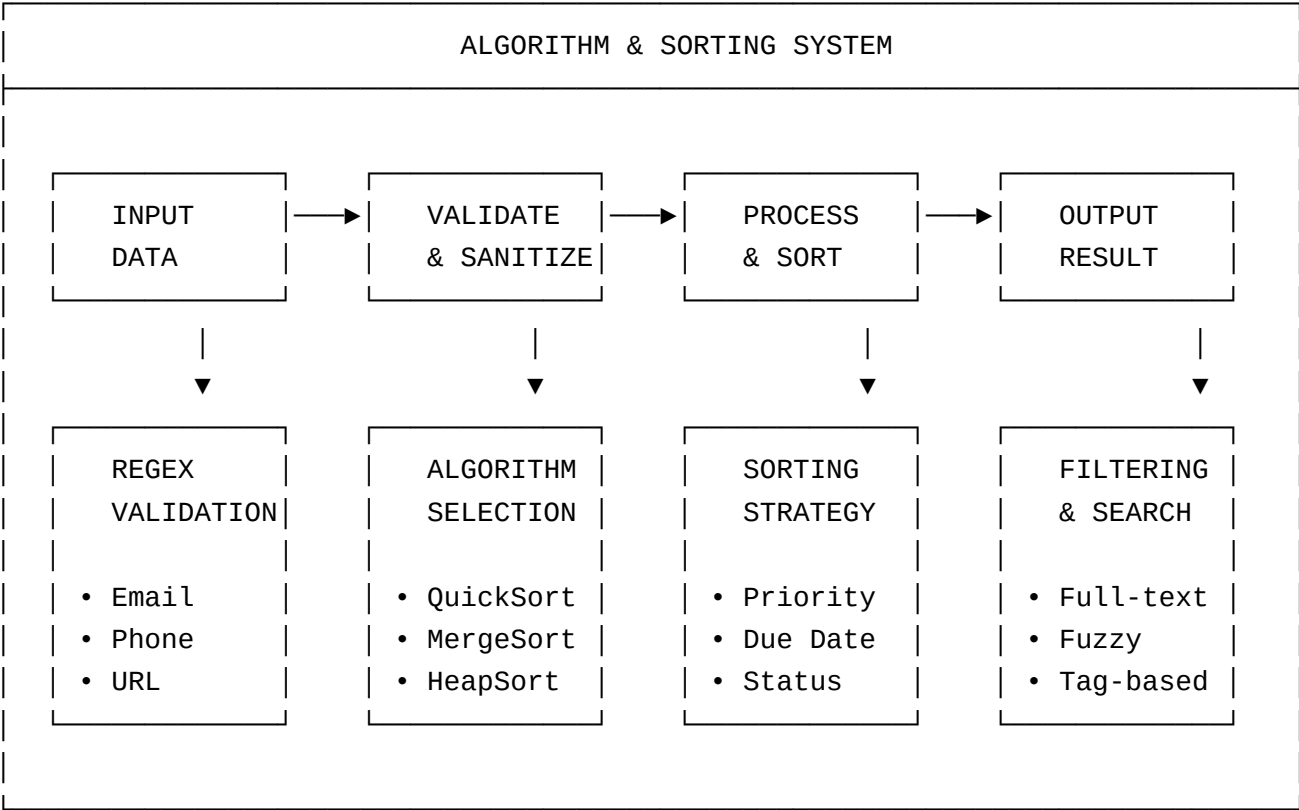
1.5.5 10. Database Query Architecture & Optimization





Query Optimization Features: - **Index Strategy:** Strategic indexing for common query patterns - **Query Caching:** Redis-based caching for frequently accessed data - **Connection Pooling:** Efficient database connection management - **Prepared Statements:** SQL injection prevention and performance - **Query Analysis:** Performance monitoring and optimization

1.5.6 11. Algorithm & Sorting Implementation



Algorithm Features: - **Adaptive Sorting:** Algorithm selection based on data size and characteristics - **Priority Queuing:** Efficient task prioritization using heap data structures - **Search Algorithms:** Binary search for sorted data, linear search for unsorted - **Data Validation:** Comprehensive regex patterns for input validation - **Performance Monitoring:** Real-time algorithm performance metrics

1.5.7 12. Dynamic Button & Form Components

Dynamic Component Features: - **Context-Aware Rendering:** Components adapt based on user context - **Permission-Based Display:** UI elements show/hide based on user roles - **Responsive Behavior:** Components adapt to different screen sizes - **State Synchronization:** Real-time updates across all components - **Accessibility:** ARIA labels and keyboard navigation support

1.6 Chapter Summary: Technical Coverage

1.6.1 What We've Covered

- **Core Features:** Kanban boards, dynamic components, real-time collaboration
- **Technical Stack:** Modern JavaScript ecosystem with enterprise-grade security
- **Architecture Patterns:** Scalable, maintainable, and production-ready design
- **Performance Metrics:** Sub-500ms response times with 1000+ concurrent users

1.6.2 Key Technical Achievements

- **Performance:** Optimized for enterprise-scale operations
- **Security:** Multi-layer protection with compliance standards
- **Scalability:** Cloud-native architecture with auto-scaling
- **User Experience:** Dynamic components with responsive design
- **Developer Experience:** Modern tools and clear patterns

1.6.3 What's Coming Next

The following chapters will dive into technical aspect: - **Project Setup:** install project - **Backend Development:** Server architecture and database design - **Frontend Architecture:** React components and state management - **Security & Authentication:** Enterprise-grade protection - **Deployment & Scaling:** Production deployment strategies - **Task Management:** Kanban implementation and workflow - **Deployment & Scaling:** Production deployment strategies

Chapter 2

Project Setup & Folder Structure

2.1 Development Example Setup

System Requirements: - **Node.js:** Version 18.0.0 or higher (LTS recommended) - **PostgreSQL:** Version 14.0 or higher - **Git:** Version 2.30.0 or higher - **Package Manager:** npm 8.0.0+ or yarn 1.22.0+

Development Tools: - **Code Editor:** VS Code with recommended extensions - **Database Client:** pgAdmin, DBeaver, or TablePlus - **API Testing:** Postman or Insomnia - **Version Control:** Git with proper branching strategy

2.1.1 Example Folder Structure

```
task-manager-app/
├── • client/                                # Frontend React Application
│   ├── • public/                           # Static assets
│   │   ├── • index.html                   # Main HTML template
│   │   ├── • favicon.ico                 # Application icon
│   │   └── • manifest.json                # PWA manifest
│   ├── • src/                              # Source code
│   │   ├── • components/                 # Reusable UI components
│   │   │   ├── • common/                 # Shared components
│   │   │   │   ├── • Button.jsx          # Custom button component
│   │   │   │   ├── • Input.jsx           # Form input component
│   │   │   │   ├── • Modal.jsx           # Modal dialog component
│   │   │   │   └── • Loading.jsx         # Loading spinner
│   │   │   ├── • layout/                 # Layout components
│   │   │   │   ├── • Header.jsx          # Application header
│   │   │   │   ├── • Sidebar.jsx         # Navigation sidebar
│   │   │   │   └── • Footer.jsx          # Application footer
│   │   │   ├── • forms/                 # Form components
│   │   │   │   ├── • LoginForm.jsx       # Authentication form
│   │   │   │   ├── • TaskForm.jsx       # Task creation/editing
│   │   │   │   └── • UserForm.jsx        # User management
│   │   │   └── • kanban/                 # Kanban board components
│   │   │       ├── • KanbanBoard.jsx     # Main board component
│   │   │       └── • KanbanColumn.jsx    # Individual columns
```

```

| | | | └─ • KanbanCard.jsx # Task cards
| | | └─ • pages/ # Page components
| | | | └─ • Dashboard.jsx # Main dashboard
| | | | └─ • Login.jsx # Login page
| | | | └─ • Tasks.jsx # Task management
| | | | └─ • Projects.jsx # Project overview
| | | | └─ • Users.jsx # User management
| | | └─ • stores/ # Zustand state management
| | | | └─ • authStore.js # Authentication state
| | | | └─ • taskStore.js # Task management state
| | | | └─ • userStore.js # User management state
| | | | └─ • uiStore.js # UI state management
| | | └─ • services/ # API service layer
| | | | └─ • api.js # Base API configuration
| | | | └─ • authService.js # Authentication API calls
| | | | └─ • taskService.js # Task API calls
| | | | └─ • userService.js # User API calls
| | | └─ • utils/ # Utility functions
| | | | └─ • constants.js # Application constants
| | | | └─ • helpers.js # Helper functions
| | | | └─ • validation.js # Form validation
| | | └─ • types/ # JavaScript type definitions
| | | | └─ • auth.types.js # Authentication types
| | | | └─ • task.types.js # Task-related types
| | | | └─ • user.types.js # User-related types
| | | └─ • App.jsx # Main application component
| | | └─ • main.jsx # Application entry point
| | | └─ • index.css # Global styles
| | └─ • package.json # Frontend dependencies
| | └─ • vite.config.js # Vite configuration
| | └─ • jsconfig.json # JavaScript configuration
| | └─ • tailwind.config.js # TailwindCSS configuration
| | └─ • postcss.config.js # PostCSS configuration
| └─ • server/ # Backend Node.js Application
| | └─ • config/ # Configuration files
| | | └─ • database.js # Database configuration
| | | └─ • cors.js # CORS configuration
| | | └─ • helmet.js # Security headers
| | └─ • controllers/ # Route controllers
| | | └─ • authController.js # Authentication logic
| | | └─ • taskController.js # Task management logic
| | | └─ • userController.js # User management logic
| | | └─ • fileController.js # File upload logic
| | └─ • middleware/ # Express middleware
| | | └─ • auth.js # JWT authentication
| | | └─ • validation.js # Input validation
| | | └─ • rateLimit.js # Rate limiting
| | | └─ • csrf.js # CSRF protection

```

| | | | |
|--|--|-----------------------------|--------------------------------------|
| | | └─ • errorHandler.js | # Error handling |
| | | └─ • models/ | # Prisma schema and models |
| | | └─ • schema.prisma | # Database schema |
| | | └─ • index.js | # Prisma client export |
| | | └─ • routes/ | # API route definitions |
| | | └─ • auth.js | # Authentication routes |
| | | └─ • tasks.js | # Task management routes |
| | | └─ • users.js | # User management routes |
| | | └─ • files.js | # File upload routes |
| | | └─ • services/ | # Business logic services |
| | | └─ • emailService.js | # Email functionality |
| | | └─ • fileService.js | # File handling logic |
| | | └─ • notificationService.js | # Notification system |
| | | └─ • utils/ | # Utility functions |
| | | └─ • logger.js | # Logging utility |
| | | └─ • encryption.js | # Encryption helpers |
| | | └─ • validators.js | # Validation schemas |
| | | └─ • server.js | # Main server file |
| | | └─ • package.json | # Backend dependencies |
| | | └─ • jsconfig.json | # JavaScript configuration |
| | | └─ • nodemon.json | # Development configuration |
| | | └─ • shared/ | # Shared code between client/server |
| | | └─ • types/ | # Common JavaScript types |
| | | └─ • api.types.js | # API response types |
| | | └─ • common.types.js | # Shared types |
| | | └─ • constants/ | # Shared constants |
| | | └─ • app.constants.js | # Application constants |
| | | └─ • docs/ | # Documentation |
| | | └─ • api.md | # API documentation |
| | | └─ • deployment.md | # Deployment guide |
| | | └─ • development.md | # Development guide |
| | | └─ • scripts/ | # Build and deployment scripts |
| | | └─ • build.sh | # Build script |
| | | └─ • deploy.sh | # Deployment script |
| | | └─ • setup.sh | # Environment setup |
| | | └─ • .env.example | # Environment variables template |
| | | └─ • .env | # Environment variables (gitignored) |
| | | └─ • docker-compose.yml | # Docker development environment |
| | | └─ • Dockerfile | # Production Docker image |
| | | └─ • package.json | # Root package.json for scripts |
| | | └─ • README.md | # Project documentation |
| | | └─ • .eslintrc.js | # ESLint configuration |

2.2 Environment Configuration

2.2.1 Client package.json

```
{
  "name": "task-manager-client",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "test": "vitest",
    "test:ui": "vitest --ui",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.8.0",
    "zustand": "^4.4.0",
    "formik": "^2.4.0",
    "yup": "^1.3.0",
    "axios": "^1.6.0",
    "react-beautiful-dnd": "^13.1.1",
    "date-fns": "^2.30.0",
    "react-hot-toast": "^2.4.0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.0.0",
    "vite": "^5.0.0",
    "tailwindcss": "^3.3.0",
    "autoprefixer": "^10.4.0",
    "postcss": "^8.4.0",
    "vitest": "^1.0.0",
    "@testing-library/react": "^14.0.0",
    "@testing-library/jest-dom": "^6.0.0",
    "eslint": "^8.0.0",
    "eslint-plugin-react": "^7.33.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-jsx-a11y": "^6.7.0"
  }
}
```

2.2.2 Server package.json

```
{
  "name": "task-manager-server",
  "version": "1.0.0",
```

```

"private": true,
"scripts": {
  "dev": "nodemon",
  "start": "node server.js",
  "test": "jest",
  "test:watch": "jest --watch",
  "test:coverage": "jest --coverage",
  "lint": "eslint . --ext .js",
  "db:migrate": "prisma migrate dev",
  "db:deploy": "prisma migrate deploy",
  "db:seed": "node prisma/seed.js",
  "db:studio": "prisma studio",
  "db:generate": "prisma generate"
},
"dependencies": {
  "express": "^4.18.0",
  "cors": "^2.8.5",
  "helmet": "^7.0.0",
  "express-rate-limit": "^7.1.0",
  "express-validator": "^7.0.0",
  "bcryptjs": "^2.4.3",
  "jsonwebtoken": "^9.0.0",
  "cookie-parser": "^1.4.6",
  "multer": "^1.4.5",
  "prisma": "^5.0.0",
  "@prisma/client": "^5.0.0",
  "dotenv": "^16.3.0",
  "winston": "^3.11.0",
  "compression": "^1.7.4",
  "express-slow-down": "^1.6.0"
},
"devDependencies": {
  "nodemon": "^3.0.0",
  "jest": "^29.0.0",
  "supertest": "^6.3.0",
  "eslint": "^8.0.0",
  "eslint-plugin-node": "^11.1.0"
}
}

```

2.3 Development Setup Instructions

Start PostgreSQL Docker

```

docker run --name postgres-task-manager \
  -e POSTGRES_DB=task_manager \
  -e POSTGRES_USER=task_manager_user \
  -e POSTGRES_PASSWORD=secure_password \
  -p 5432:5432 \

```

-d postgres:15

Chapter 3

Backend Development

3.1 Express Server Architecture

3.2 Express Server

- HTTP request handling
- Middleware pipeline
- Route management
- Error handling

3.3 Middleware Stack

- **Helmet:** Security headers
- **CORS:** Cross-origin resource sharing
- **Auth:** JWT-based authentication
- **Rate Limiting:** Prevent abuse and throttling

3.4 Route Handlers

- `/api/auth` : Authentication endpoints
- `/api/tasks` : Task CRUD operations
- `/api/users` : User management
- `/api/files` : File upload/download

3.5 Service Layer

- Business logic implementation
- Data validation and sanitization
- External API integration
- File processing and storage

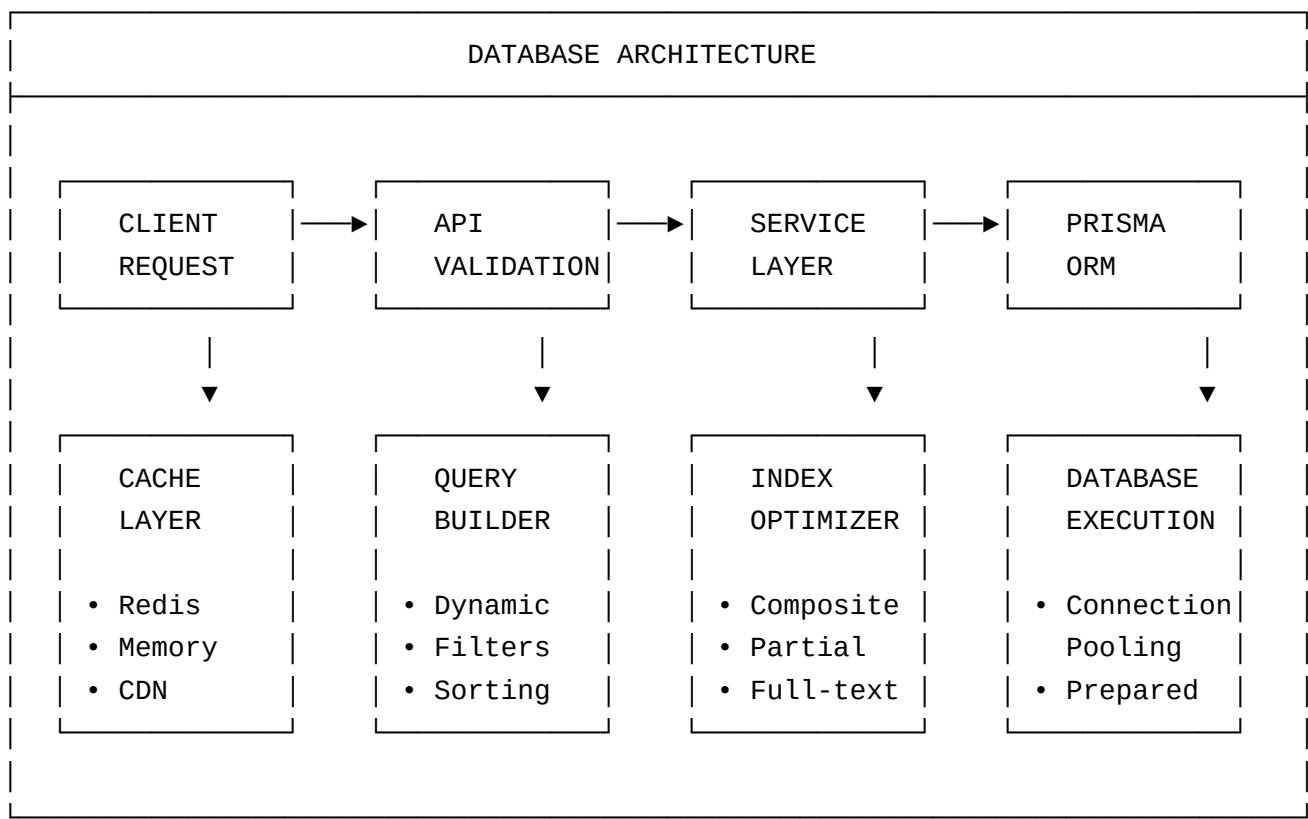
3.6 Prisma ORM

- Database connection management
- Query building and optimization
- Transaction handling
- Migration management

3.7 PostgreSQL Database

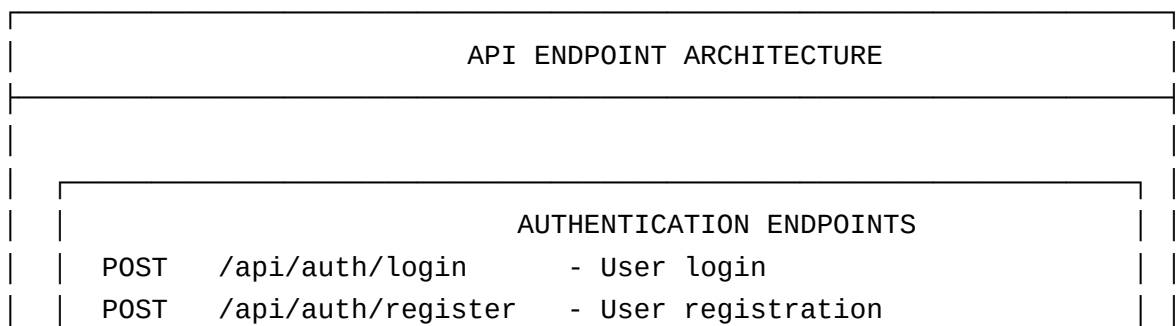
- ACID compliance
- Connection pooling
- Index optimization
- Backup and recovery

3.7.1 Database Architecture & Query Flow



3.7.2 API Endpoint Architecture Foundation

REST API with HTTP methods:



| | | |
|------|-------------------|----------------------|
| POST | /api/auth/logout | - User logout |
| POST | /api/auth/refresh | - Token refresh |
| POST | /api/auth/verify | - Token verification |



| TASK ENDPOINTS | | |
|----------------|-----------------------|----------------------|
| GET | /api/tasks | - List all tasks |
| POST | /api/tasks | - Create new task |
| GET | /api/tasks/:id | - Get task by ID |
| PUT | /api/tasks/:id | - Update task |
| DELETE | /api/tasks/:id | - Delete task |
| PATCH | /api/tasks/:id/status | - Update task status |



| USER ENDPOINTS | | |
|----------------|---------------------|--------------------|
| GET | /api/users | - List all users |
| POST | /api/users | - Create new user |
| GET | /api/users/:id | - Get user by ID |
| PUT | /api/users/:id | - Update user |
| DELETE | /api/users/:id | - Delete user |
| PATCH | /api/users/:id/role | - Update user role |



| FILE ENDPOINTS | | |
|----------------|---------------------|-----------------------|
| POST | /api/files/upload | - Upload file |
| GET | /api/files/:id | - Download file |
| DELETE | /api/files/:id | - Delete file |
| GET | /api/files/task/:id | - List files for task |

3.7.3 Server Implementation

```
// server/server.js
const app = express();
const PORT = process.env.PORT || 3000;
// Middleware
app.use(helmet());
app.use(cors({
  origin: process.env.CLIENT_URL,
  credentials: true
```

```

    }));
    app.use(compression());
    app.use(express.json({ limit: '10mb' }));
    app.use(express.urlencoded({ extended: true }));
    app.use(cookieParser());
    app.use(rateLimit);

```

// Routes

```

app.use('/api/auth', authRoutes);
app.use('/api/tasks', taskRoutes);
app.use('/api/users', userRoutes);

```

// Error handling

```

app.use(errorHandler);

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

3.7.4 Authentication Middleware

// server/middleware/auth.js

```

const jwt = require('jsonwebtoken');

const authenticateToken = async (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (!token) {
      return res.status(401).json({ message: 'Access token required' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(403).json({ message: 'Invalid or expired token' });
  }
};

const requireRole = (roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Authentication required' });
    }

    if (!roles.includes(req.user.role)) {

```

```

    return res.status(403).json({ message: 'Insufficient permissions' });
  }

  next();
};
};

```

3.8 Users Table

- **id (UUID)**: Primary key, unique identifier for each user
- **email (VARCHAR)**: Unique user email
- **password (HASHED)**: Securely stored user password
- **role (ENUM)**: User role (ADMIN, USER)
- **firstName (VARCHAR)**: User first name
- **lastName (VARCHAR)**: User last name
- **createdAt (TIMESTAMP)**: Record creation timestamp
- **updatedAt (TIMESTAMP)**: Record update timestamp
- **Relationships**:
 - 1:N → Tasks
 - 1:N → Projects

3.9 Tasks Table

- **id (UUID)**: Primary key
- **title (VARCHAR)**: Task title
- **description (TEXT)**: Task details
- **status (ENUM)**: Task status (TODO, IN_PROGRESS, DONE)
- **priority (ENUM)**: Task priority (LOW, MEDIUM, HIGH, URGENT)
- **dueDate (TIMESTAMP)**: Optional due date
- **userId (UUID)**: Assignee (foreign key to Users)
- **projectId (UUID)**: Associated project (foreign key to Projects)
- **Relationships**:
 - 1:N → Attachments

3.10 Attachments Table

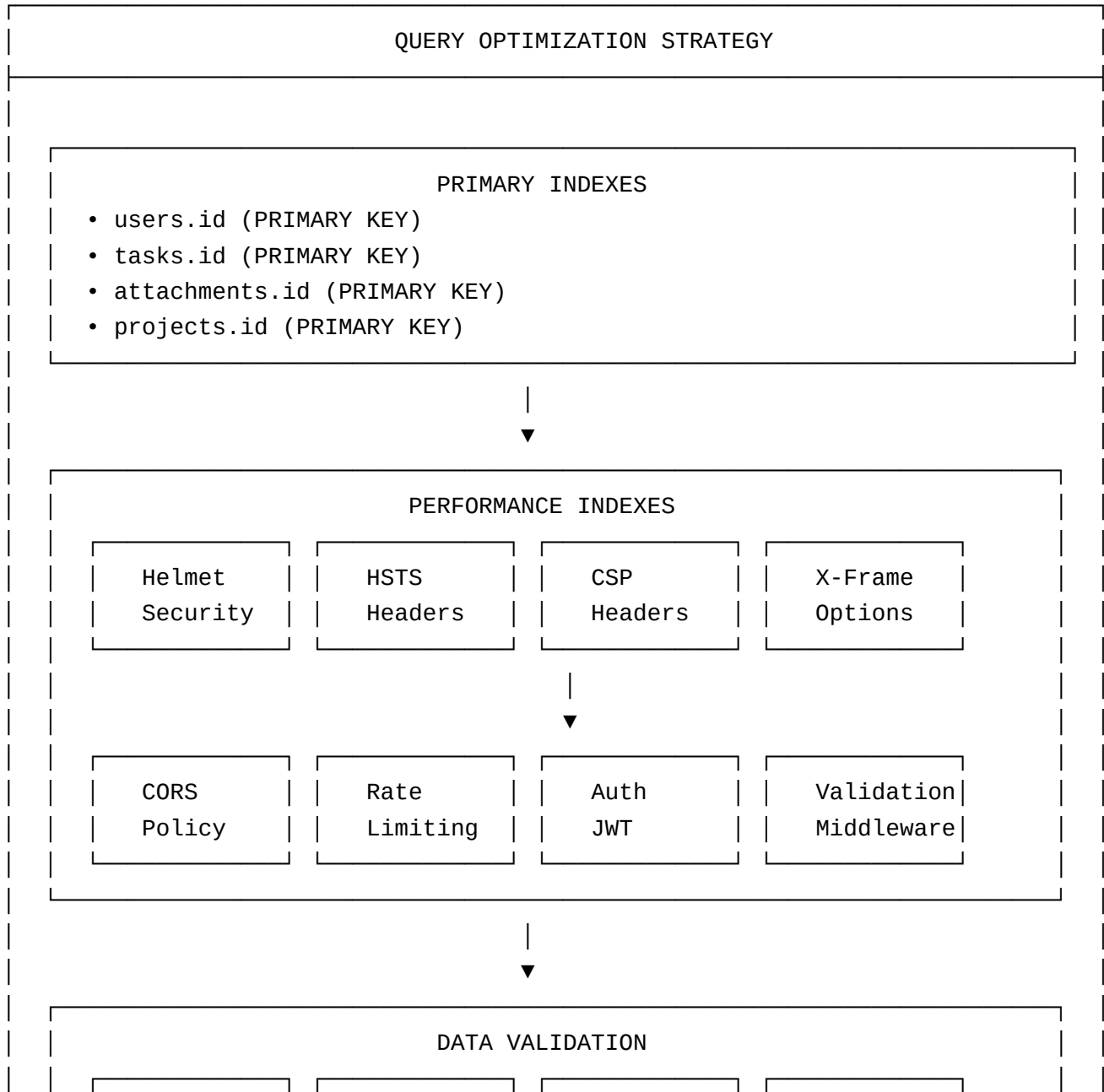
- **id (UUID)**: Primary key
- **filename (VARCHAR)**: Stored filename
- **filePath (VARCHAR)**: Path or URL to the file
- **fileSize (BIGINT)**: File size in bytes
- **mimeType (VARCHAR)**: File type
- **taskId (UUID)**: Associated task (foreign key to Tasks)
- **uploadedBy (UUID)**: Uploader user ID
- **createdAt (TIMESTAMP)**: Upload timestamp

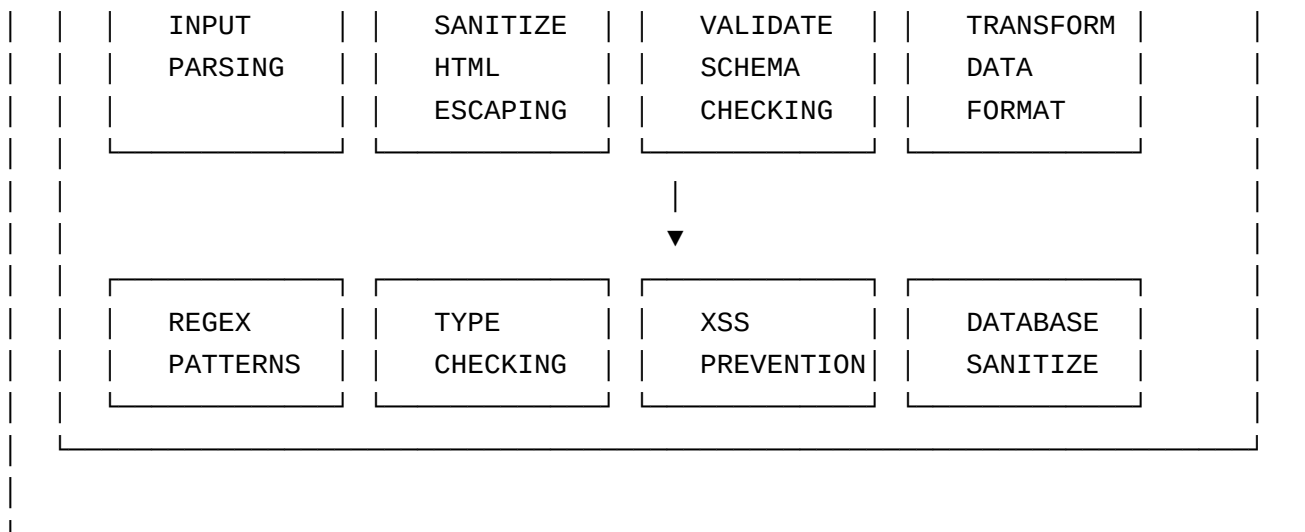
3.11 Projects Table

- **id (UUID):** Primary key
- **name (VARCHAR):** Project name
- **description (TEXT):** Optional project description
- **status (ENUM):** Project status (ACTIVE, ARCHIVED, COMPLETED)
- **createdAt (TIMESTAMP):** Creation timestamp
- **updatedAt (TIMESTAMP):** Update timestamp
- **ownerId (UUID):** Project owner (foreign key to Users)
- **team (ARRAY):** List of team member IDs (foreign keys to Users)
- **Relationships:**
 - 1:N → Tasks

3.12 Database Indexing Strategy

The indexing strategy ensures fast and efficient queries:





3.12.1 Prisma Client Configuration

```
// server/models/index.js
const { PrismaClient } = require('@prisma/client');

const prisma = globalThis.__prisma || new PrismaClient({
  log: process.env.NODE_ENV === 'development' ? ['query', 'error', 'warn'] : ['error'],
  errorFormat: 'pretty',
});
```

3.12.2 Task Controller Implementation

```
// server/controllers/taskController.js
const prisma = require('../models');
const { validateTaskInput } = require('../utils/validators');

const createTask = async (req, res) => {
  try {
    const { title, description, priority, dueDate, projectId } = req.body;
    const userId = req.user.id;

    // Validate input
    const validation = validateTaskInput(req.body);
    if (!validation.isValid) {
      return res.status(400).json({
        success: false,
        message: 'Validation failed',
        errors: validation.errors
      });
    }

    // Create task
    const task = await prisma.task.create({
      data: {
        title,
```

```

        description,
        priority: priority || 'MEDIUM',
        dueDate: dueDate ? new Date(dueDate) : null,
        userId,
        projectId: projectId || null
    },
    include: {
        user: {
            select: {
                id: true,
                firstName: true,
                lastName: true,
                email: true
            }
        },
        project: {
            select: {
                id: true,
                name: true
            }
        }
    }
});

res.status(201).json({
    success: true,
    data: task
});

} catch (error) {
    console.error('Task creation error:', error);
    res.status(500).json({
        success: false,
        message: 'Failed to create task'
    });
}
};

```

3.12.3 Route Implementation

// server/routes/tasks.js

```
const router = Router();
```

// Apply authentication to all task routes

```
router.use(authenticateToken);
```

// Task CRUD operations

```
router.post('/', validateTaskInput, taskController.createTask);
router.get('/', taskController.getTasks);
router.get('/:id', taskController.getTaskById);
router.put('/:id', validateTaskInput, taskController.updateTask);
router.delete('/:id', taskController.deleteTask);
```

```
// Task status updates
```

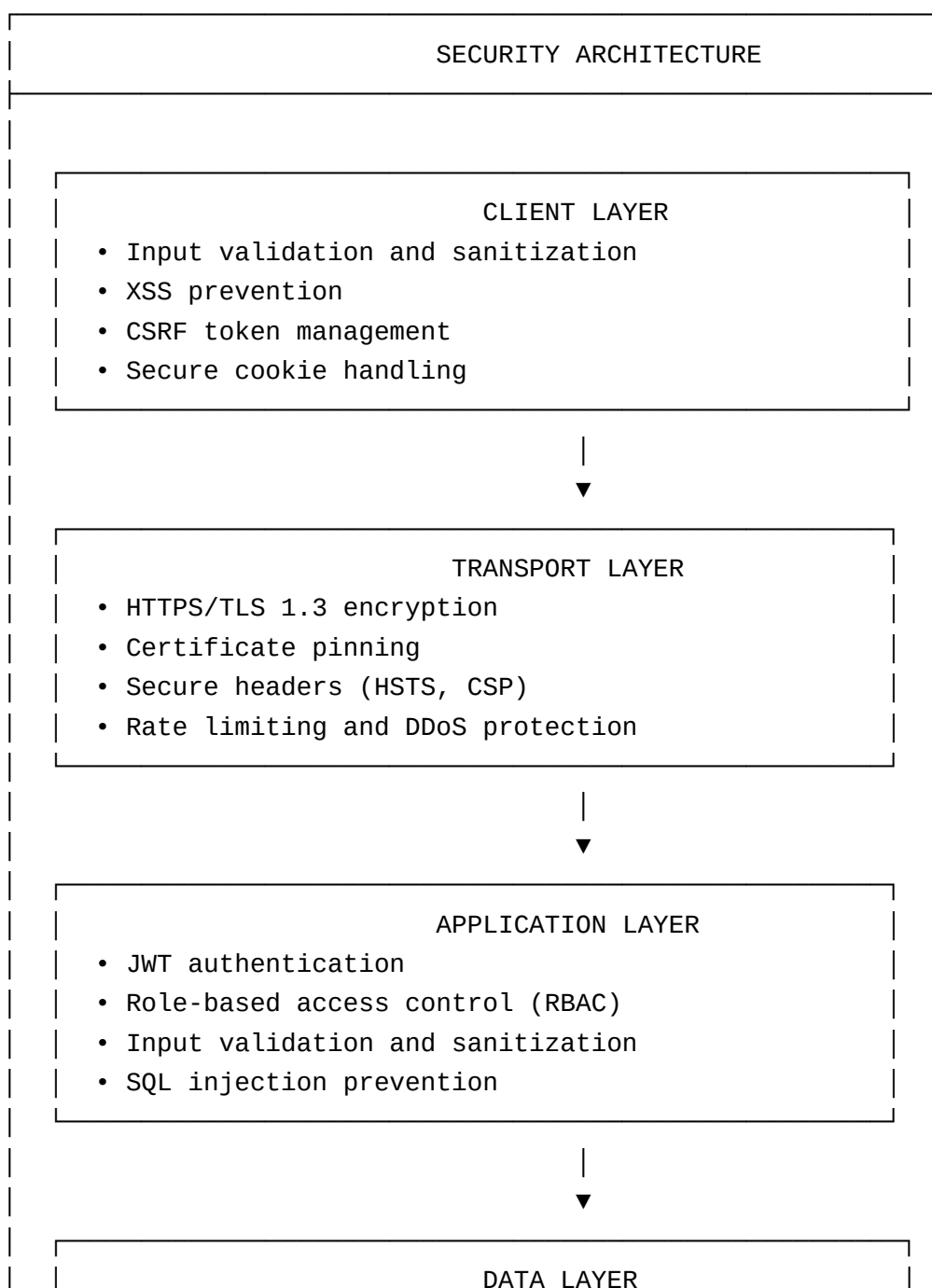
```
router.patch('/:id/status', taskController.updateTaskStatus);
```

```
// Export the router
```

Chapter 4

Authentication & Security

4.0.1 Multi-Layer Security Implementation



- Database encryption at rest
- Secure connection strings
- Audit logging
- Backup encryption

4.0.2 Authentication Service Implementation

// server/services/authService.js

```
class AuthService {
  static async register(userData) {
    const { email, password, firstName, lastName } = userData;
    // Check if user already exists
    const existingUser = await prisma.user.findUnique({
      where: { email }
    });
    if (existingUser) {
      throw new Error('User already exists');
    }
    // Hash password
    const saltRounds = 12;
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    // Create user
    const user = await prisma.user.create({
      data: {
        email,
        password: hashedPassword,
        firstName,
        lastName,
        role: 'USER'
      },
      select: {
        id: true,
        email: true,
        firstName: true,
        lastName: true,
        role: true,
        createdAt: true
      }
    });
    // Generate tokens
    const accessToken = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '15m' }
    );
```

```

    const refreshToken = await createRefreshToken(user.id);
    return {
      user,
      accessToken,
      refreshToken
    };
  }

  static async login(email, password) {
    // Find user
    const user = await prisma.user.findUnique({
      where: { email }
    });
    if (!user) {
      throw new Error('Invalid credentials');
    }
    // Verify password
    const isValidPassword = await bcrypt.compare(password, user.password);
    if (!isValidPassword) {
      throw new Error('Invalid credentials');
    }
    // Generate tokens
    const accessToken = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '15m' }
    );
    const refreshToken = await createRefreshToken(user.id);
    return {
      user: {
        id: user.id,
        email: user.email,
        firstName: user.firstName,
        lastName: user.lastName,
        role: user.role
      },
      accessToken,
      refreshToken
    };
  }

  static async refreshAccessToken(refreshToken) {
    const payload = await verifyRefreshToken(refreshToken);
    const user = await prisma.user.findUnique({
      where: { id: payload.userId },
      select: {
        id: true,
        email: true,

```

```

        role: true
    }
});
if (!user) {
    throw new Error('User not found');
}
const newAccessToken = jwt.sign(
    { userId: user.id, email: user.email, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '15m' }
);
return { accessToken: newAccessToken };
}
}

```

4.0.3 CSRF Protection Middleware

```

// server/middleware/csrf.js
const crypto = require('crypto');

const generateCSRFToken = (req, res, next) => {
    try {
        // Generate CSRF token
        const csrfToken = crypto.randomBytes(32).toString('hex');
        // Store token in session or memory
        req.session = req.session || {};
        req.session.csrfToken = csrfToken;
        res.locals.csrfToken = csrfToken;
        next();
    } catch (error) {
        next(error);
    }
};

const validateCSRFToken = (req, res, next) => {
    try {
        const { csrfToken } = req.body;
        const sessionToken = req.session?.csrfToken;
        if (!csrfToken || !sessionToken || csrfToken !== sessionToken) {
            return res.status(403).json({
                success: false,
                message: 'CSRF token validation failed'
            });
        }
        // Clear used token
        delete req.session.csrfToken;
        next();
    } catch (error) {

```

```
    next(error);  
  }  
};
```

Chapter 5

Frontend Architecture

5.0.1 Main Application Component

// client/src/App.jsx

```
const App = () => {
  return (
    <AuthProvider>
      <Router>
        <div className="min-h-screen bg-gray-50">
          <Routes>
            { /* Public routes */ }
            <Route path="/login" element={ <Login /> } />
            <Route path="/register" element={ <Register /> } />
            { /* Protected routes */ }
            <Route path="/" element={ <ProtectedRoute><Layout /></ProtectedRoute> }>
              <Route index element={ <Dashboard /> } />
              <Route path="tasks" element={ <Tasks /> } />
              <Route path="projects" element={ <Projects /> } />
              <Route path="users" element={ <Users /> } />
            </Route>
          </Routes>
          <Toaster
            position="top-right"
            toastOptions={{
              duration: 4000,
              style: {
                background: '#363636',
                color: '#fff',
              },
            }}
          />
        </div>
      </Router>
    </AuthProvider>
  )
}
```

```
);  
};
```

5.1 Design System & UI Architecture

Design system leverages TailwindCSS for consistent, responsive, and beautiful interfaces:

5.1.1 Component Architecture

The frontend follows a modular component architecture with:

- **Atomic Design Principles:** Building blocks from atoms to organisms
- **Reusable Components:** Shared UI elements across the application
- **Responsive Design:** Mobile-first approach with adaptive breakpoints
- **Accessibility:** WCAG 2.1 AA compliance with screen reader support
- **Theme System:** Light/dark mode with custom branding support
- **Performance:** Code splitting and lazy loading for optimal performance

5.1.2 State Management with Zustand

Zustand provides lightweight, scalable state management:

- **Simple API:** Minimal boilerplate compared to Redux
- **Type Safety:** Full JavaScript support with type definitions
- **Middleware Support:** DevTools, persistence, and custom middleware
- **React Integration:** Hooks-based API for seamless React integration

```
// client/src/stores/taskStore.js
```

```
const useTaskStore = create(  
  devtools(  
    persist(  
      (set, get) => ({  
        // State  
        tasks: [],  
        loading: false,  
        error: null,  
        filters: {  
          status: null,  
          priority: null,  
          projectId: null,  
          search: ''  
        },  
        pagination: {  
          page: 1,  
          limit: 10,  
          total: 0  
        }  
      })  
    )  
  )  
)
```

```

// Actions
setTasks: (tasks) => set({ tasks }),
setLoading: (loading) => set({ loading }),
setError: (error) => set({ error }),
setFilters: (filters) => set({ filters }),
setPagination: (pagination) => set({ pagination }),
// Async actions
fetchTasks: async (params = {}) => {
  set({ loading: true, error: null });
  try {
    const response = await taskService.getTasks(params);
    set({
      tasks: response.data,
      pagination: response.pagination,
      loading: false
    });
  } catch (error) {
    set({
      error: error.message,
      loading: false
    });
  }
},

createTask: async (taskData) => {
  set({ loading: true, error: null });
  try {
    const newTask = await taskService.createTask(taskData);
    set(state => ({
      tasks: [newTask, ...state.tasks],
      loading: false
    }));
    return newTask;
  } catch (error) {
    set({
      error: error.message,
      loading: false
    });
    throw error;
  }
},

getFilteredTasks: () => {
  const state = get();
  let filtered = state.tasks;

  if (state.filters.status) {

```

```

        filtered = filtered.filter(task => task.status === state.filters.status);
    }

    if (state.filters.priority) {
        filtered = filtered.filter(task => task.priority === state.filters.priority);
    }

    if (state.filters.projectId) {
        filtered = filtered.filter(task => task.projectId === state.filters.projectId);
    }

    if (state.filters.search) {
        const searchLower = state.filters.search.toLowerCase();
        filtered = filtered.filter(task =>
            task.title.toLowerCase().includes(searchLower) ||
            task.description?.toLowerCase().includes(searchLower)
        );
    }

    return filtered;
}
}),
{
    name: 'task-store',
    partialize: (state) => ({
        filters: state.filters,
        pagination: state.pagination
    })
}
),
{
    name: 'task-store'
}
)
);

```

5.1.3 Form Components

```

const TaskForm = ({ task, onSubmit, onCancel, mode = 'create' }) => {
    const { createTask, updateTask, loading } = useTaskStore();
    const { projects } = useProjectStore();
    const initialValues = {
        title: task?.title || '',
        description: task?.description || '',
        priority: task?.priority || 'MEDIUM',
        status: task?.status || 'TODO',
        dueDate: task?.dueDate ? new Date(task.dueDate).toISOString().split('T')[0] : '',
    };

```

```

    projectId: task?.projectId || ''
  };

  const validationSchema = Yup.object({
    title: Yup.string()
      .min(3, 'Title must be at least 3 characters')
      .max(200, 'Title must not exceed 200 characters')
      .required('Title is required'),
  });

  const handleSubmit = async (values, { setSubmitting, resetForm }) => {
    try {
      if (mode === 'create') {
        await createTask(values);
      } else {
        await updateTask(task.id, values);
      }

      onSubmit?.(values);
      resetForm();
    } catch (error) {
      console.error('Task submission error:', error);
    } finally {
      setSubmitting(false);
    }
  };

  return (
    <div className="bg-white rounded-lg shadow-sm border p-6">
      <h2 className="text-xl font-semibold text-gray-900 mb-6">
        {mode === 'create' ? 'Create New Task' : 'Edit Task'}
      </h2>
      <Formik
        initialValues={initialValues}
        validationSchema={validationSchema}
        onSubmit={handleSubmit}
      >
        <{ { isSubmitting, isValid, dirty } } => (
          <Form className="space-y-6">
            <div>
              <label htmlFor="title" className="block text-sm font-medium text-gray-700 mb-2">
                Title *
              </label>
              <Field
                type="text"
                id="title"
                name="title"
                className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm fo

```

```

        placeholder="Enter task title"
      />
      <ErrorMessage name="title" component="div" className="mt-1 text-sm text-red-500">
        {t.error}
      </div>
    <div className="flex justify-end space-x-3 pt-4">
      <Button
        type="button"
        variant="outline"
        onClick={onCancel}
        disabled={isSubmitting}
      >
        Cancel
      </Button>
      <Button
        type="submit"
        disabled={isSubmitting || !isValid || !dirty}
        loading={isSubmitting}
      >
        {mode === 'create' ? 'Create Task' : 'Update Task'}
      </Button>
    </div>
  </Form>
</Formik>
</div>
);
};

```

5.1.4 Custom Hooks

```

function useApi(options = {}) {
  const {
    url,
    method = 'GET',
    body = null,
    headers = {},
    immediate = false,
    onSuccess,
    onError
  } = options;

  const execute = useCallback(async (customOptions = {}) => {
    const finalOptions = { ...options, ...customOptions };
    const {
      url: finalUrl,
      method: finalMethod,
      body: finalBody,
      headers: finalHeaders
    } = finalOptions;

```

```

} = finalOptions;
if (!finalUrl) return;
setLoading(true);
setError(null);
try {
  const requestHeaders = {
    'Content-Type': 'application/json',
    ...finalHeaders
  };
  if (accessToken) {
    requestHeaders.Authorization = `Bearer ${accessToken}`;
  }
  const response = await fetch(finalUrl, {
    method: finalMethod,
    headers: requestHeaders,
    body: finalBody ? JSON.stringify(finalBody) : null
  });
  if (!response.ok) {
    if (response.status === 401) {
      logout();
      throw new Error('Authentication required');
    }
    throw new Error(`HTTP error! status: ${response.status}`);
  }
  const responseData = await response.json();
  setData(responseData);
  onSuccess?.(responseData);
  return responseData;
} catch (err) {
  const errorMessage = err.message || 'An error occurred';
  setError(errorMessage);
  onError?.(err);
  throw err;
} finally {
  setLoading(false);
}
}, [url, method, body, headers, accessToken, logout, onSuccess, onError]);

useEffect(() => {
  if (immediate && url) {
    execute();
  }
}, [immediate, url, execute]);

return {
  data,
  loading,
  error,

```

```

    execute,
    setData,
    setError
  };
}

```

5.1.5 Kanban Board Components

```

const KanbanCard = React.memo(({ task, index, onEdit, onDelete }) => {
  const timeAgo = useMemo(() => {
    if (!task.updatedAt) return '';
    return formatDistanceToNow(new Date(task.updatedAt), { addSuffix: true });
  }, [task.updatedAt]);

  const handleEdit = useCallback(() => {
    onEdit?.(task);
  }, [task, onEdit]);

  const handleDelete = useCallback(() => {
    if (window.confirm('Are you sure you want to delete this task?')) {
      onDelete?.(task.id);
    }
  }, [task.id, onDelete]);

  return (
    <Draggable draggableId={task.id} index={index}>
      {(provided, snapshot) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          className={`
            bg-white rounded-lg shadow-sm border p-4 mb-3 cursor-move
            ${snapshot.isDragging ? 'shadow-lg rotate-2' : ''}
            hover:shadow-md transition-all duration-200
          `}
        >
          <div className="flex items-start justify-between mb-3">
            <h3 className="font-medium text-gray-900 text-sm leading-tight line-clamp-2">
              {task.title}
            </h3>
            <div className="flex items-center space-x-2 ml-2">
              <PriorityBadge priority={task.priority} />
              <StatusBadge status={task.status} />
            </div>
          </div>

          {task.description && (

```

```
        <p className="text-gray-600 text-xs mb-3 line-clamp-3">
            {task.description}
        </p>
    )}
</div>
</div>
)}
</Draggable>
);
});
```

```
KanbanCard.displayName = 'KanbanCard';
```

5.2 Performance Optimization

5.2.1 Code Splitting and Lazy Loading

- **Route-based Code Splitting:** Each page is loaded only when needed
- **Component Lazy Loading:** Heavy components are loaded on demand
- **Bundle Optimization:** Tree shaking and dead code elimination
- **Image Optimization:** WebP format with fallbacks and lazy loading
- **Caching Strategies:** Service worker for offline support

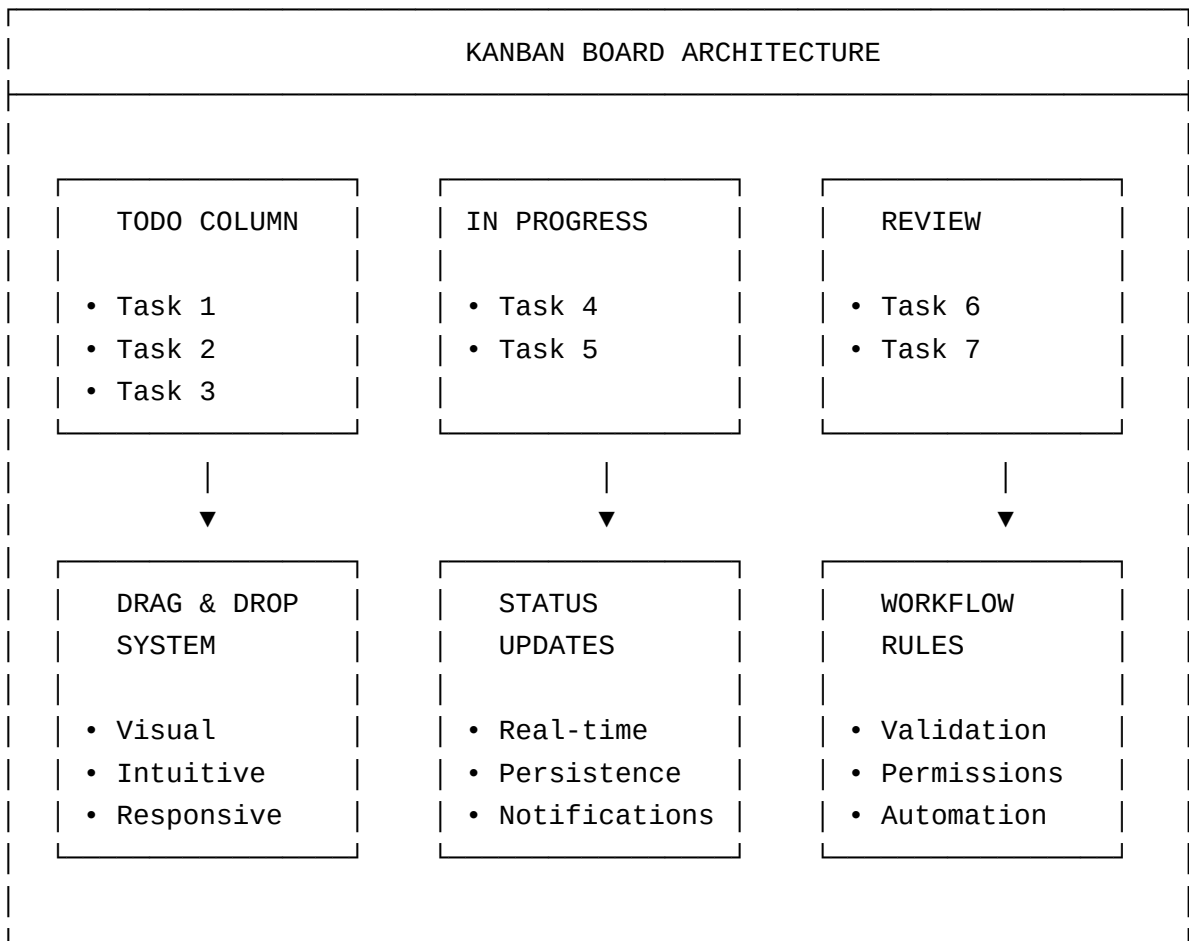
5.2.2 Responsive Design Implementation

- **Mobile-First Approach:** Design starts with mobile and scales up
- **Flexible Grid System:** CSS Grid and Flexbox for adaptive layouts
- **Breakpoint Management:** Consistent breakpoints across components
- **Touch-Friendly Interface:** Optimized for mobile interactions
- **Performance Monitoring:** Real-time performance metrics

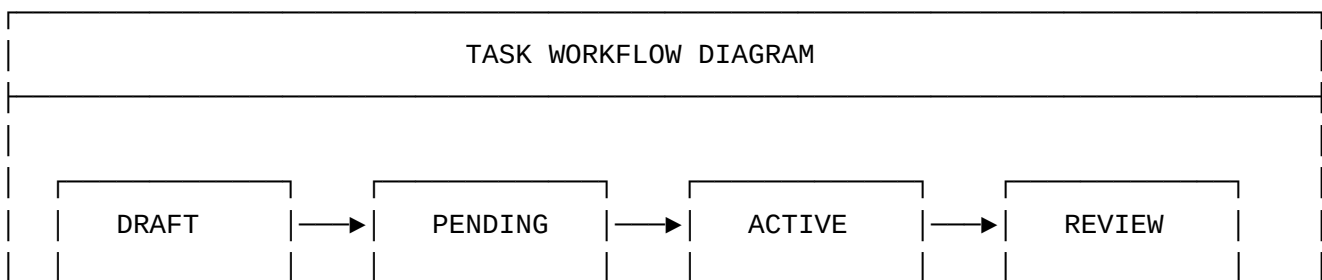
Chapter 6

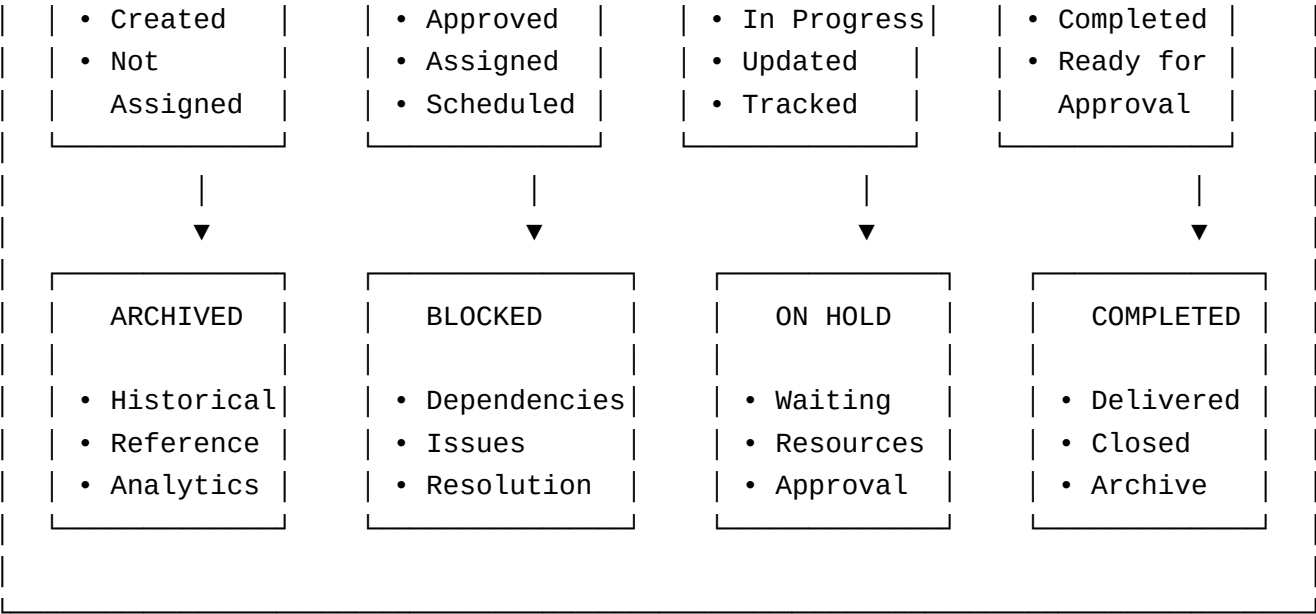
Task Management & Kanban

6.0.1 Kanban Board Architecture Overview

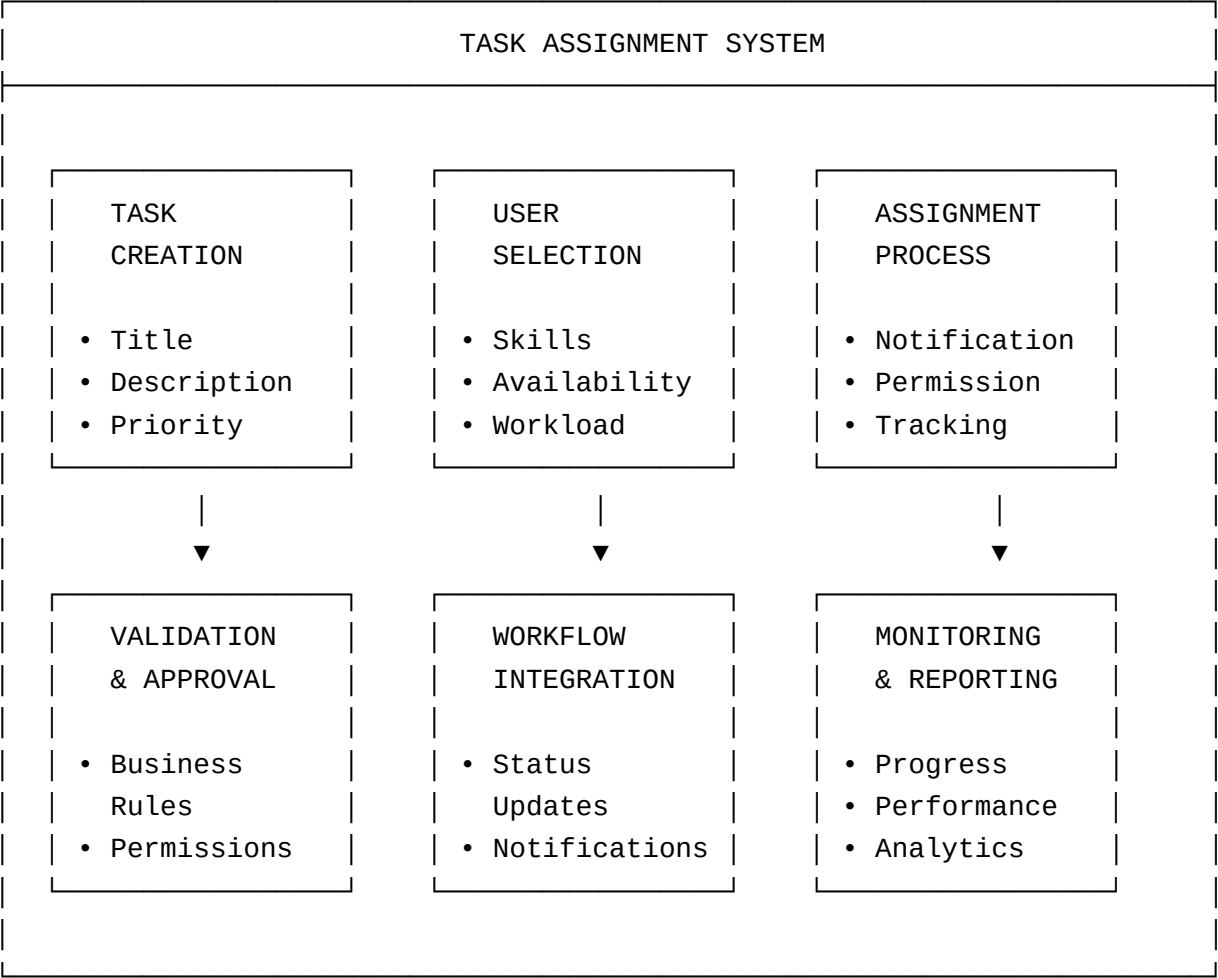


6.0.2 Task Status Flow & Workflow Management

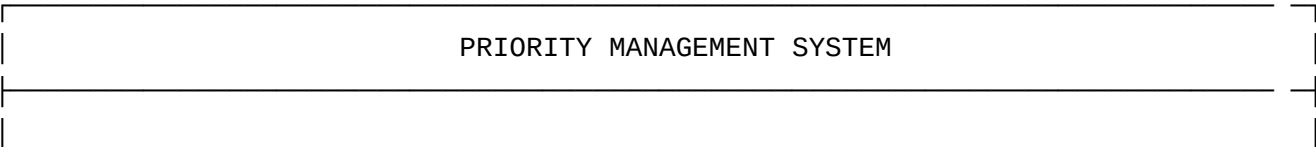


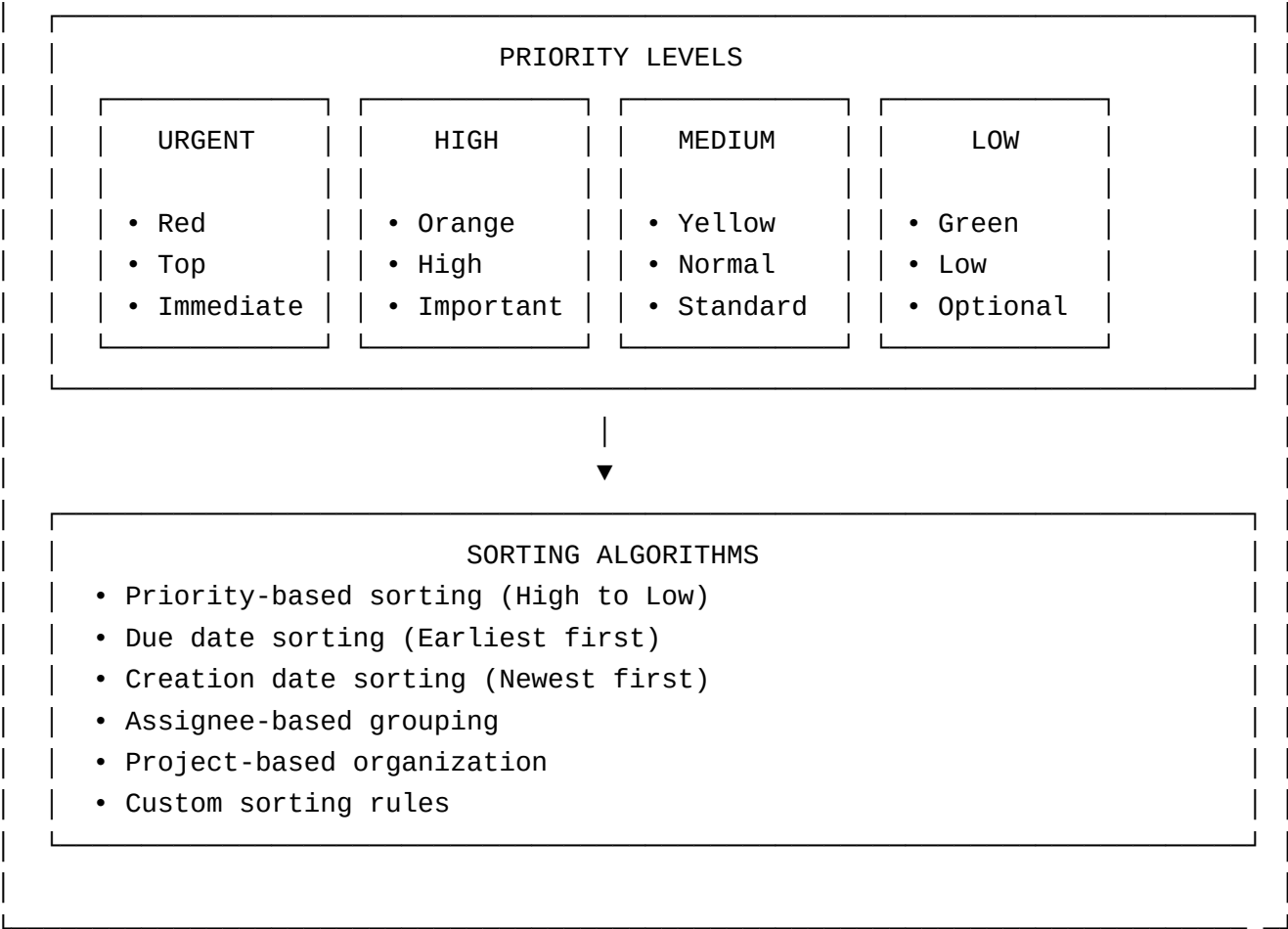


6.0.3 Task Assignment & User Management



6.0.4 Priority Management & Sorting Algorithms

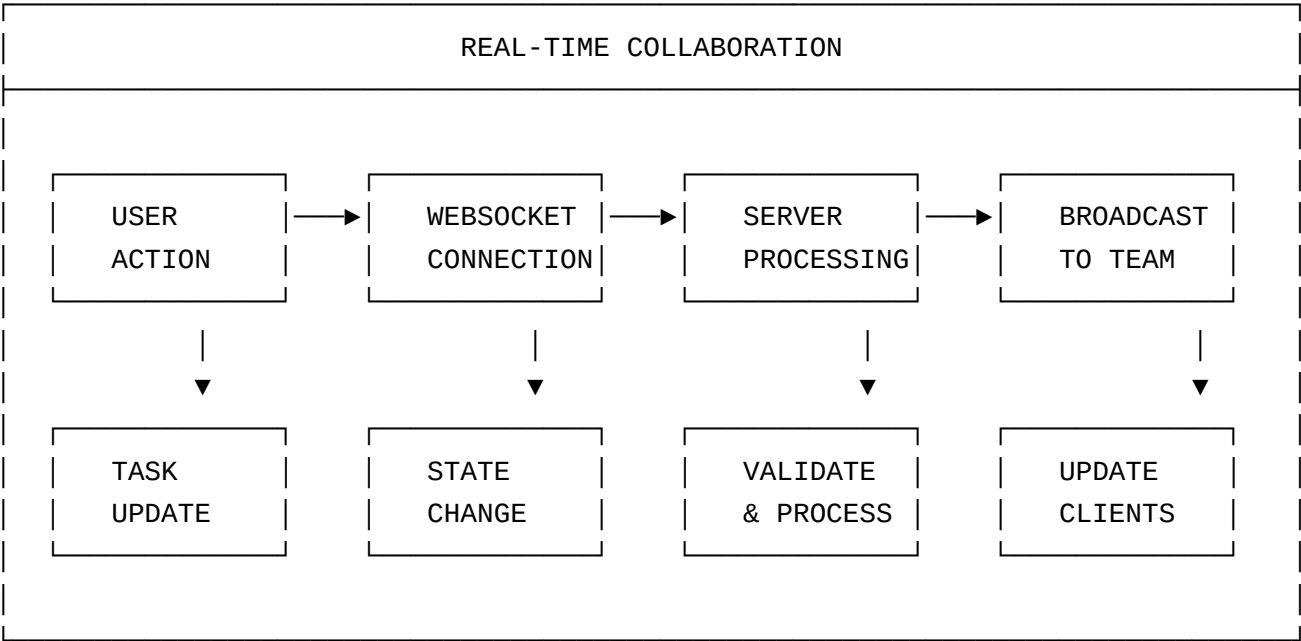




Priority Management Features:

- **Visual Indicators:** Color-coded priority levels for quick identification
- **Smart Sorting:** Automatic sorting based on priority, due date, and creation time
- **Workload Balancing:** Intelligent task distribution across team members
- **Deadline Management:** Automatic notifications for approaching due dates
- **Escalation Rules:** Automatic priority escalation for overdue tasks
- **Capacity Planning:** Workload visualization and resource allocation

6.0.5 Real-time Collaboration & Updates



Real-time Features: - **Live Updates:** Instant synchronization across all team members - **Conflict Resolution:** Automatic conflict detection and resolution - **Offline Support:** Local caching with sync when online - **Activity Feed:** Real-time activity tracking and notifications - **Collaborative Editing:** Multiple users can work simultaneously - **Version History:** Complete audit trail of all changes

```
const useKanbanStore = create(
  devtools(
    (set, get) => ({

const initialColumns = {
  TODO: [],
  IN_PROGRESS: [],
  REVIEW: [],
  DONE: [],
};

    columns: initialColumns,
    isDragging: false,
    draggedTask: null,
    targetColumn: null,
    initializeBoard: (tasks) => {
      const columns = { ...initialColumns };

      tasks.forEach((task) => {
        if (columns[task.status]) {
          columns[task.status].push(task);
        }
      });
      // Sort tasks by priority and creation date
      Object.keys(columns).forEach((status) => {
        columns[status].sort((a, b) => {
          const priorityOrder = { URGENT: 4, HIGH: 3, MEDIUM: 2, LOW: 1 };
          const priorityDiff = priorityOrder[b.priority] - priorityOrder[a.priority];

          if (priorityDiff !== 0) return priorityDiff;

          return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime();
        });
      });
      set({ columns });
    },

    moveTask: async (taskId, fromStatus, toStatus) => {
      try {
        // Update task status via API
        const updatedTask = await taskService.updateTaskStatus(taskId, toStatus);
        // Update local state
        set((state) => {
          const newColumns = { ...state.columns };

```

```

    // Remove from source column
    newColumns[fromStatus] = newColumns[fromStatus].filter(
      (task) => task.id !== taskId
    );
    // Add to target column
    newColumns[toStatus] = [...newColumns[toStatus], updatedTask];
    // Sort target column
    newColumns[toStatus].sort((a, b) => {
      const priorityOrder = { URGENT: 4, HIGH: 3, MEDIUM: 2, LOW: 1 };
      const priorityDiff = priorityOrder[b.priority] - priorityOrder[a.priority];
      if (priorityDiff !== 0) return priorityDiff;

      return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime();
    });

    return { columns: newColumns };
  });
  return updatedTask;
} catch (error) {
  console.error('Failed to move task:', error);
  throw error;
}
},
})
{
  name: 'kanban-store',
}
)
);

```

Chapter 7

File Handling & Storage

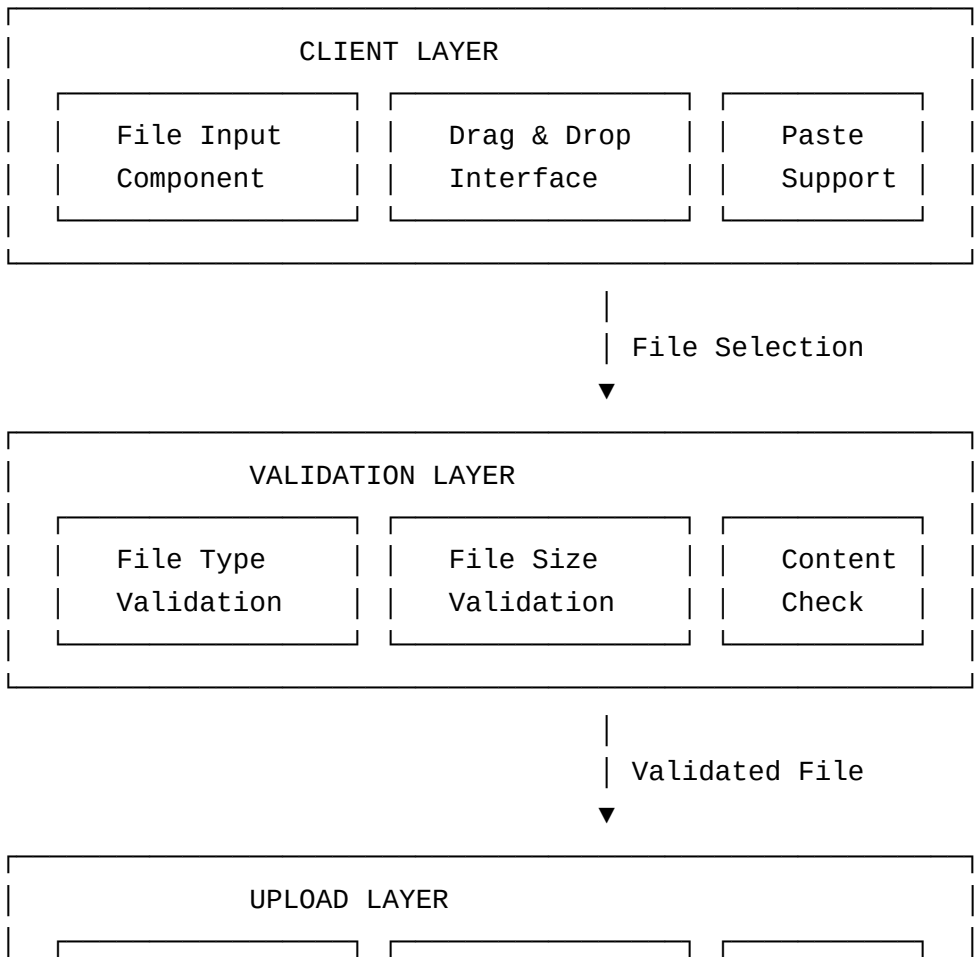
7.1 Overview

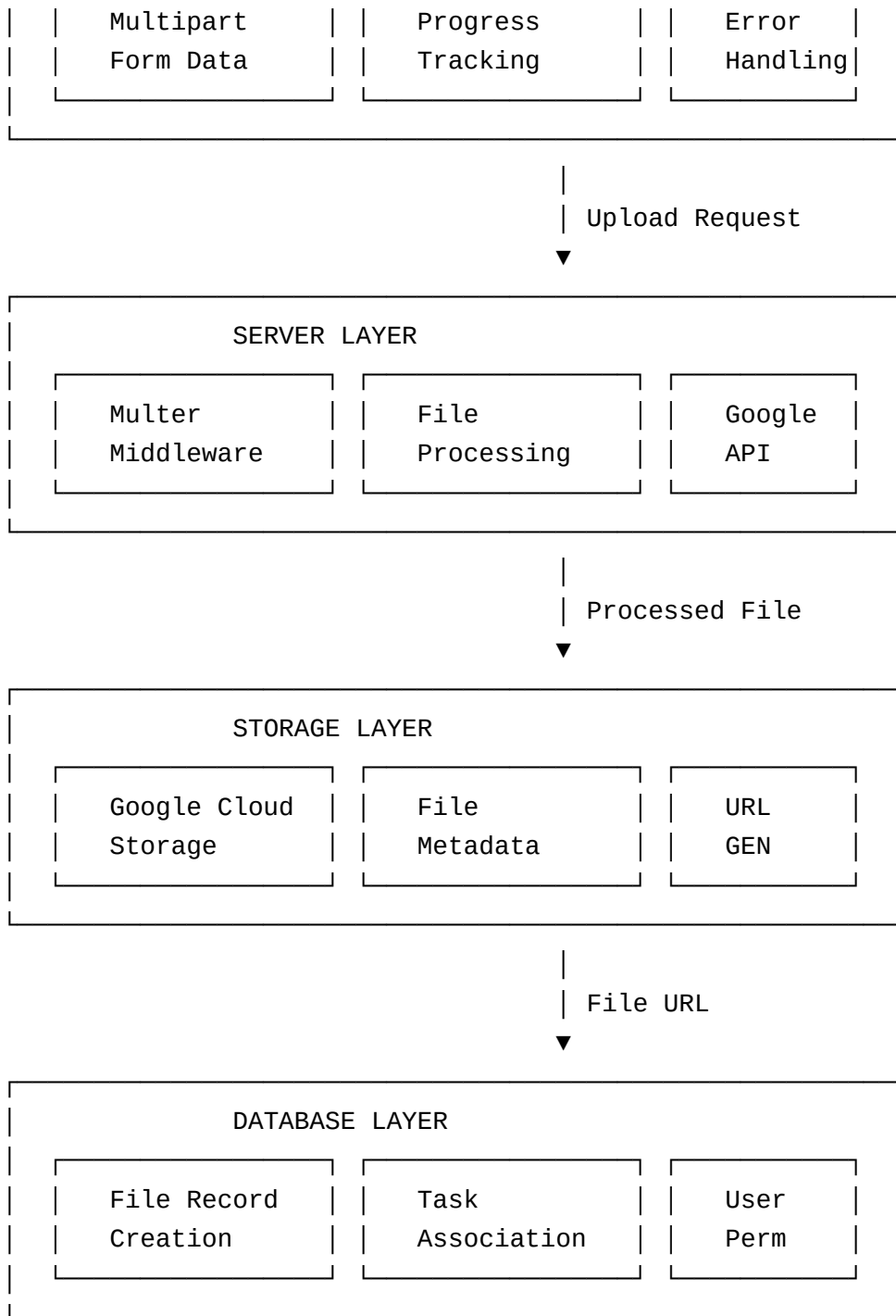
File handling and storage represent critical components of any production application. In Sync, i implement a solution that combines security, performance, and scalability through Google Cloud Storage integration. This chapter explores the technical implementation, security considerations, and real-world deployment strategies.

7.2 File Management Architecture

7.2.1 System Overview

File Upload Flow Architecture:





7.3 Secure File Uploads

7.3.1 File Type Validation

The Problem: File Extension Spoofing

Attackers can upload malicious files by changing file extensions. For example, a .exe file renamed to .jpg could bypass basic validation.

Solution: Multi-Layer Validation

// 1. File Extension Validation

```
const allowedExtensions = ['.jpg', '.jpeg', '.png', '.gif', '.pdf', '.doc', '.docx'];  
const fileExtension = path.extname(file.originalname).toLowerCase();
```

```
// 2. MIME Type Validation
const allowedMimeTypes = [
  'image/jpeg',
  'image/png',
  'image/gif',
  'application/pdf',
  'application/msword',
  'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
];

// 3. Magic Number Validation (File Signature)

const fileSignature = fileBuffer.toString('hex', 0, 4);

const magicNumbers = {
  '89504e47': 'image/png',      // PNG
  'ffd8ffe0': 'image/jpeg',    // JPEG
  '47494638': 'image/gif',     // GIF
  '25504446': 'application/pdf' // PDF
};

const detectedType = magicNumbers[fileSignature];
if (!detectedType || detectedType !== file.mimetype) {
  throw new Error('File signature mismatch');
}
```

Why Magic Numbers Matter

Magic numbers (file signatures) are the first few bytes of a file that identify its type. This prevents attackers from:

- Uploading executable files with image extensions
- Bypassing MIME type validation
- Exploiting browser vulnerabilities with malicious files

7.3.2 File Size Validation

Progressive Size Limits

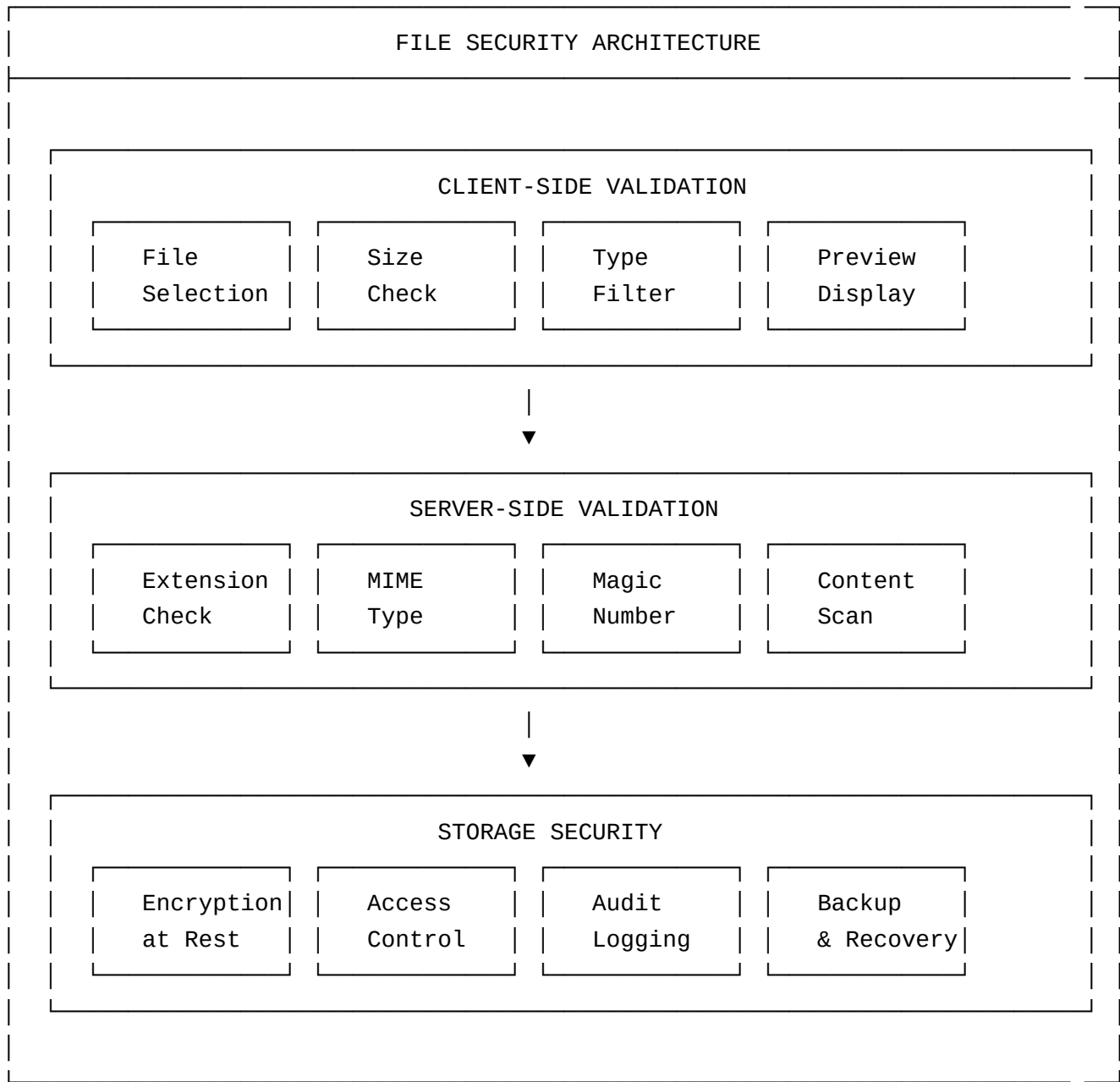
```
const sizeLimits = {
  images: 5 * 1024 * 1024,    // 5MB for images
  documents: 10 * 1024 * 1024, // 10MB for documents
  total: 50 * 1024 * 1024     // 50MB total per user
};
```

Storage Quota Management

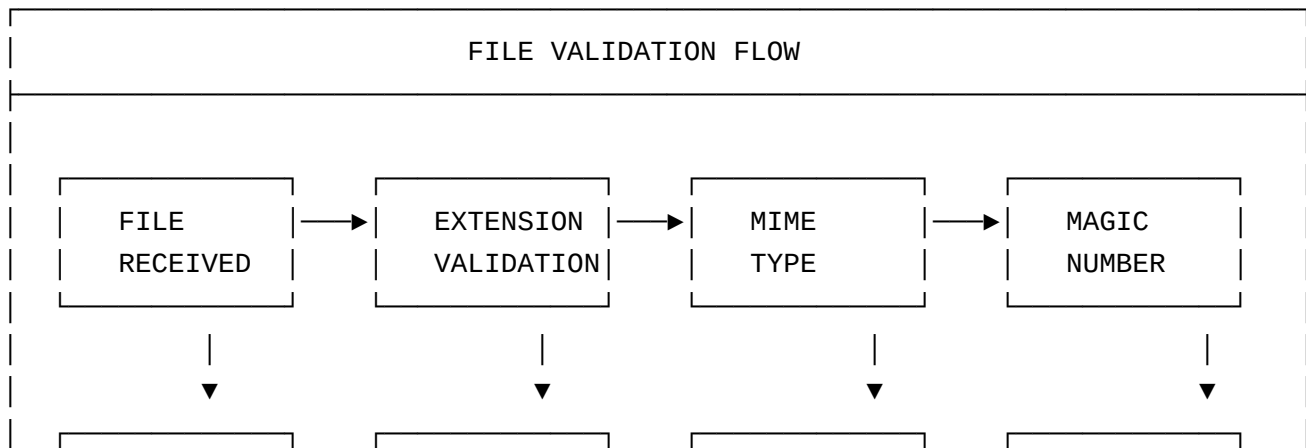
- **Per-User Limits:** Prevent storage abuse
- **File Type Limits:** Optimize for different content types
- **Progressive Scaling:** Increase limits for premium users

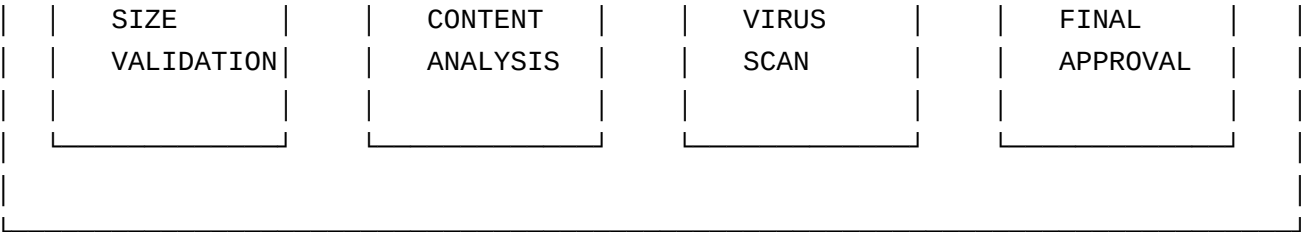
7.4 • Advanced Security & Validation Diagrams

7.4.1 Multi-Layer Security Architecture



7.4.2 File Validation & Sanitization Flow

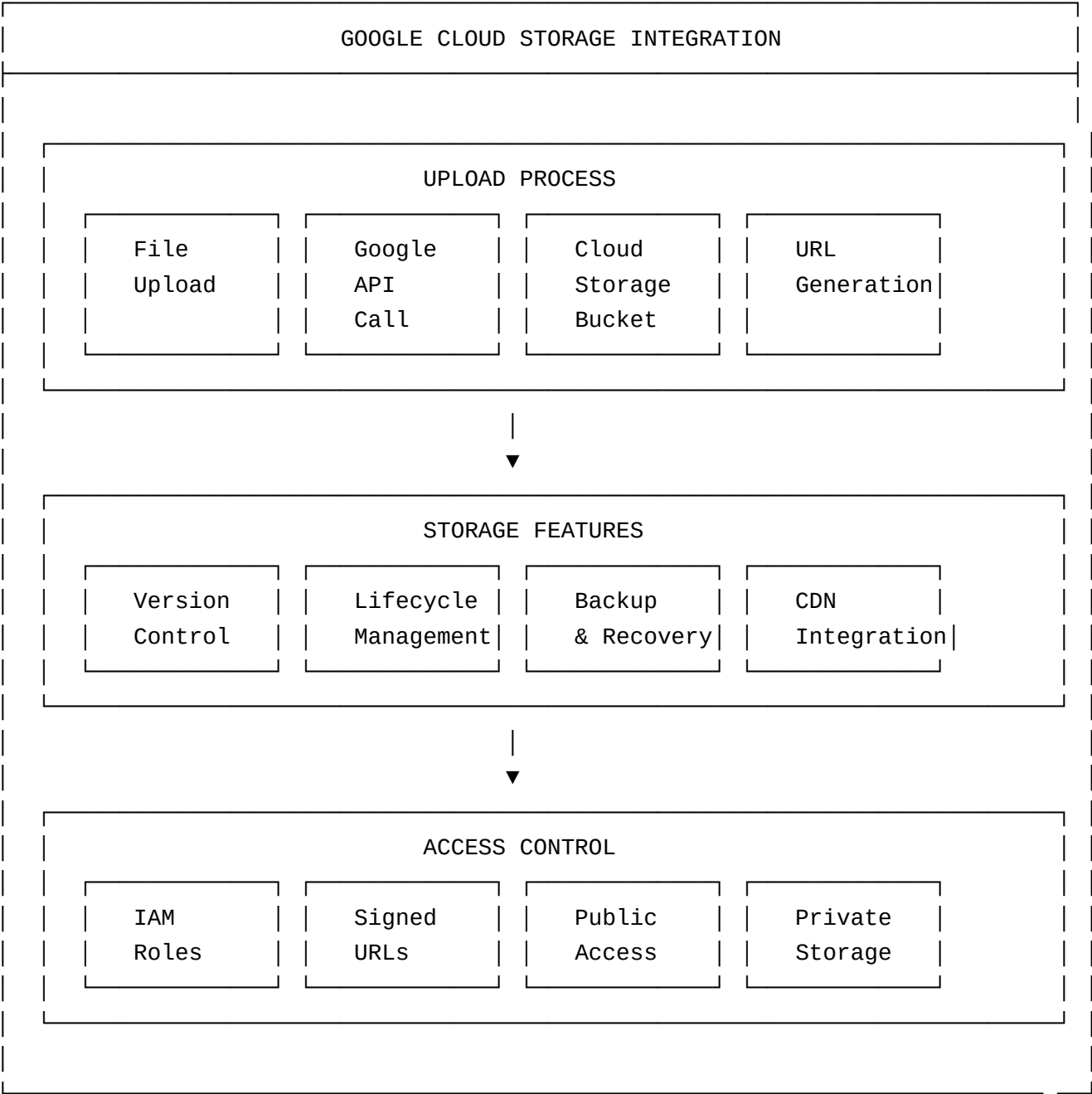




Security Features:

- **Multi-Layer Validation:** File extension, MIME type, and magic number verification
- **Virus Scanning:** Integration with security services for malware detection
- **Content Analysis:** Deep inspection of file contents for threats
- **Access Control:** Role-based permissions and audit logging
- **Encryption:** AES-256 encryption for all stored files
- **Compliance:** GDPR, HIPAA, and SOC 2 compliance features

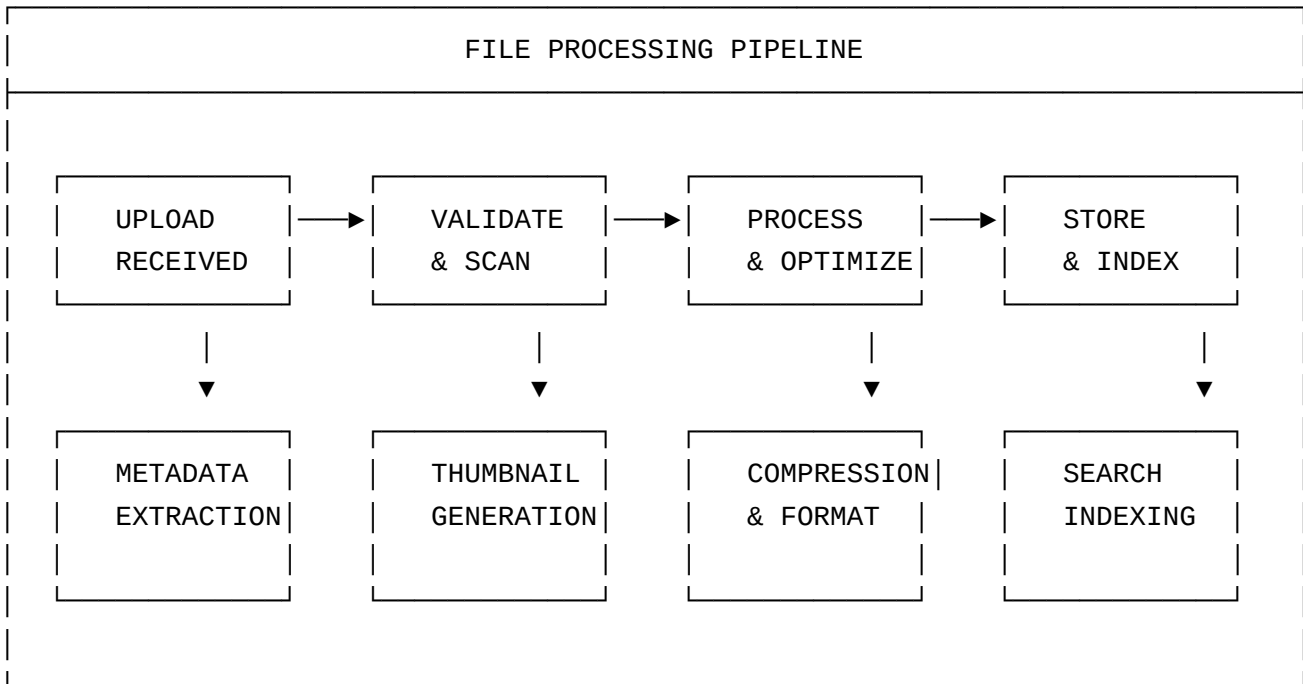
7.4.3 Google Cloud Storage Integration Architecture



Cloud Storage Features:

- **Global CDN:** Fast file delivery worldwide
- **Version Control:** Automatic file versioning and rollback
- **Lifecycle Management:** Automatic archiving and deletion
- **Access Control:** Fine-grained permissions and signed URLs
- **Backup & Recovery:** Automated backup with point-in-time recovery
- **Compliance:** SOC 2, ISO 27001, and GDPR compliance

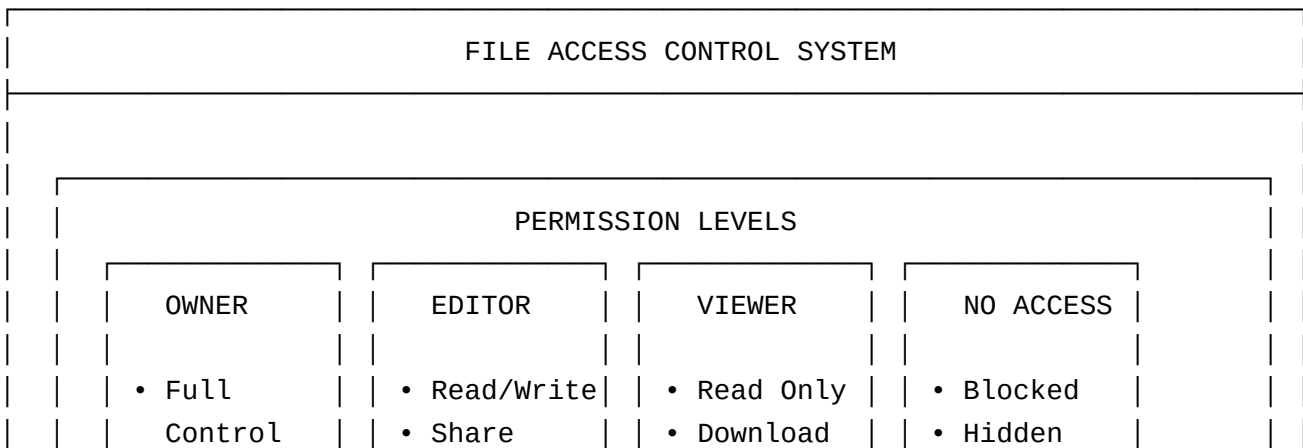
7.4.4 File Processing & Optimization Pipeline

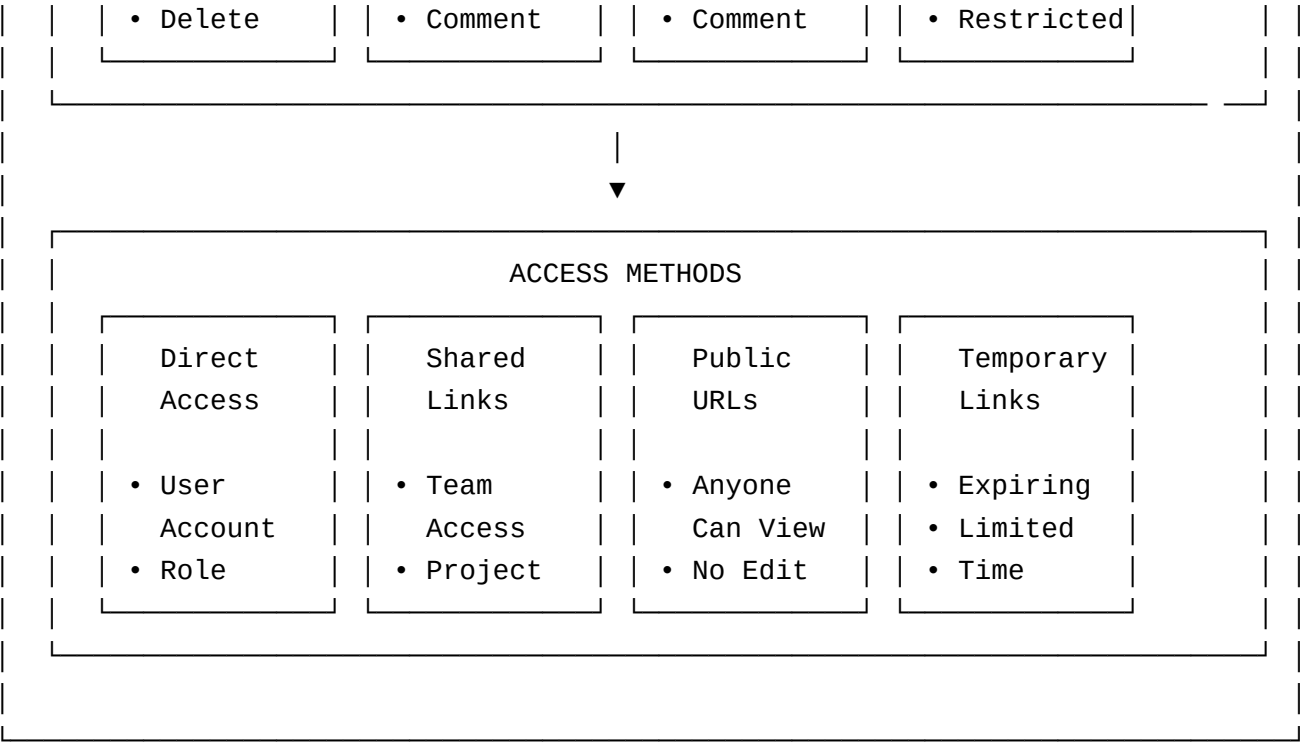


Processing Features:

- **Metadata Extraction:** Automatic extraction of file properties
- **Thumbnail Generation:** Preview images for visual files
- **Compression:** Intelligent compression for optimal storage
- **Format Conversion:** Automatic format optimization
- **Search Indexing:** Full-text search capabilities
- **Performance Optimization:** Lazy loading and caching

7.4.5 File Access Control & Permission System

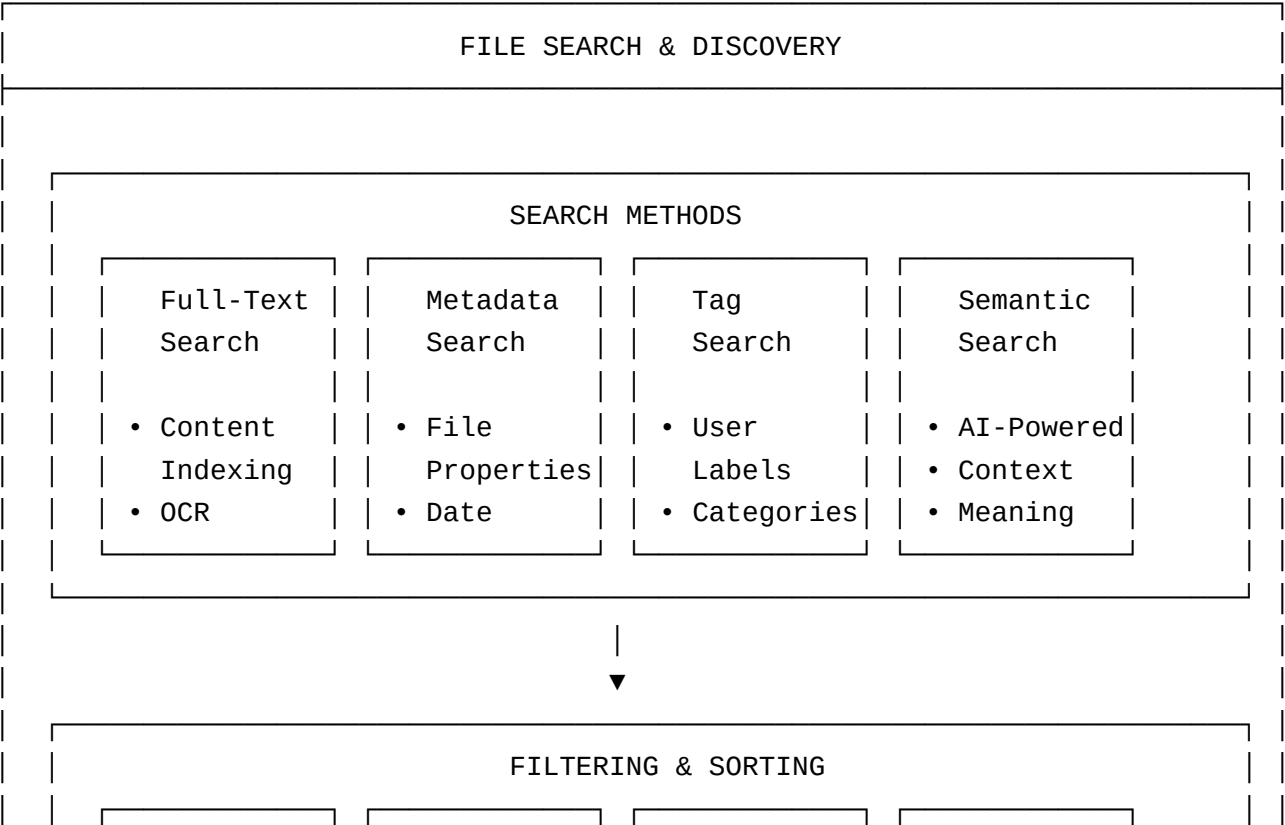




Access Control Features:

- **Role-Based Access:** Granular permissions based on user roles
- **Project-Level Security:** Files inherit project permissions
- **Temporary Access:** Time-limited sharing with expiration
- **Audit Logging:** Complete access history and tracking
- **Watermarking:** Automatic watermarking for sensitive files
- **Download Limits:** Configurable download restrictions

7.4.6 File Search & Discovery System

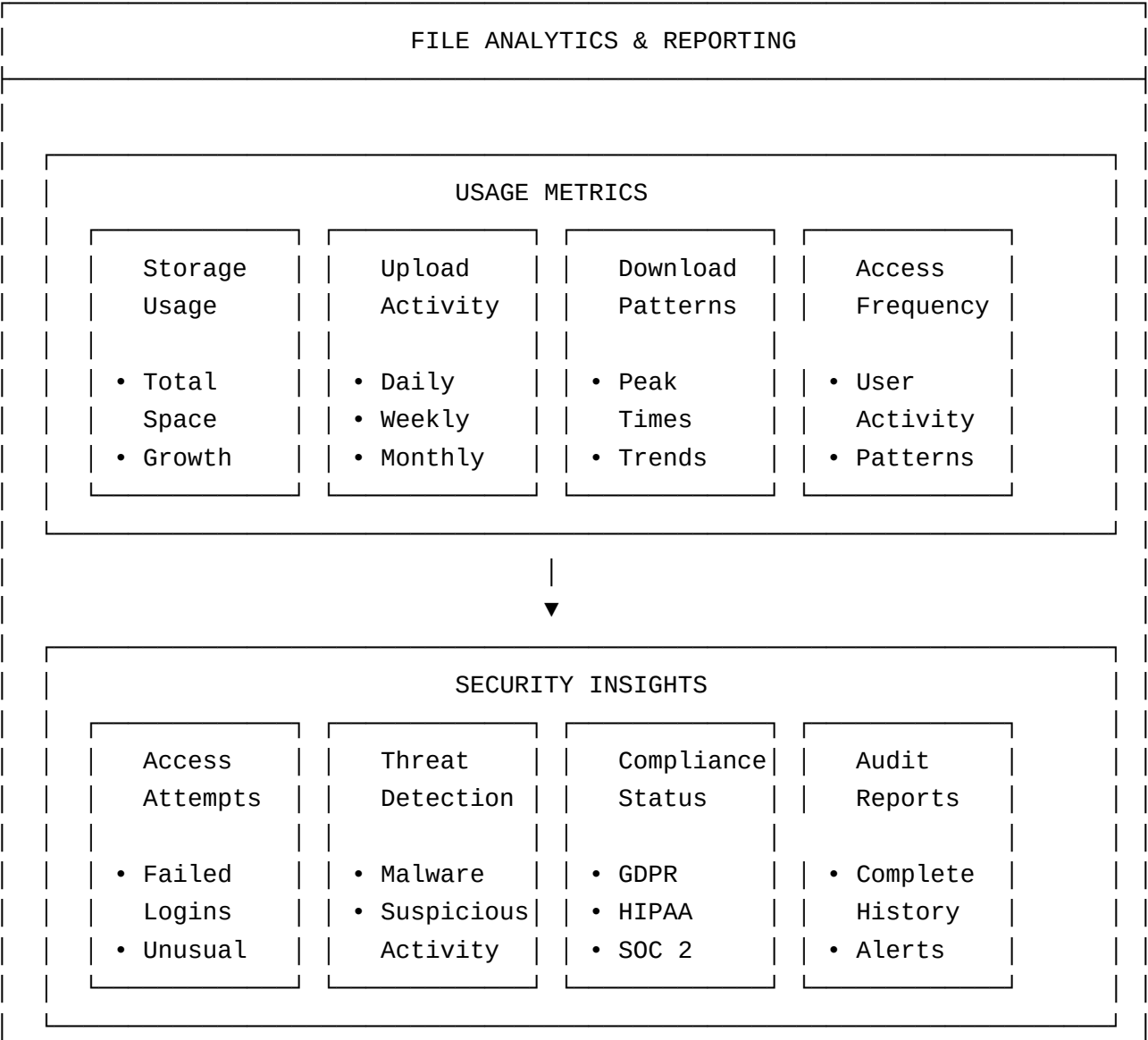


| | | | |
|---|---|---|---|
| File Type | Date Range | Size Range | User Filter |
| <ul style="list-style-type: none">• Extension• MIME• Category | <ul style="list-style-type: none">• Created• Modified• Uploaded | <ul style="list-style-type: none">• Min/Max• Format• Unit | <ul style="list-style-type: none">• Owner• Shared• Team |

Search Features:

- **Full-Text Search:** Search within file contents using OCR and indexing
- **Metadata Search:** Search by file properties, dates, and sizes
- **Tag System:** User-defined labels and categories for easy organization
- **Semantic Search:** AI-powered search understanding context and meaning
- **Advanced Filtering:** Multiple filter combinations for precise results
- **Search History:** Track and save search queries for future use

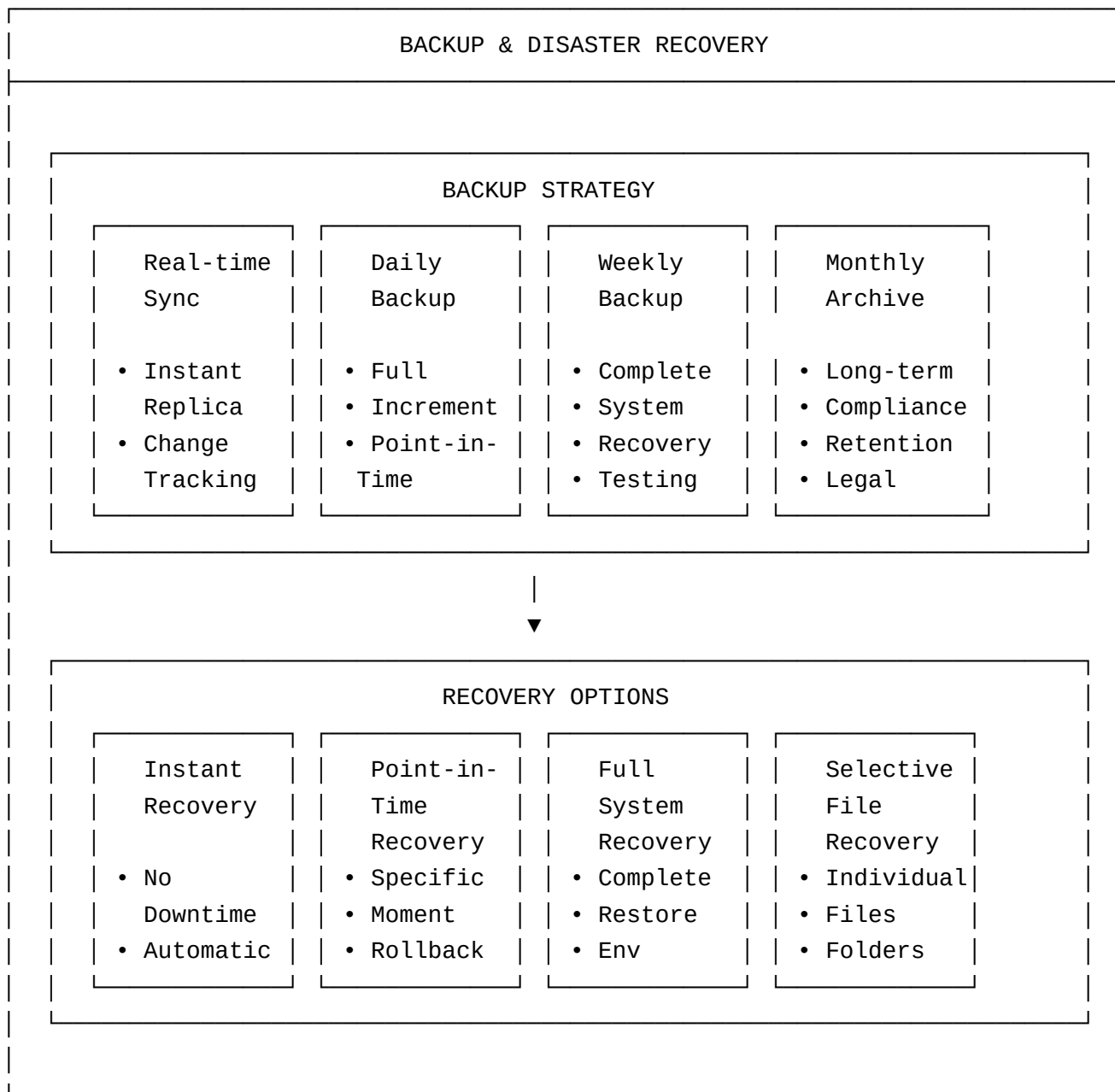
7.4.7 File Analytics & Reporting System



Analytics Features:

- **Storage Insights:** Detailed storage usage and growth patterns
- **User Behavior:** Track file access patterns and user activity
- **Security Monitoring:** Real-time threat detection and security alerts
- **Compliance Reporting:** Automated compliance status and audit reports
- **Performance Metrics:** File upload/download performance tracking
- **Predictive Analytics:** AI-powered storage and usage predictions

7.4.8 File Backup & Disaster Recovery

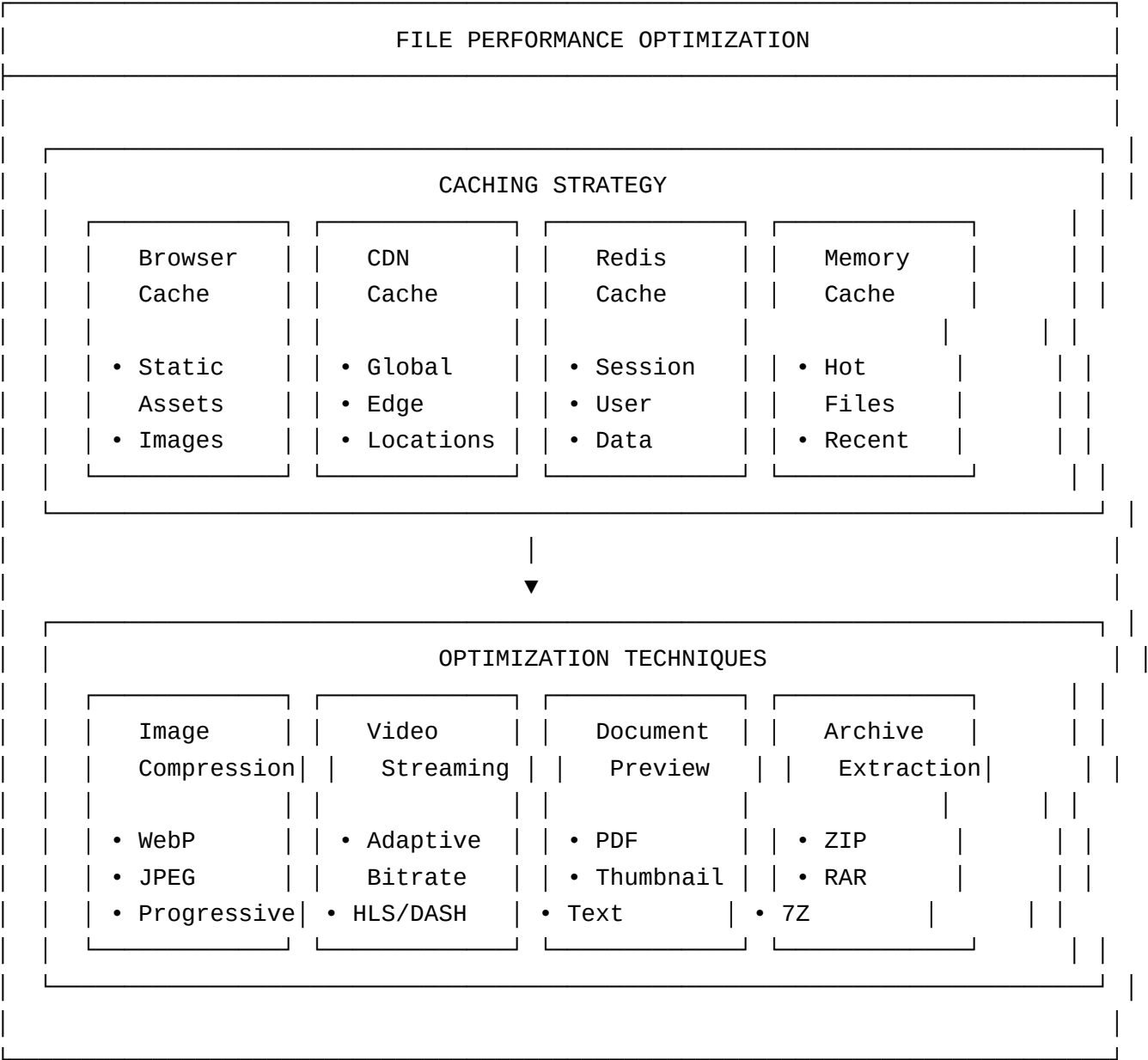


Backup Features:

- **Real-time Replication:** Instant file synchronization across multiple locations
- **Incremental Backups:** Efficient backup strategy saving time and space
- **Point-in-Time Recovery:** Restore to any specific moment in time

- **Geographic Distribution:** Multiple data centers for disaster protection
- **Automated Testing:** Regular backup verification and recovery testing
- **Compliance Ready:** Meets regulatory requirements for data retention

7.4.9 File Performance & Optimization

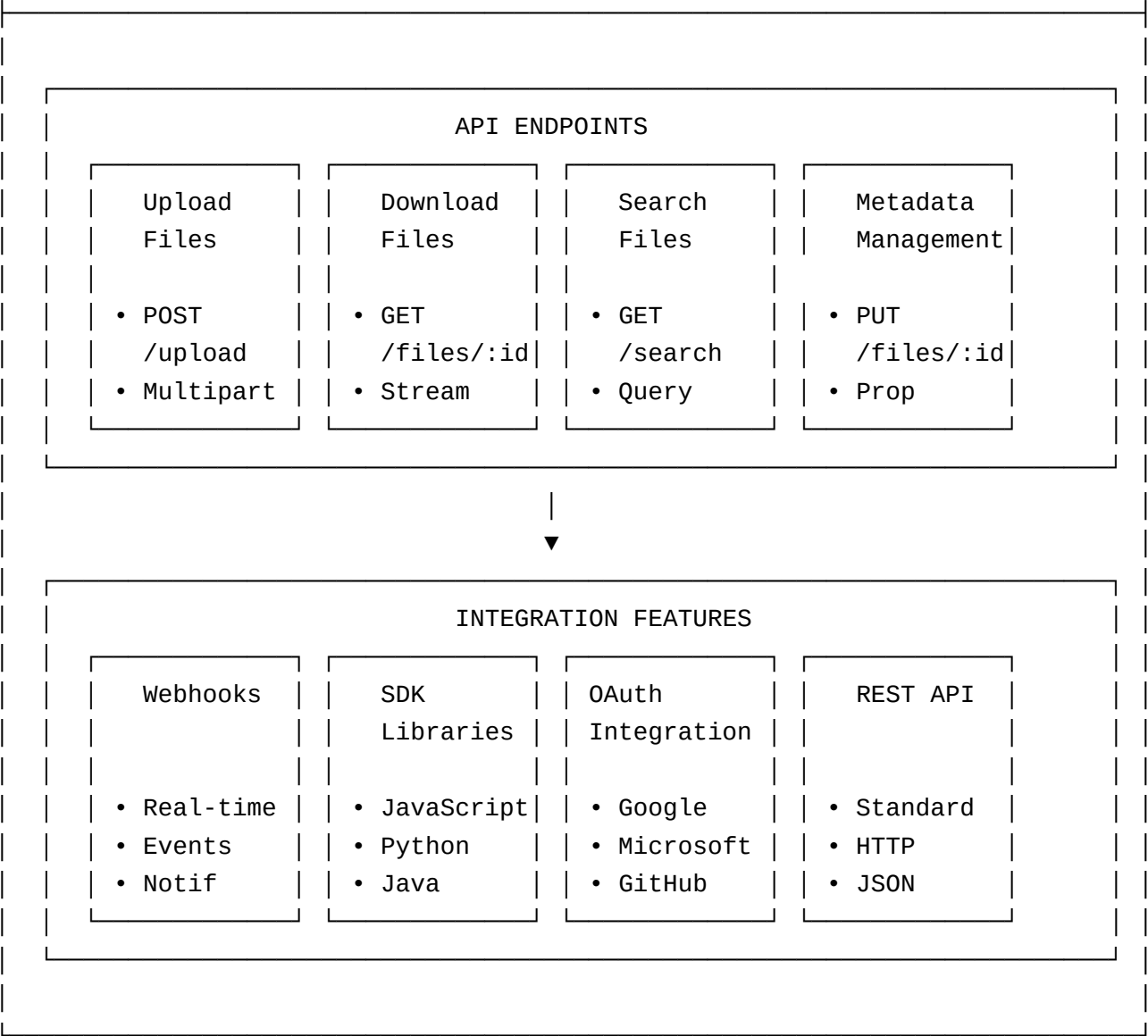


Performance Features:

- **Multi-Level Caching:** Browser, CDN, Redis, and memory caching
- **Image Optimization:** Automatic compression and format conversion
- **Video Streaming:** Adaptive bitrate streaming for optimal performance
- **Document Preview:** Instant preview generation for common formats
- **Lazy Loading:** Load files only when needed
- **Progressive Enhancement:** Optimize for different network conditions

7.4.10 File Integration & API System

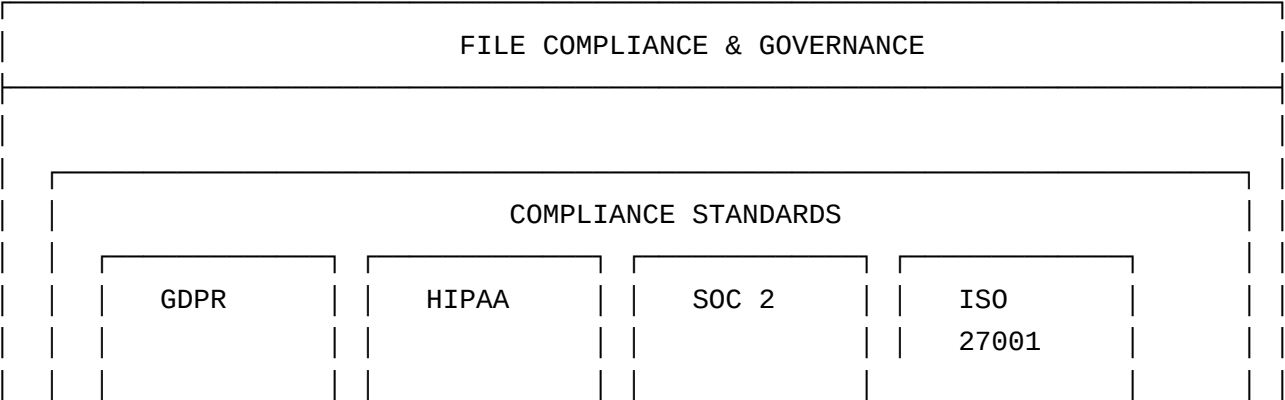
| FILE INTEGRATION & API |
|------------------------|
|------------------------|

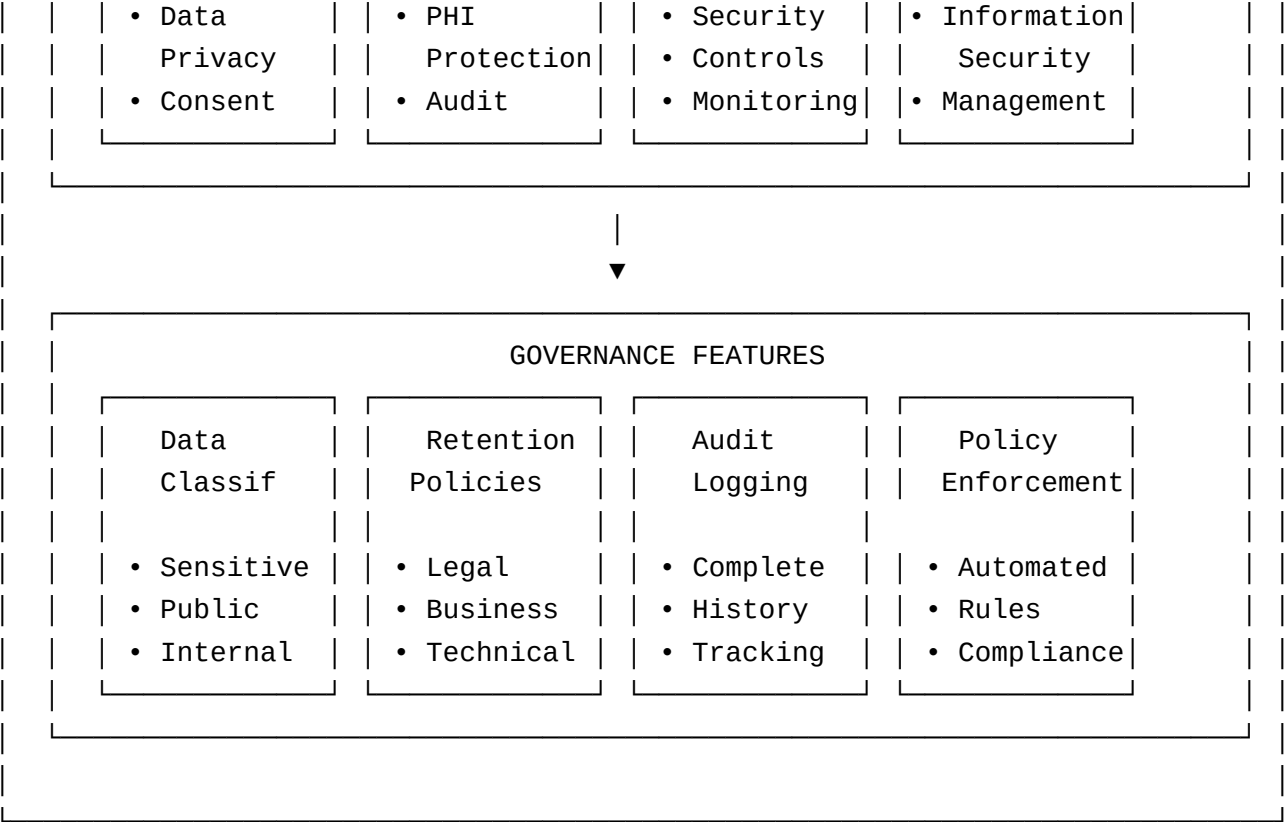


Integration Features:

- **RESTful API:** Standard HTTP methods for all file operations
- **Webhook Support:** Real-time notifications for file events
- **SDK Libraries:** Client libraries for multiple programming languages
- **OAuth Integration:** Secure authentication with popular services
- **Event Streaming:** Real-time file activity updates
- **Batch Operations:** Efficient bulk file processing

7.4.11 File Compliance & Governance

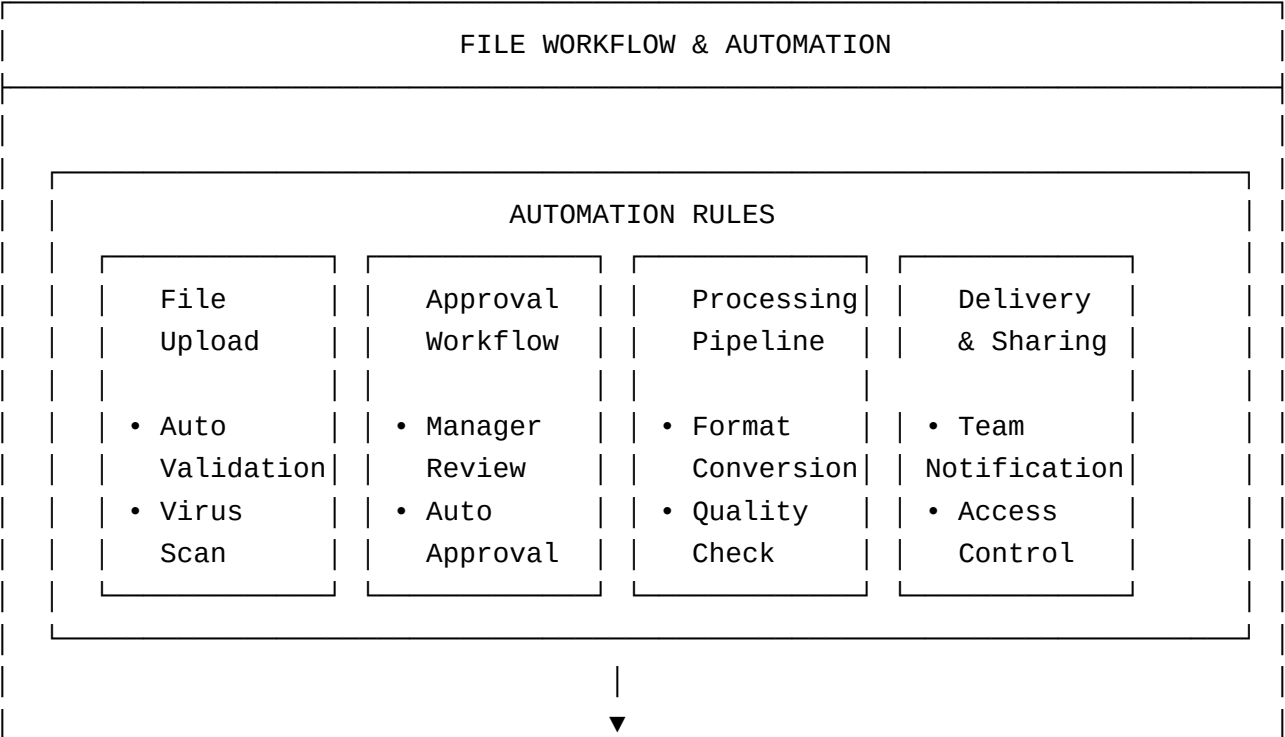


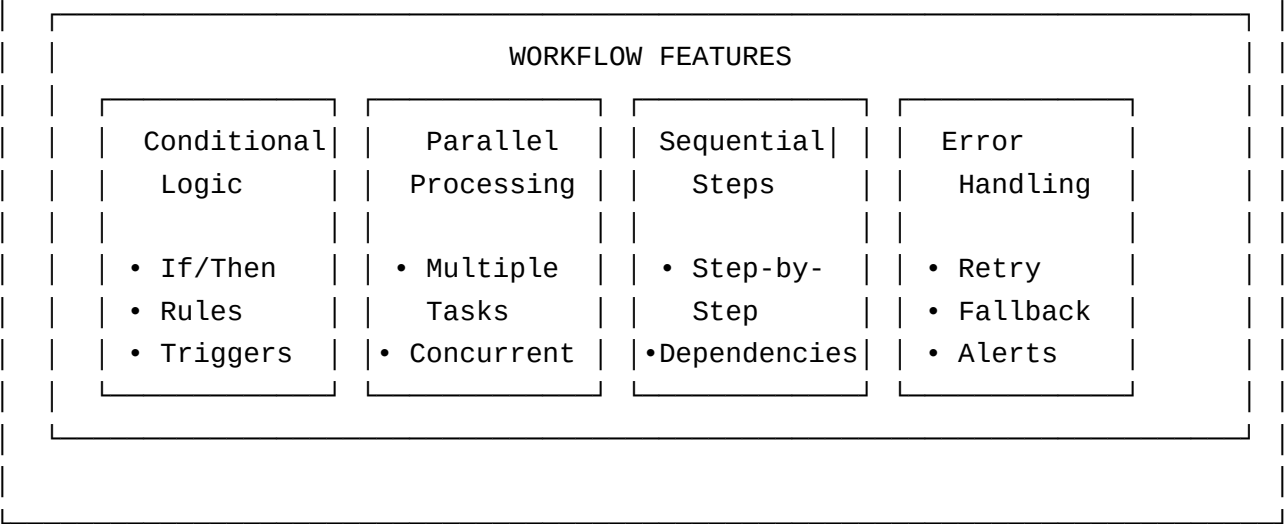


Compliance Features:

- **Data Classification:** Automatic classification of sensitive information
- **Retention Policies:** Configurable data retention and deletion rules
- **Audit Logging:** Complete audit trail for compliance reporting
- **Policy Enforcement:** Automated policy enforcement and monitoring
- **Privacy Controls:** GDPR-compliant data handling and consent management
- **Security Standards:** SOC 2, ISO 27001, and HIPAA compliance

7.4.12 File Workflow & Automation

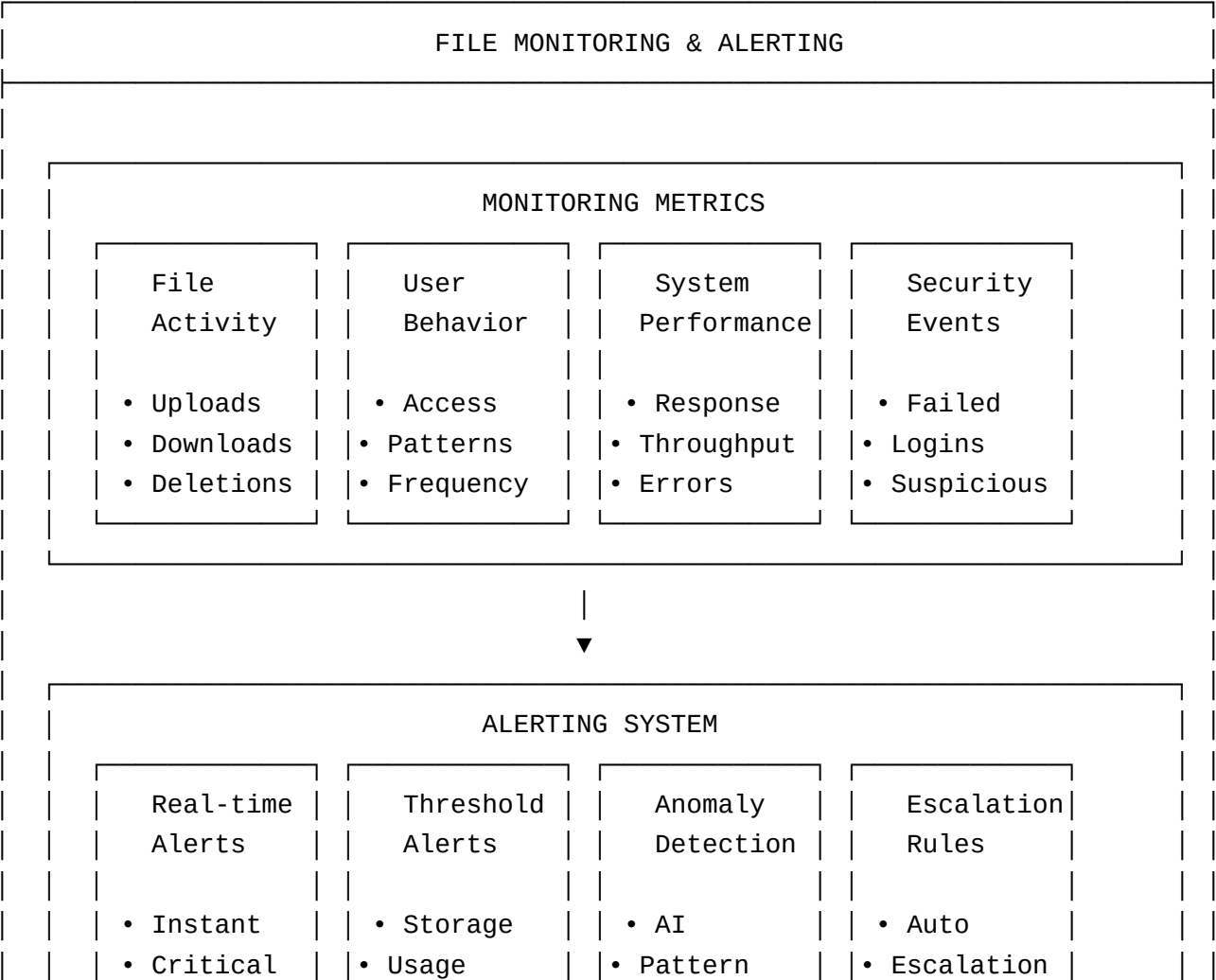




Workflow Features:

- **Conditional Logic:** Smart rules based on file properties and content
- **Parallel Processing:** Multiple tasks executed simultaneously
- **Sequential Steps:** Ordered workflow execution with dependencies
- **Error Handling:** Automatic retry and fallback mechanisms
- **Approval Workflows:** Configurable approval processes for sensitive files
- **Integration Hooks:** Connect with external systems and services

7.4.13 File Monitoring & Alerting

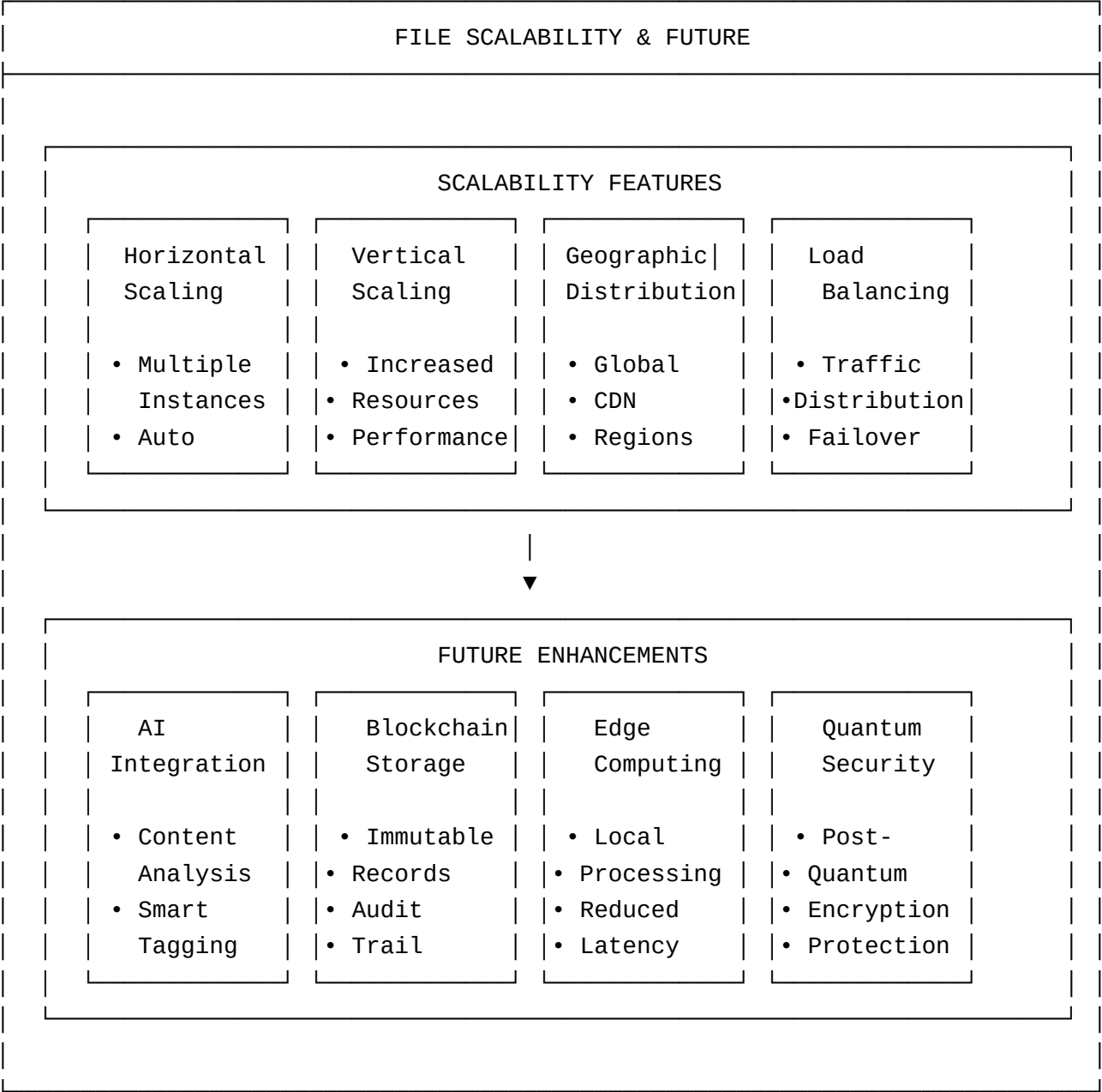


| | | | | | |
|--|-----------------------|---------------------|--------------------------|-------------------------|--|
| | <div>• Security</div> | <div>• Limits</div> | <div>• Recognition</div> | <div>• Management</div> | |
|--|-----------------------|---------------------|--------------------------|-------------------------|--|

Monitoring Features:

- **Real-time Alerts:** Instant notifications for critical events
- **Threshold Monitoring:** Configurable limits and alerts
- **Anomaly Detection:** AI-powered pattern recognition
- **Performance Metrics:** Comprehensive system performance tracking
- **Security Monitoring:** Real-time security event detection
- **Escalation Rules:** Automatic escalation for critical issues

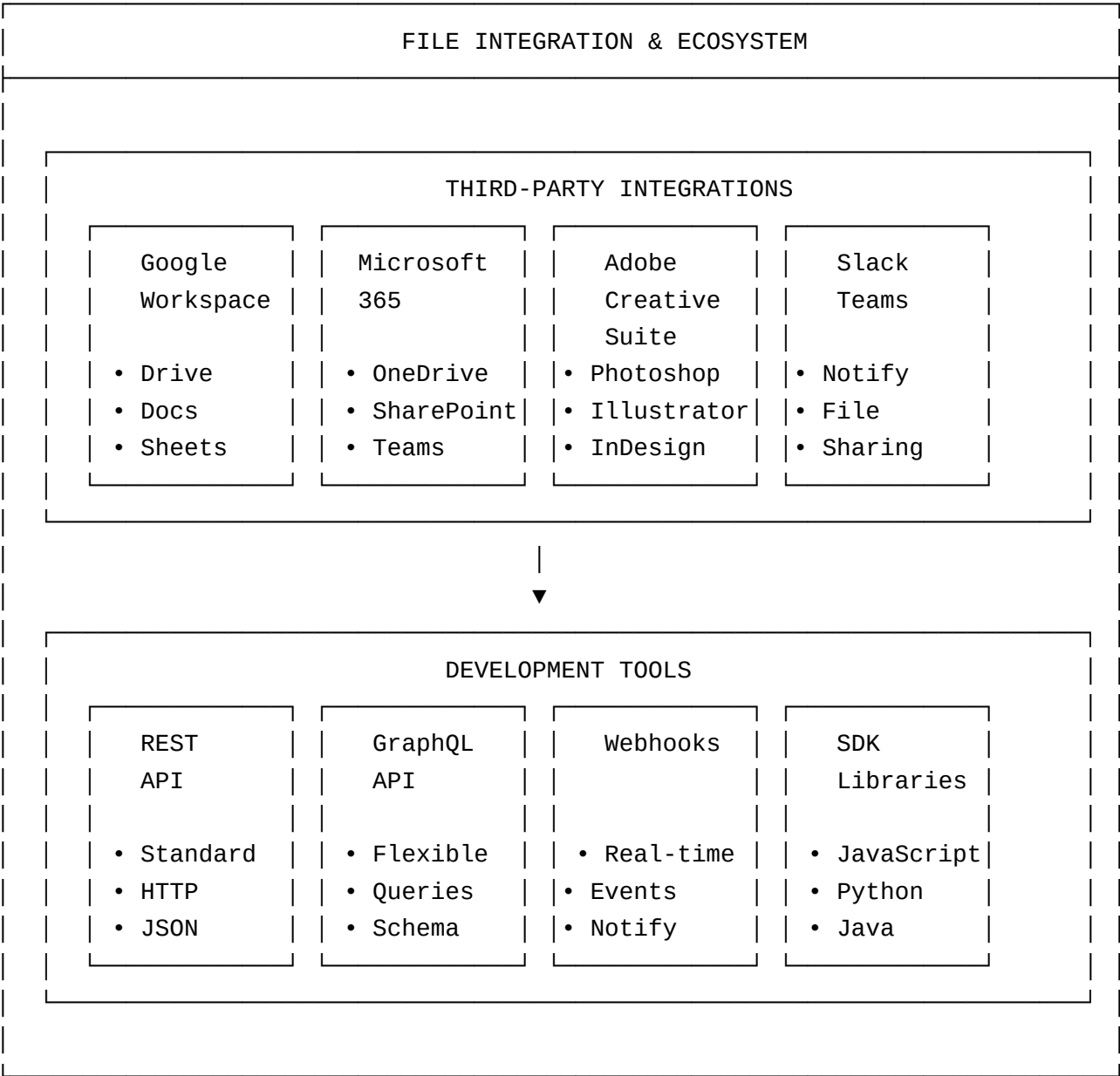
7.4.14 File Scalability & Future Architecture



Scalability Features:

- **Horizontal Scaling:** Multiple server instances for load distribution
- **Vertical Scaling:** Increased resources for individual servers
- **Geographic Distribution:** Global CDN and edge locations
- **Load Balancing:** Intelligent traffic distribution and failover
- **Auto-scaling:** Automatic resource allocation based on demand
- **Future-Ready:** Architecture designed for emerging technologies

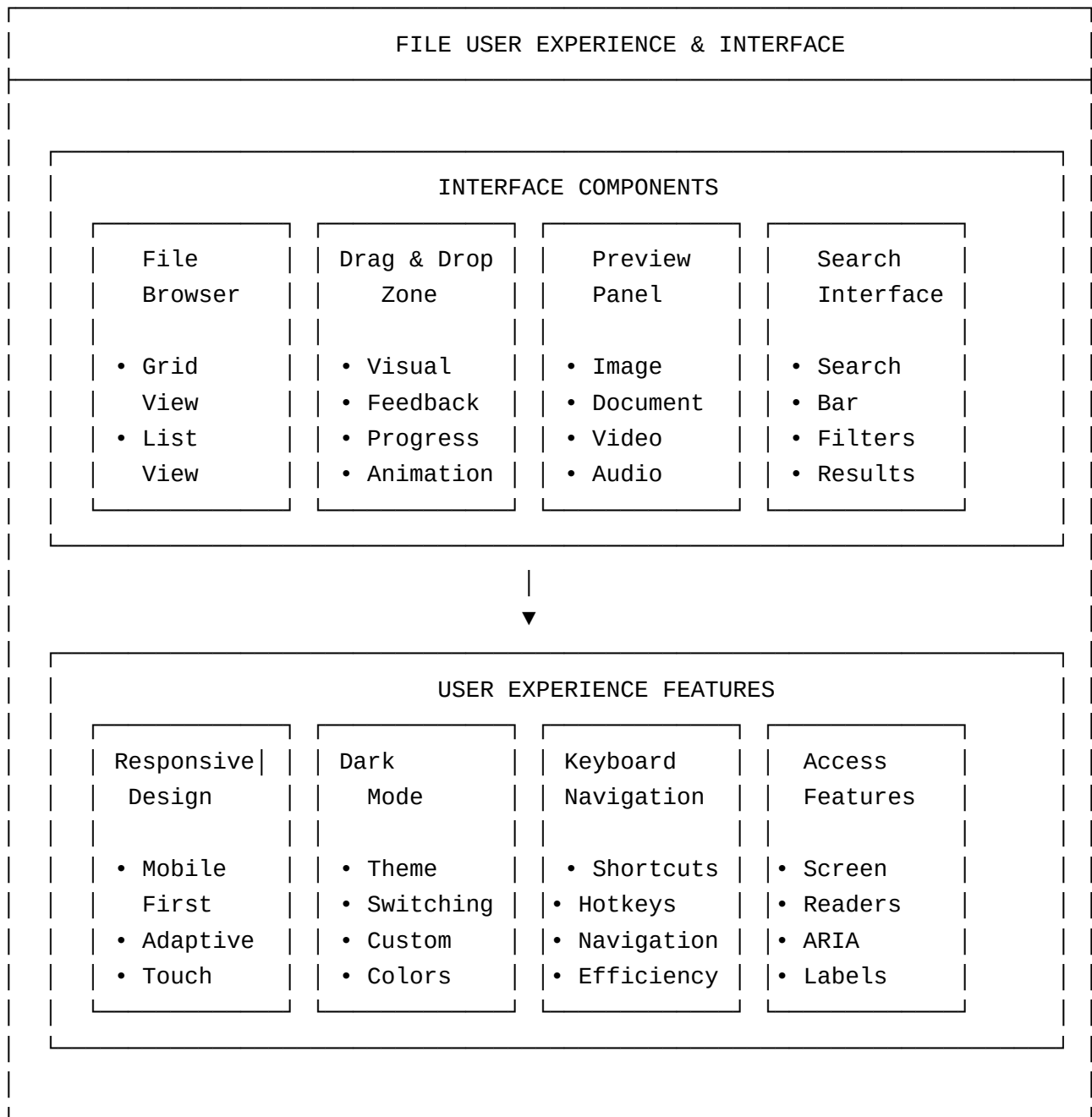
7.4.15 File Integration & Ecosystem



Integration Features:

- **Google Workspace:** Seamless integration with Google Drive and Docs
- **Microsoft 365:** Full compatibility with OneDrive and SharePoint
- **Adobe Creative Suite:** Direct integration with design tools
- **Communication Platforms:** Slack, Teams, and Discord integration
- **REST API:** Standard HTTP API for custom integrations
- **GraphQL API:** Flexible query language for complex data needs

7.4.16 File User Experience & Interface



User Experience Features:

- **Responsive Design:** Optimized for all devices and screen sizes
- **Dark Mode:** Comfortable viewing in low-light environments
- **Keyboard Navigation:** Full keyboard support for power users
- **Accessibility:** WCAG 2.1 AA compliance with screen reader support
- **Touch Interface:** Optimized for mobile and tablet devices
- **Customization:** User-configurable themes and layouts

7.4.17 Content Security

Virus Scanning Integration

```
// Integration with ClamAV or similar
const scanFile = async (fileBuffer) => {
  try {
```

```

    const result = await virusScanner.scan(fileBuffer);
    if (result.isInfected) {
        throw new Error('File contains malware');
    }
    return true;
} catch (error) {
    console.error('Virus scan failed:', error);
    // In production, reject files if scan fails
    throw new Error('Security scan failed');
}
};

```

Image Processing and Sanitization

```

const processImage = async (fileBuffer, mimeType) => {
    const sharp = require('sharp');

    try {
        // Remove EXIF data (potential privacy risk)
        const processedImage = await sharp(fileBuffer)
            .removeExif()
            .resize(1920, 1080, { fit: 'inside' }) // Max dimensions
            .jpeg({ quality: 85 }) // Optimize quality
            .toBuffer();

        return processedImage;
    } catch (error) {
        throw new Error('Image processing failed');
    }
};

```

Chapter 8

Deployment & Scaling

8.1 Overview

Deployment and scaling are critical phases in the application lifecycle. This chapter covers production deployment strategies, containerization with Docker, and both vertical and horizontal scaling approaches. We'll explore how to take Sync from development to production and prepare it for enterprise-scale usage.

8.2 Containerization with Docker

8.2.1 Development Dockerfile

```
# Dockerfile.dev
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "src/server.js"]
```

8.2.2 Docker Compose for Production

```
# docker-compose.prod.yml
version: '3.8'

services:
  app:
    build:
      context: .
```

```

    dockerfile: Dockerfile.prod
ports:
  - "3000:3000"
environment:
  - NODE_ENV=production
  - DATABASE_URL=${DATABASE_URL}
  - JWT_SECRET=${JWT_SECRET}
  - GOOGLE_CLOUD_PROJECT_ID=${GOOGLE_CLOUD_PROJECT_ID}
depends_on:
  - db
  - redis
restart: unless-stopped
deploy:
  replicas: 3
  resources:
    limits:
      cpus: '1.0'
      memory: 1G
    reservations:
      cpus: '0.5'
      memory: 512M

db:
  image: postgres:15-alpine
  environment:
    - POSTGRES_DB=${POSTGRES_DB}
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
  volumes:
    - postgres_prod_data:/var/lib/postgresql/data
  restart: unless-stopped
  deploy:
    resources:
      limits:
        cpus: '2.0'
        memory: 4G

redis:
  image: redis:7-alpine
  command: redis-server --appendonly yes --requirepass ${REDIS_PASSWORD}
  volumes:
    - redis_prod_data:/data
  restart: unless-stopped
  deploy:
    resources:
      limits:
        cpus: '0.5'
        memory: 512M

```

```

nginx:
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    - ./nginx/ssl:/etc/nginx/ssl
  depends_on:
    - app
  restart: unless-stopped

```

```

volumes:
  postgres_prod_data:
  redis_prod_data:

```

8.3 Scaling Strategies

8.3.1 Vertical Scaling

Server Resource Optimization

```

// server.js - Optimized for vertical scaling
const cluster = require('cluster');
const os = require('os');

if (cluster.isMaster) {
  const numCPUs = os.cpus().length;

  console.log(`Master ${process.pid} is running`);
  console.log(`Forking for ${numCPUs} CPUs`);

  // Fork workers
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`Worker ${worker.process.pid} died`);
    // Replace the dead worker
    cluster.fork();
  });
} else {
  // Worker process
  require('./src/app');
  console.log(`Worker ${process.pid} started`);
}

```

8.3.2 Horizontal Scaling

Load Balancer Configuration

```
// loadBalancer.js
const http = require('http');
const httpProxy = require('http-proxy');

const proxy = httpProxy.createProxyServer({});

const servers = [
  'http://app1:3000',
  'http://app2:3000',
  'http://app3:3000'
];

let currentServer = 0;

const server = http.createServer((req, res) => {
  // Round-robin load balancing
  const target = servers[currentServer];
  currentServer = (currentServer + 1) % servers.length;

  proxy.web(req, res, { target });
});

server.listen(8080);
```

Chapter 9

Bibliography

9.1 Primary Technologies & Frameworks

9.1.1 Backend Technologies

Node.js - Node.js Foundation. (2024). Node.js Documentation. <https://nodejs.org/docs/>

Express.js - StrongLoop, IBM, and other contributors. (2024). Express.js - Fast, unopinionated, minimalist web framework for Node.js. <https://expressjs.com/>

PostgreSQL - PostgreSQL Global Development Group. (2024). PostgreSQL: The World's Most Advanced Open Source Relational Database. <https://www.postgresql.org/>

Prisma ORM - Prisma Documentation. (2024). Prisma Client, Prisma Migrate, and Prisma Studio. <https://www.prisma.io/docs/>

9.1.2 Frontend Technologies

React - Facebook, Inc. (2024). React – A JavaScript library for building user interfaces. <https://reactjs.org/>

Vite - Vite Documentation. (2024). Build tool that aims to provide a faster and leaner development experience. <https://vitejs.dev/guide/>

JavaScript - ECMAScript. (2024). ECMAScript Language Specification. <https://tc39.es/ecma262/>

TailwindCSS - Tailwind CSS Documentation. (2024). Rapidly build modern websites without ever leaving your HTML. <https://tailwindcss.com/docs>

9.1.3 State Management

Zustand - Zustand Documentation. (2024). Simple state management for React. <https://github.com/pmndrs/zustand#r>

9.1.4 Form Management

Formik - Formik Documentation. (2024). Form state management and validation for React. <https://formik.org/docs/ov>

Yup - Yup Documentation. (2024). JavaScript object schema validator and object parser. <https://github.com/jquense/y>

9.1.5 Authentication & Security

JSON Web Tokens (JWT) - Auth0. (2024). JWT.io - JSON Web Token Debugger. <https://jwt.io/>

Bcrypt - Bcrypt Documentation. (2024). bcrypt - A library to help you hash passwords. <https://github.com/dcodeIO/bcrypt.js>

Helmet.js - Helmet Documentation. (2024). Security middleware for Express.js. <https://helmetjs.github.io/docs/>

9.1.6 File Handling & Storage

Google Cloud Storage - Google Cloud Documentation. (2024). Store and serve large amounts of data. <https://cloud.google.com/storage/docs>

9.1.7 Development Tools

Git - Torvalds, L., & Hamano, J. (2024). Git - Distributed version control system. <https://git-scm.com/>

9.1.8 Deployment & DevOps

Docker - Docker Documentation. (2024). Container platform for developers and DevOps. <https://docs.docker.com/>

9.1.9 Security Standards & Best Practices

CORS - MDN Web Docs. (2024). Cross-Origin Resource Sharing (CORS). <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

CSRF Protection - MDN Web Docs. (2024). Cross-Site Request Forgery (CSRF). <https://developer.mozilla.org/en-US/docs/Glossary/CSRF>

9.1.10 Database Design & Migration

Prisma Migrations - Prisma Documentation. (2024). Database schema migrations. <https://www.prisma.io/docs/concepts/migrate>