System Architecture Documentation

File Management System - Frontend Architecture

Table of Contents

Overview

The File Management System is a modern, full-stack web application built with Next.js 15, designed to provide comprehensive file storage, management, and sharing capabilities. The frontend architecture follows a modular, component-based approach with strict type safety, modern UI/UX patterns, and real-time notification support.

Key Features

- **Multi-file type support**: Images, Videos, Audio, Documents, and Other files
- **Real-time notifications**: Firebase Cloud Messaging (FCM) integration
- **Secure authentication**: JWT-based with 2FA support
- **Storage analytics**: Visual dashboards with usage metrics
- **Role-based access control**: Multiple user roles with different permissions
- **Responsive design**: Mobile-first approach with Tailwind CSS

Technology Stack

Core Framework

- **Next.js 15.3.4**: React framework with App Router
- **React 19.0.0**: UI library with latest features
- **TypeScript 5**: Type-safe development

UI/UX Layer

- **Tailwind CSS 4**: Utility-first CSS framework
- **shadcn/ui**: High-quality, accessible component library

- **Lucide React**: Icon system
- **next-themes**: Dark/light mode support
- **Recharts**: Data visualization for analytics

State Management & Data Fetching

- **React Query (TanStack Query)**: Server state management
- **nuqs**: URL state management
- **Context API**: Global state for notifications and theme

Form Management & Validation

- **React Hook Form 7.62.0**: Performant form handling
- **Zod 4.1.5**: Schema validation and type inference
- **@hookform/resolvers**: Integration bridge

Backend Communication

- **Axios 1.11.0**: HTTP client with interceptors
- **JWT Decode**: Token parsing and validation
- **js-cookie**: Cookie management

Real-time Features

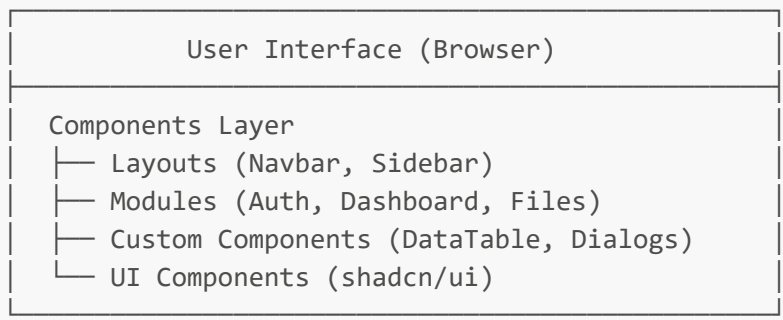
- **Firebase 12.3.0**: Push notifications and messaging
- **Service Workers**: Background notification handling

Data Display & Interaction

- **@tanstack/react-table 8.21.3**: Advanced table features
- **moment**: Date formatting and manipulation
- **Sonner**: Toast notifications

Architecture Layers

1. Presentation Layer



2. Application Layer

```
┌─────────────────────────────────────────────────┐
│  App Router (Next.js)                           │
│  ├── (auth) - Public routes                     │
│  ├── (dashboard) - Protected routes             │
│  └── Middleware - Route protection              │
├─────────────────────────────────────────────────┤
│  Providers                                      │
│  ├── QueryProvider (React Query)                │
│  ├── ThemeProvider (Dark/Light mode)            │
│  ├── NuqsAdapter (URL state)                    │
│  └── HasCheckedProvider (Notifications)         │
└─────────────────────────────────────────────────┘
```

3. Business Logic Layer

```
┌─────────────────────────────────────────────────┐
│  Data Layer (React Query Hooks)                 │
│  ├── auth.ts - Authentication operations        │
│  ├── files.ts - File management operations      │
│  ├── user.ts - User profile operations          │
│  └── notifications.ts - Notification ops        │
├─────────────────────────────────────────────────┤
│  Schema Validation (Zod)                        │
│  ├── auth.ts - Auth schemas                     │
│  ├── files.ts - File schemas                    │
│  ├── users.ts - User schemas                    │
│  └── notifications.ts - Notification schemas    │
└─────────────────────────────────────────────────┘
```

4. Network Layer

```
┌─────────────────────────────────────────────────┐
│  Axios Instances                                │
│  ├── AxiosInstance (base)                       │
│  ├── AxiosInstanceWithToken (authenticated)     │
│  ├── AxiosInstancemultipart (file uploads)      │
│  └── AxiosInstancemultipartWithToken            │
├─────────────────────────────────────────────────┤
│  Interceptors                                   │
│  ├── Request: JWT token injection               │
│  ├── Response: Error handling                   │
│  └── Token refresh logic                        │
└─────────────────────────────────────────────────┘
```

5. External Services Layer

```
┌─────────────────────────────────────────────────┐
│  Backend API (Django REST Framework)            │
│  Base URL: http://localhost:8000                │
│  ├── /api/v1/auth/* - Authentication            │
│  ├── /api/v1/user/* - User management           │
│  ├── /api/v1/files/* - File operations          │
│  └── /api/v1/notifications/* - Notifications    │
├─────────────────────────────────────────────────┤
│  Firebase Services                              │
│  ├── Cloud Messaging (FCM)                      │
│  ├── Service Worker                             │
│  └── Push Notifications                         │
└─────────────────────────────────────────────────┘
```

Core System Components

1. Authentication System

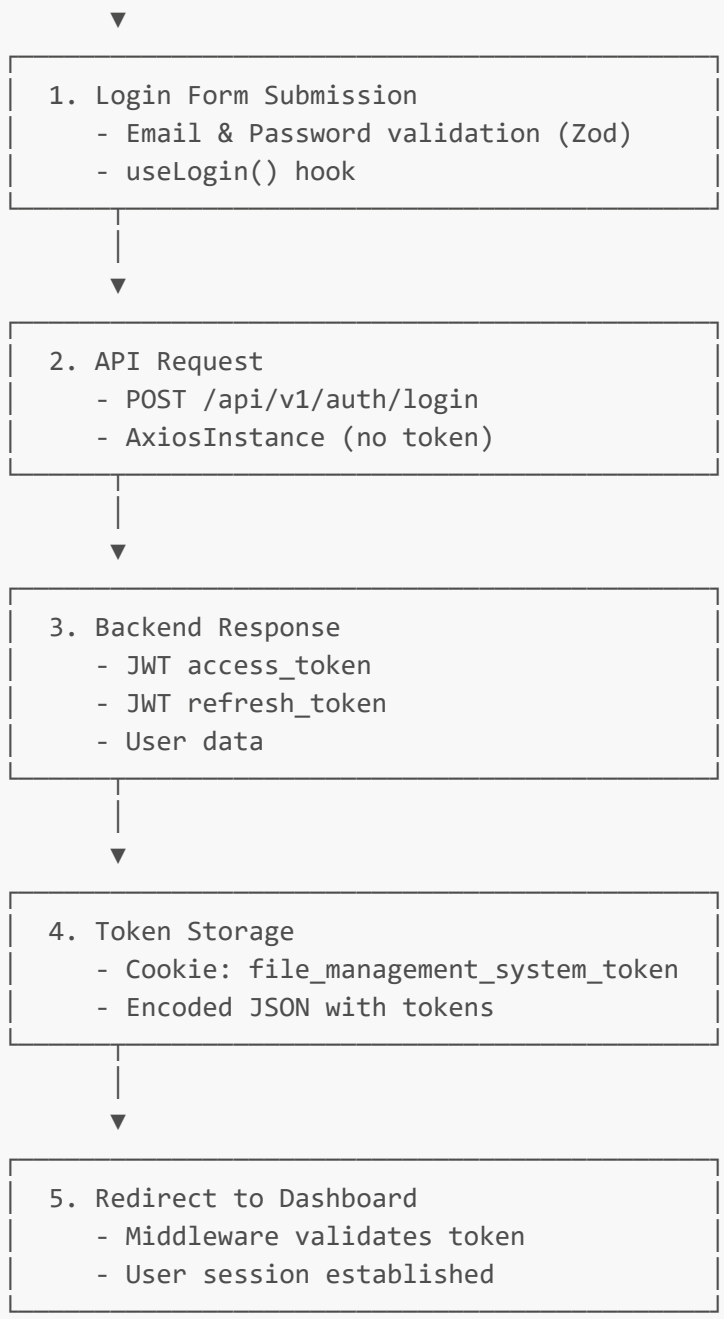
Component Structure

```
// Entry Points
├── app/(auth)/
│   ├── login/ - Login page
│   ├── register/ - Registration page
│   ├── verify-email/ - Email verification
│   ├── verify-2FA/ - Two-factor authentication
│   ├── forgot-password/ - Password reset request
│   ├── reset-password/ - Password reset confirmation
│   └── oauth_success/ - OAuth callback handling

// Authentication Components
├── components/modules/auth/
│   ├── login-form.tsx - Login form with validation
│   ├── register-form.tsx - Registration form
│   ├── verify-email.tsx - Email verification UI
│   ├── verify-2FA.tsx - 2FA code input
│   ├── forgot-password.tsx - Password reset form
│   ├── reset-password.tsx - New password form
│   └── logout-modal.tsx - Logout confirmation
```

Authentication Flow

```
┌──────────────┐
│   User       │
└──────┬───────┘
       │
```



Middleware Protection

```
// middleware.ts
// Route Classification:
// - authRoutes: login, register, verify-email, etc.
// - publicRoutes: blogs, privacy-policy, terms
// - protected: everything else

Flow:
1. Check for auth token cookie
2. If token exists:
   - Decode and validate JWT
   - Check expiration
   - Redirect authenticated users away from auth routes
3. If no token:
```

```
    - Allow access to public/auth routes
    - Redirect to login for protected routes
```

---

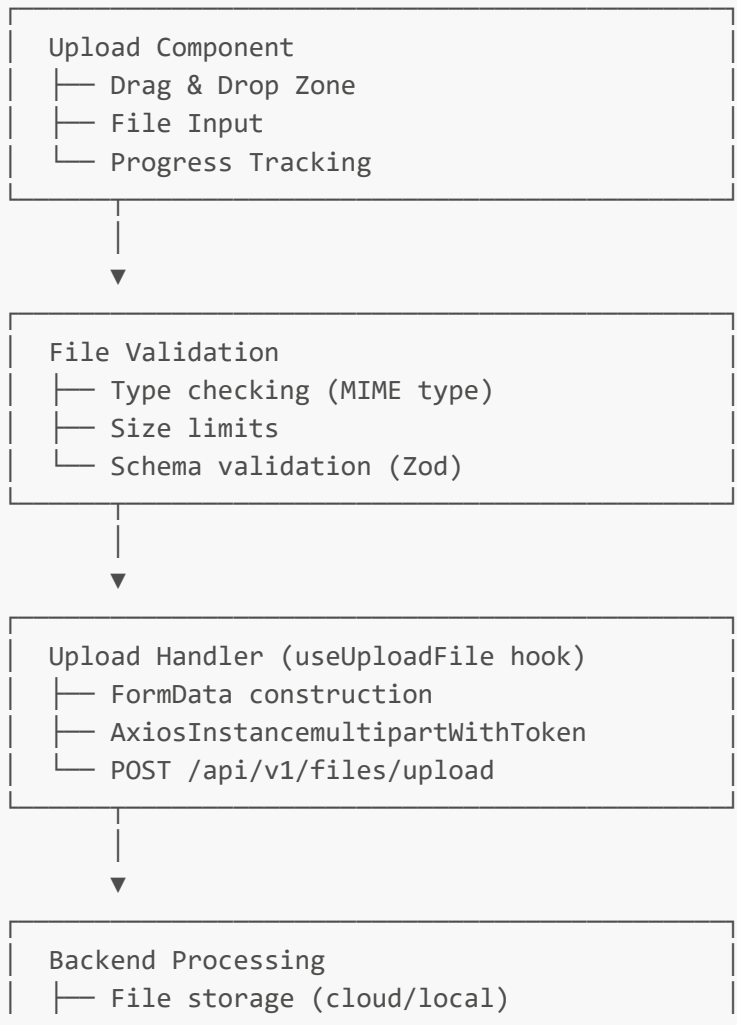
2. File Management System

File Type Categories

```
type FileType = 'image' | 'document' | 'video' | 'audio' | 'other';

Supported Operations:
├── Upload (single & multiple)
├── Download
├── Delete (single & batch)
├── Update metadata
├── Share (social & link)
├── Search & Filter
└── Analytics & Storage tracking
```

File Upload Architecture



```
│   ├── Metadata extraction                    │
│   ├── Storage quota update                   │
│   └── Database record creation               │
└──────────────────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────┐
│  Response & UI Update                        │
│   ├── File URL returned                      │
│   ├── React Query cache invalidation         │
│   ├── Toast notification                     │
│   └── UI refresh with new file               │
└──────────────────────────────────────────────┘
```

Data Table Features

```
// components/custom/datatable.tsx
Features:
├── Multi-column sorting
├── Advanced filtering (by name, type, date)
├── Pagination (configurable page size)
├── Column visibility toggle
├── Row selection (single & bulk)
├── Bulk operations (delete)
├── Search across columns
├── Loading skeletons
└── Responsive design

State Management:
├── columnFilters: ColumnFiltersState
├── columnVisibility: VisibilityState
├── pagination: PaginationState
├── sorting: SortingState
└── React Table instance
```

---

3. Dashboard & Analytics

Dashboard Components

```
├── components/modules/dashboard/
│   ├── container.tsx - Main dashboard layout
│   ├── storage-panel.tsx - Storage usage display
│   ├── storage-chart.tsx - Radial chart component
│   ├── quick-action.tsx - Quick action buttons
│   └── dummy-data.tsx - Sidebar configuration

Dashboard Features:
```

```
├── Storage Overview
│   ├── Total space vs. used space
│   ├── Radial progress chart
│   └── File type breakdown
├── File Type Distribution
│   ├── Images count & size
│   ├── Videos count & size
│   ├── Audio count & size
│   ├── Documents count & size
│   └── Other files count & size
└── Quick Actions
    ├── Upload new file
    ├── View recent files
    └── Access file categories
```

Storage Calculation

```
// Real-time calculation from backend
Storage Info:
├── total_space: Allocated storage limit
├── used_space: Currently used storage
└── Percentage: (used_space / total_space) * 100

File Aggregation:
├── Group files by type
├── Sum sizes per type
├── Count files per type
└── Display with icons (Lucide)
```

---

4. Notification System

Firebase Cloud Messaging Integration

```
┌─────────────────────────────────────────────┐
│  useFcmToken Hook (Client-side)             │
│  ├── Request notification permission        │
│  ├── Fetch FCM token                        │
│  ├── Register service worker                │
│  └── Send token to backend                  │
└─────────────────────────────────────────────┘
        │
        ▼
┌─────────────────────────────────────────────┐
│  Service Worker (firebase-messaging-sw.js)  │
│  ├── Listen for background messages         │
│  ├── Display notifications                  │
│  └── Handle notification clicks             │
```

```
└──────────────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────┐
│  Backend Notification Service                │
│  ├── Store user FCM tokens                   │
│  ├── Trigger notifications                   │
│  └── Send via Firebase Admin SDK             │
└──────────────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────┐
│  Frontend Reception                          │
│  ├── onMessage listener (foreground)         │
│  ├── Update notification list                │
│  ├── Show toast notification                 │
│  └── Update unread count badge               │
└──────────────────────────────────────────────┘
```

Notification Management

```
// Notification Types
├── System notifications (backend-triggered)
├── User notifications (sent by users)
└── Activity notifications (file operations)

// UI Components
├── components/custom/navbar/notification-btn.tsx
│   ├── Bell icon with badge
│   ├── Unread count display
│   └── Dropdown with recent notifications
├── components/modules/notifications/
│   ├── notification-container.tsx
│   ├── notification-card.tsx
│   └── notification-pagination.tsx
└── app/(dashboard)/notifications/page.tsx
```

Data Flow Architecture

Request Flow (Protected Route)

```
User Action
    │
    ▼
Component Event Handler
    │
    ▼
React Query Hook (useMutation/useQuery)
```

```
    │
    ▼
Data Layer Function (e.g., useGetAllFiles)
    │
    ▼
Axios Instance with Interceptor
    │
    ├──> Request Interceptor
    │    ├── Get token from cookies
    │    ├── Decode & validate JWT
    │    ├── Add Authorization header
    │    └── Continue request
    │
    ▼
Backend API Endpoint
    │
    ▼
Response from Backend
    │
    ├──> Response Interceptor
    │    ├── Check for errors
    │    ├── Handle 401 (redirect to login)
    │    └── Return data
    │
    ▼
React Query Cache Update
    │
    ▼
Component Re-render with New Data
    │
    ▼
UI Update (optimistic or after success)
```

Form Submission Flow

```
User fills form
    │
    ▼
React Hook Form (validation on blur/change)
    │
    ▼
Zod Schema Validation
    │
    ├──> If invalid: Show error messages
    └──> If valid: Continue
         │
         ▼
Form submit handler
    │
    ▼
useMutation hook (React Query)
```

```
    │
    ▼
API call via Axios
    │
    ▼
Backend processing
    │
    ▼
Response handling
    │
    ├──> Success:
    │    ├── Cache invalidation
    │    ├── Toast notification
    │    ├── Redirect or UI update
    │    └── Reset form
    │
    └──> Error:
         ├── Display error message
         └── Keep form data
```

Authentication & Authorization

JWT Token Management

```
// Token Structure
{
  access_token: string;   // Short-lived (e.g., 15 min)
  refresh_token: string;  // Long-lived (e.g., 7 days)
  user: {
    id: string;
    email: string;
  }
}

// Storage: HTTP-only cookie approach
Cookie Name: file_management_system_token
Value: JSON.stringify(token)
Secure: true (production)
SameSite: strict
```

Role-Based Access Control (RBAC)

```
// User Roles (from schema)
enum Role {
  SUPER_ADMIN = "super_admin",
  ADMIN = "admin",
  BUSINESS_USER = "business_user",
  STANDARD_USER = "standard_user"
```

```
}

// Permission Matrix (backend enforced)
Permission by Role:
├── super_admin: All operations
├── admin: User management, file management
├── business_user: Extended file operations, analytics
└── standard_user: Basic file operations
```

Protected Route Implementation

```
// middleware.ts checks:
1. Token existence
2. Token validity (decode JWT)
3. Token expiration
4. Route accessibility based on auth state

// Component-level protection:
1. useGetCurrentUserProfile() hook
2. Conditional rendering based on user role
3. API-level permission checks
```

State Management

React Query (Server State)

```
// Query Keys Organization
const queryKeys = {
  // User
  user: {
    all: ['user', 'all'],
    profile: ['user', 'profile'],
    activity: (userId: string) => ['user', 'activity', userId]
  },

  // Files
  files: {
    all: (params?: object) => ['files', 'all', params],
    detail: (id: string) => ['files', 'detail', id],
    byType: (type: string) => ['files', 'type', type]
  },

  // Storage
  storage: {
    info: ['storage', 'info'],
    analytics: ['storage', 'analytics']
  },
```

```
  // Notifications
  notifications: {
    all: ['notifications'],
    unread: ['userUnreadNotifications'],
    userSent: ['userNotifications']
  }
};

// Cache Configuration
QueryClient Settings:
├── staleTime: 5 minutes (default)
├── cacheTime: 30 minutes
├── refetchOnWindowFocus: true
├── refetchOnReconnect: true
└── retry: 3 attempts
```

Context API (Client State)

```
// Theme Context (next-themes)
├── Current theme (light/dark/system)
├── Toggle function
└── Persisted in localStorage

// Notification Context (HasCheckedProvider)
├── hasChecked: boolean
├── setHasChecked: function
└── Tracks notification permission state

// User Context (via React Query)
├── Cached in React Query
├── Accessible globally
└── Auto-synced with backend
```

URL State Management (nuqs)

```
// Used for:
├── Search parameters
├── Filter states
├── Pagination state
├── Sort order
└── Tab selection

// Benefits:
├── Shareable URLs
├── Browser back/forward support
├── Server-side rendering compatible
└── Type-safe with TypeScript
```

Security Architecture

Client-Side Security Measures

1. Token Security

```
// Token validation before each request
const getToken = () => {
  const tokenData = Cookies.get(COOKIE_NAME);
  const decoded = jwtDecode<JwtPayload>(token);

  // Check expiration
  if (decoded.exp < Date.now() / 1000) {
    Cookies.remove(COOKIE_NAME);
    return null;
  }

  return token;
};
```

2. XSS Prevention

```
├── React automatic escaping
├── No dangerouslySetInnerHTML usage
├── Content Security Policy headers (backend)
└── Input sanitization via Zod schemas
```

3. CSRF Protection

```
├── SameSite cookie attribute
├── CSRF tokens from backend
├── Origin validation
└── Secure cookie flag in production
```

4. Input Validation

```
// All inputs validated through Zod schemas
Example:
├── Email format validation
├── Password strength requirements
├── File size limits
├── File type restrictions
└── SQL injection prevention (backend)
```

Performance Optimization

1. Code Splitting

```
Next.js automatic code splitting:
├── Route-based splitting (each page)
├── Component-level dynamic imports
├── Lazy loading for heavy components
└── Reduced initial bundle size
```

2. Image Optimization

```
// next/image component
├── Automatic format conversion (WebP)
├── Responsive images
├── Lazy loading by default
├── Blur placeholder support
└── Remote image optimization
```

3. React Query Optimizations

```
// Caching Strategy
├── Stale-while-revalidate
├── Background refetching
├── Optimistic updates
├── Query deduplication
└── Automatic garbage collection

// Example: File list query
useGetAllFiles({
  staleTime: 5 * 60 * 1000,  // 5 minutes
  cacheTime: 30 * 60 * 1000,  // 30 minutes
  refetchOnMount: false,       // Use cache if fresh
});
```

4. Bundle Optimization

```
├── Tree shaking (automatic)
├── Dynamic imports for large libraries
├── SVG optimization
├── Font optimization (next/font)
└── Minification in production
```

5. Loading States

```
├── Skeleton screens (shadcn/ui)
├── Loading spinners
├── Suspense boundaries
├── Streaming SSR (Next.js)
└── Progressive enhancement
```

Deployment Architecture

Build Process

```
# Development
npm run dev  # Turbopack for fast HMR

# Production
npm run build  # Next.js optimized build
npm run start  # Production server
```

Environment Variables

```
Required Variables:
├── NEXT_PUBLIC_API_URL - Backend API URL
├── NEXT_PUBLIC_SITE_URL - Frontend URL
├── NEXT_PUBLIC_COOKIE_NAME - Auth cookie name
├── NEXT_PUBLIC_FIREBASE_* - Firebase config
└── See .env.local for complete list
```

Production Considerations

```
├── Static asset hosting (Vercel/CDN)
├── API proxy configuration
├── Error tracking (Sentry)
├── Analytics integration
├── HTTPS enforcement
├── CORS configuration
└── Rate limiting (backend)
```

Conclusion

This architecture provides a robust, scalable, and maintainable foundation for the File Management System. Key architectural decisions include:

1. **Separation of Concerns**: Clear boundaries between presentation, business logic, and data layers
2. **Type Safety**: End-to-end TypeScript with Zod schema validation
3. **Performance First**: React Query caching, code splitting, and optimistic updates
4. **Security**: JWT authentication, role-based access, and input validation
5. **Developer Experience**: Modern tooling, clear patterns, and comprehensive type inference
6. **User Experience**: Real-time updates, loading states, and responsive design

The modular structure allows for easy feature additions and maintenance while maintaining code quality and performance standards.