Project Usage Guide

File Management System - Developer Guide

Table of Contents

Getting Started

Prerequisites

- **Node.js**: 20.x or higher
- **Package Manager**: pnpm (recommended), npm, or yarn
- **Backend API**: Running Django backend at `http://localhost:8000`
- **Firebase Account**: For push notifications (optional for basic functionality)
- **Git**: For version control

Quick Start

```
# 1. Clone the repository
git clone <repository-url>
cd projectfrontend

# 2. Install dependencies
pnpm install

# 3. Copy environment file
cp .env.example .env.local

# 4. Configure environment variables
# Edit .env.local with your settings

# 5. Run development server
pnpm dev

# 6. Open browser
# Navigate to http://localhost:3000
```

Project Setup

1. Environment Configuration

Create a `.env.local` file in the root directory:

```
# API Configuration
NEXT_PUBLIC_API_URL=http://localhost:8000
API_URL=http://localhost:8000
NEXT_PUBLIC_SITE_URL=http://localhost:3000

# Environment
NODE_ENV=development

# Authentication
NEXT_PUBLIC_COOKIE_NAME=file_management_system_token

# Firebase Configuration (for notifications)
NEXT_PUBLIC_FIREBASE_API_KEY=your_api_key
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=your_project.firebaseapp.com
NEXT_PUBLIC_FIREBASE_PROJECT_ID=your_project_id
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=your_project.appspot.com
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=your_sender_id
NEXT_PUBLIC_FIREBASE_APP_ID=your_app_id
NEXT_PUBLIC_FIREBASE_MEASUREMENT_ID=your_measurement_id
NEXT_PUBLIC_FIREBASE_VAPIDKEY=your_vapid_key

# Optional: Firebase Private Key for server-side
FIREBASE_PRIVATE_KEY=your_private_key
```

2. Backend Setup

Ensure the Django backend is running:

```
# Backend should be accessible at:
http://localhost:8000

# Required endpoints:
- /api/v1/auth/*       - Authentication
- /api/v1/user/*       - User management
- /api/v1/files/*      - File operations
- /api/v1/notifications/* - Notifications
```

3. Firebase Setup (Optional)

For push notifications:

1. Create a Firebase project at Firebase Console

2. Enable Cloud Messaging
3. Generate Web Push certificates (VAPID key)
4. Download service account credentials
5. Add configuration to `.env.local`

4. Install Dependencies

```
# Using pnpm (recommended)
pnpm install

# Or using npm
npm install

# Or using yarn
yarn install
```

Development Workflow

Running the Development Server

```
# Start development server with Turbopack (fast HMR)
pnpm dev

# Development server runs at:
# http://localhost:3000
```

Available Scripts

```
# Development
pnpm dev              # Start dev server with Turbopack
pnpm dev:standard     # Start dev server without Turbopack

# Production Build
pnpm build            # Create optimized production build
pnpm start            # Start production server

# Code Quality
pnpm lint             # Run ESLint
pnpm type-check       # Run TypeScript type checking

# Testing (if configured)
pnpm test             # Run tests
pnpm test:watch       # Run tests in watch mode
```

Project Structure

```
projectfrontend/
├── app/                      # Next.js App Router
│   ├── (auth)/              # Auth routes (login, register, etc.)
│   ├── (dashboard)/         # Protected dashboard routes
│   ├── layout.tsx           # Root layout
│   └── globals.css          # Global styles
│
├── components/              # React components
│   ├── custom/             # Custom reusable components
│   ├── layouts/            # Layout components (Navbar, Sidebar)
│   ├── modules/            # Feature-specific components
│   └── ui/                 # shadcn/ui components
│
├── data/                    # API integration layer
│   ├── auth.ts            # Authentication API hooks
│   ├── files.ts           # File management API hooks
│   ├── user.ts            # User API hooks
│   ├── notifications.ts   # Notification API hooks
│   ├── instance.ts        # Axios instances
│   └── constants.ts       # API constants
│
├── schemas/                # Zod validation schemas
│   ├── auth.ts           # Auth schemas
│   ├── files.ts          # File schemas
│   ├── users.ts          # User schemas
│   └── notifications.ts  # Notification schemas
│
├── types/                 # TypeScript type definitions
│   ├── auth.ts
│   ├── files.ts
│   ├── user.ts
│   └── notifications.ts
│
├── hooks/                 # Custom React hooks
│   ├── useFcmToken.ts   # Firebase token management
│   ├── use-file-upload.ts
│   └── use-mobile.ts
│
├── providers/             # Context providers
│   ├── index.tsx        # Main providers wrapper
│   ├── react-query/     # React Query provider
│   ├── theme/           # Theme provider
│   └── context/         # Custom contexts
│
├── lib/                   # Utility libraries
│   ├── firebase.ts      # Firebase configuration
│   └── utils.ts         # Utility functions
│
├── utils/                # Helper functions
│   ├── auth.ts         # Auth utilities
│   └── utility.ts      # General utilities
│
```

```
├── public/             # Static assets
│   ├── firebase-messaging-sw.js  # Service worker
│   └── images/
│
└── Configuration Files
    ├── .env.local           # Environment variables
    ├── next.config.ts       # Next.js configuration
    ├── tsconfig.json        # TypeScript configuration
    ├── components.json      # shadcn/ui configuration
    ├── tailwind.config.js   # Tailwind CSS configuration
    └── package.json         # Dependencies and scripts
```

Feature Guide

1. Authentication Features

User Registration

```
// Location: app/(auth)/register/page.tsx
// Component: components/modules/auth/register-form.tsx

Features:
- Email and password registration
- Real-time form validation (Zod)
- Password strength indicator
- Email verification flow
- Error handling and display
```

Usage Flow:

1. User fills registration form
2. Form validates via `RegisterSchema`
3. `useSignup()` hook submits to API
4. Email verification sent
5. User redirected to verify-email page

User Login

```
// Location: app/(auth)/login/page.tsx
// Component: components/modules/auth/login-form.tsx

Features:
- Email/password authentication
- Remember me functionality
- Password reset link
- 2FA support (if enabled)
- OAuth integration ready
```

Usage Flow:

1. User enters credentials
2. `useLogin()` hook authenticates
3. JWT tokens stored in cookie
4. Optional 2FA verification
5. Redirect to dashboard

Two-Factor Authentication (2FA)

```
// Location: app/(auth)/verify-2FA/page.tsx
// Component: components/modules/auth/verify-2FA.tsx

Features:
- 6-digit code input (input-otp)
- Countdown timer
- Resend code option
- Auto-submit on complete
```

Password Reset

```
// Forgot Password: app/(auth)/forgot-password/page.tsx
// Reset Password: app/(auth)/reset-password/page.tsx

Flow:
1. User requests reset (email)
2. Backend sends reset link with token
3. User clicks link → reset-password page
4. New password set via token validation
```

---

2. File Management Features

File Upload

Single File Upload:

```
// Example usage in components
import { useUploadFile } from "@/data/files";

const UploadComponent = () => {
  const uploadFile = useUploadFile();

  const handleUpload = (file: File) => {
    const formData = new FormData();
```

```
    formData.append('file', file);

    uploadFile.mutate(formData, {
      onSuccess: (data) => {
        console.log('File uploaded:', data.file_id);
      }
    });
  };
};
```

Bulk Upload:

```
import { useUploadMultipleFiles } from "@/data/files";

const handleBulkUpload = (files: File[]) => {
  const formData = new FormData();
  files.forEach(file => formData.append('files', file));

  uploadMultiple.mutate(formData);
};
```

Upload Locations:

- Images: `app/(dashboard)/images/upload/page.tsx`
- Videos: `app/(dashboard)/videos/upload/page.tsx`
- Audio: `app/(dashboard)/audios/upload/page.tsx`
- Documents: `app/(dashboard)/documents/upload/page.tsx`

File Listing & Search

```
// Data Table component
// Location: components/custom/datatable.tsx

Features:
├── Column sorting (asc/desc)
├── Search by filename
├── Filter by file type
├── Multi-select rows
├── Bulk delete
├── Pagination
├── Column visibility toggle
└── Responsive design

Usage:
<FilesDataTable
  data={files}
  loading={isLoading}
  onDeleteFiles={handleDelete}
  uploadlink="/images/upload"
```

```
  buttonText="Upload Image"
/>
```

File Operations

View File:

```
const { data: file } = useGetFileById(fileId);
// Returns: FileResponse with URL, metadata
```

Delete File:

```
const deleteFile = useDeleteFile();

deleteFile.mutate(fileId, {
  onSuccess: () => {
    toast.success('File deleted successfully');
  }
});
```

Update File Metadata:

```
const updateFile = useUpdateFile();

updateFile.mutate({
  fileId: 'uuid',
  data: { file_name: 'New Name' }
});
```

Share File:

```
// Component: components/custom/social-share.tsx

Features:
- Copy link to clipboard
- Social media sharing (Facebook, Twitter, WhatsApp, LinkedIn)
- Email sharing
- Direct link generation
```

---

3. Dashboard Features

Storage Panel

```
// Location: components/modules/dashboard/storage-panel.tsx

Features:
├── Total storage display
├── Used storage visualization
├── Radial progress chart
├── File type breakdown
│   ├── Images
│   ├── Videos
│   ├── Audio
│   ├── Documents
│   └── Other
└── Loading skeleton

Data Source:
- useGetStorageInfo() - Overall storage
- useGetAllFiles() - File breakdown
```

Quick Actions

```
// Location: components/modules/dashboard/quick-action.tsx

Available Actions:
├── Upload new file
├── View recent files
├── Access file categories
└── Manage notifications
```

Recent Activity

```
// Location: app/(dashboard)/recent/page.tsx
// Component: components/modules/recent/recent-timeline.tsx

Features:
├── Timeline view
├── Activity types (upload, delete, update, share)
├── Timestamp display
├── Activity filtering
└── Pagination
```

---

4. Notification System

Setting Up Notifications

```
// Automatic setup in dashboard layout
// Location: app/(dashboard)/layout.tsx

const { token, notificationPermissionStatus } = useFcmToken();

Flow:
1. Request permission on dashboard load
2. Generate FCM token
3. Send token to backend
4. Register service worker
5. Listen for messages
```

Creating Notifications

```
// Location: app/(dashboard)/manage-notifications/page.tsx
// Component: components/modules/manage-notifications/notification-form.tsx

import { useCreateNotification } from "@/data/notifications";

const createNotification = useCreateNotification();

createNotification.mutate({
  title: "New File Uploaded",
  message: "Your file has been successfully uploaded",
  recipients: ["user_id_1", "user_id_2"]
});
```

Viewing Notifications

```
// Notification button in navbar
// Location: components/custom/navbar/notification-btn.tsx

Features:
├── Unread count badge
├── Dropdown with recent notifications
├── Mark as read functionality
├── Navigate to full notification page
└── Real-time updates (FCM)

// Full notification page
// Location: app/(dashboard)/notifications/page.tsx
```

---

5. User Profile Management

View Profile

```
// Location: app/(dashboard)/profile/page.tsx
// Component: components/modules/profile/profile-card.tsx

Displays:
├── Avatar
├── Name
├── Email
├── Role
├── Join date
└── Account statistics
```

Edit Profile

```
// Component: components/modules/profile/profile-edit-modal.tsx

Features:
├── Update name
├── Update email
├── Change avatar
├── Update phone number
├── Location selection
├── Form validation (react-hook-form + Zod)
└── Optimistic updates

Usage:
const updateUser = useUpdateUser();

updateUser.mutate({
  first_name: "John",
  last_name: "Doe",
  phone_number: "+1234567890"
});
```

Component Usage

shadcn/ui Components

The project uses shadcn/ui for consistent, accessible components.

Adding New Components

```
# Add a new shadcn/ui component
npx shadcn@latest add [component-name]

# Examples:
npx shadcn@latest add button
```

```
npx shadcn@latest add dialog
npx shadcn@latest add form
```

Commonly Used Components

Button:

```
import { Button } from "@/components/ui/button";

<Button variant="default" size="md">
  Click me
</Button>

// Variants: default, destructive, outline, secondary, ghost, link
// Sizes: default, sm, lg, icon
```

Dialog:

```
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";

<Dialog>
  <DialogTrigger asChild>
    <Button>Open</Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Title</DialogTitle>
      <DialogDescription>Description</DialogDescription>
    </DialogHeader>
    {/* Content */}
  </DialogContent>
</Dialog>
```

Form with Validation:

```
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";
import {
  Form,
```

```
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from "@/components/ui/form";
import { Input } from "@/components/ui/input";

const formSchema = z.object({
  email: z.string().email(),
  password: z.string().min(6),
});

function MyForm() {
  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      email: "",
      password: "",
    },
  });

  const onSubmit = (values: z.infer<typeof formSchema>) => {
    console.log(values);
  };

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)}>
        <FormField
          control={form.control}
          name="email"
          render={({ field }) => (
            <FormItem>
              <FormLabel>Email</FormLabel>
              <FormControl>
                <Input placeholder="email@example.com" {...field} />
              </FormControl>
              <FormMessage />
            </FormItem>
          )}
        />
        <Button type="submit">Submit</Button>
      </form>
    </Form>
  );
}
```

Data Table:

```
// Full example in: components/custom/datatable.tsx
```

```
import { useReactTable } from "@tanstack/react-table";

// Define columns
const columns: ColumnDef<FileResponse>[] = [
  {
    accessorKey: "file_name",
    header: "File Name",
  },
  // ... more columns
];

// Use in component
<FilesDataTable
  data={files}
  loading={isLoading}
  onDeleteFiles={(ids) => console.log(ids)}
/>
```

API Integration

Creating New API Hooks

Step 1: Define Types

```
// types/your-feature.ts
export interface YourDataType {
  id: string;
  name: string;
  // ... more fields
}

export interface YourListResponse {
  items: YourDataType[];
  count: number;
}
```

Step 2: Create Zod Schema

```
// schemas/your-feature.ts
import { z } from "zod";

export const YourDataSchema = z.object({
  name: z.string().min(1, "Name is required"),
  // ... validation rules
});

export type YourDataSchemaType = z.infer<typeof YourDataSchema>;
```

Step 3: Create API Hook

```
// data/your-feature.ts
import { useQuery, useMutation, useQueryClient } from "react-query";
import { AxiosInstanceWithToken } from "@/data/instance";

// GET - Fetch data
export const useGetYourData = () => {
  return useQuery({
    queryKey: ["your-data", "all"],
    queryFn: async (): Promise<YourListResponse> => {
      const response = await AxiosInstanceWithToken.get("/api/v1/your-endpoint/");
      return response.data;
    },
    onError: (error: any) => {
      console.error("Error fetching data:", error);
    },
  });
};

// POST - Create data
export const useCreateYourData = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (data: YourDataSchemaType): Promise<YourDataType> => {
      const validatedData = YourDataSchema.parse(data);
      const response = await AxiosInstanceWithToken.post(
        "/api/v1/your-endpoint/",
        validatedData
      );
      return response.data;
    },
    onSuccess: () => {
      // Invalidate and refetch
      queryClient.invalidateQueries(["your-data", "all"]);
      toast.success("Data created successfully");
    },
    onError: (error: any) => {
      toast.error("Failed to create data");
    },
  });
};

// PUT/PATCH - Update data
export const useUpdateYourData = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async ({
      id,
```

```
      data
    }: {
      id: string;
      data: Partial<YourDataSchemaType>
    }) => {
      const response = await AxiosInstanceWithToken.patch(
        `/api/v1/your-endpoint/${id}/`,
        data
      );
      return response.data;
    },
    onSuccess: () => {
      queryClient.invalidateQueries(["your-data"]);
    },
  });
};

// DELETE - Remove data
export const useDeleteYourData = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (id: string) => {
      const response = await AxiosInstanceWithToken.delete(
        `/api/v1/your-endpoint/${id}/`
      );
      return response.data;
    },
    onSuccess: () => {
      queryClient.invalidateQueries(["your-data"]);
      toast.success("Data deleted");
    },
  });
};
```

Step 4: Use in Component

```
"use client";

import { useGetYourData, useCreateYourData } from "@/data/your-feature";

export default function YourComponent() {
  const { data, isLoading, error } = useGetYourData();
  const createData = useCreateYourData();

  const handleCreate = () => {
    createData.mutate({
      name: "New Item",
    });
  };
```

```
  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error loading data</div>;

  return (
    <div>
      {data?.items.map((item) => (
        <div key={item.id}>{item.name}</div>
      ))}
      <button onClick={handleCreate}>Create New</button>
    </div>
  );
}
```

Common Development Tasks

1. Adding a New Protected Route

```
// 1. Create page file
// app/(dashboard)/your-feature/page.tsx

"use client";

import { useGetCurrentUserProfile } from "@/data/user";

export default function YourFeaturePage() {
  const { data: user, isLoading } = useGetCurrentUserProfile();

  if (isLoading) return <div>Loading...</div>;

  return (
    <div>
      <h1>Your Feature</h1>
      {/* Your content */}
    </div>
  );
}

// 2. Add to sidebar
// components/modules/dashboard/dummy-data.tsx

export const sidebarItems = [
  // ... existing items
  {
    label: "Your Feature",
    href: "/your-feature",
    icon: YourIcon,
  },
];

// 3. Middleware automatically protects routes in (dashboard)
```

2. Adding a New Form

```
// 1. Create schema
// schemas/your-form.ts
import { z } from "zod";

export const YourFormSchema = z.object({
  field1: z.string().min(1, "Required"),
  field2: z.string().email("Invalid email"),
});

// 2. Create component
"use client";

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { Form, FormField, FormItem, FormLabel, FormControl, FormMessage } from
"@/components/ui/form";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { YourFormSchema } from "@/schemas/your-form";

export function YourForm() {
  const form = useForm({
    resolver: zodResolver(YourFormSchema),
    defaultValues: {
      field1: "",
      field2: "",
    },
  });

  const onSubmit = (data: z.infer<typeof YourFormSchema>) => {
    console.log(data);
    // Call API
  };

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
        <FormField
          control={form.control}
          name="field1"
          render={({ field }) => (
            <FormItem>
              <FormLabel>Field 1</FormLabel>
              <FormControl>
                <Input {...field} />
              </FormControl>
              <FormMessage />
            </FormItem>
          )}
```

```
        />
        <Button type="submit">Submit</Button>
      </form>
    </Form>
  );
}
```

3. Adding Theme Support

```
// Components automatically support dark mode via Tailwind classes

// Use dark: prefix for dark mode styles
<div className="bg-white dark:bg-gray-900 text-black dark:text-white">
  Content
</div>

// Access theme in component
import { useTheme } from "next-themes";

function ThemeAwareComponent() {
  const { theme, setTheme } = useTheme();

  return (
    <button onClick={() => setTheme(theme === 'dark' ? 'light' : 'dark')}>
      Toggle Theme
    </button>
  );
}
```

4. Adding Loading States

```
// Skeleton loading (preferred)
import { Skeleton } from "@/components/ui/skeleton";

{isLoading ? (
  <div className="space-y-2">
    <Skeleton className="h-4 w-full" />
    <Skeleton className="h-4 w-3/4" />
    <Skeleton className="h-8 w-1/2" />
  </div>
) : (
  <ActualContent />
)}

// Spinner loading
import { Spinner } from "@/components/custom/spinner";

{isLoading && <Spinner />}
```

5. Error Handling

```
// In components
const { data, isLoading, error } = useGetYourData();

if (error) {
  return (
    <div className="text-red-500">
      Error: {error.message}
    </div>
  );
}

// In mutations
const mutation = useMutation({
  mutationFn: apiCall,
  onError: (error: any) => {
    // Extract backend error message
    const message = error.response?.data?.message || "An error occurred";
    toast.error(message);
  },
  onSuccess: () => {
    toast.success("Success!");
  },
});
```

Troubleshooting

Common Issues

1. "Token expired" errors

Problem: User session expired, causing 401 errors.

Solution:

```
// Automatic handling in middleware.ts
// Manual refresh:
1. Clear cookies
2. Redirect to login
3. User re-authenticates

// Check token validity:
import { getToken } from "@/utils/auth";
const token = getToken(); // Returns null if expired
```

2. CORS errors

Problem: Backend rejecting requests due to CORS policy.

Solution:

```
# Backend (Django settings.py)
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://localhost:8000",
]

CORS_ALLOW_CREDENTIALS = True
```

3. Firebase notification permission denied

Problem: Browser blocking notification permissions.

Solution:

```
// Check permission status
const permission = await Notification.requestPermission();

// User must manually enable in browser settings if blocked
// Guide user to: Site Settings → Notifications → Allow
```

4. Build errors with shadcn/ui

Problem: Component import errors after adding new component.

Solution:

```
# Ensure proper installation
npx shadcn@latest add [component-name]

# Check components.json configuration
# Verify aliases in tsconfig.json
```

5. Hydration errors

Problem: Client/server mismatch causing hydration errors.

Solution:

```
// Use "use client" directive for client-only components
"use client";
```

```
// Or use dynamic import with ssr: false
import dynamic from 'next/dynamic';

const ClientComponent = dynamic(
  () => import('./ClientComponent'),
  { ssr: false }
);
```

6. Environment variables not loading

Problem: `process.env.NEXT_PUBLIC_*` returning undefined.

Solution:

```
# 1. Ensure variables have NEXT_PUBLIC_ prefix for client-side
# 2. Restart dev server after changing .env.local
# 3. Check file name is exactly .env.local

# Verify variables are loaded:
console.log(process.env.NEXT_PUBLIC_API_URL);
```

Best Practices

1. Code Organization

```
// ☑ Good: Organized, single responsibility
components/
  modules/
    dashboard/
      storage-panel.tsx      // Single component
      storage-chart.tsx      // Extracted chart logic

// ❌ Bad: Everything in one file
components/
  dashboard.tsx  // 1000+ lines
```

2. Type Safety

```
// ☑ Good: Full type safety
interface User {
  id: string;
  email: string;
}

const user: User = await fetchUser();
```

```
// ❌ Bad: Using any
const user: any = await fetchUser();
```

3. Form Validation

```
// ☑ Good: Zod schema validation
const schema = z.object({
  email: z.string().email(),
});

// ❌ Bad: Manual validation
if (!email.includes('@')) {
  // error
}
```

4. Error Handling

```
// ☑ Good: Specific error handling
try {
  await uploadFile(file);
} catch (error: any) {
  const message = error.response?.data?.message || "Upload failed";
  toast.error(message);
}

// ❌ Bad: Generic error handling
try {
  await uploadFile(file);
} catch (error) {
  console.log("Error");
}
```

5. Component Composition

```
// ☑ Good: Composable components
<Dialog>
  <DialogTrigger>
    <Button>Open</Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Title</DialogTitle>
    </DialogHeader>
  </DialogContent>
</Dialog>
```

```
// ❌ Bad: Props drilling
<Dialog
  trigger={<Button />}
  title="Title"
  content={<div />}
/>
```

6. Performance

```
// ☑ Good: Memoization for expensive calculations
const sortedFiles = useMemo(() => {
  return files.sort((a, b) => a.name.localeCompare(b.name));
}, [files]);

// ❌ Bad: Calculation on every render
const sortedFiles = files.sort((a, b) => a.name.localeCompare(b.name));
```

7. Accessibility

```
// ☑ Good: Accessible components
<button
  aria-label="Delete file"
  onClick={handleDelete}
>
  <TrashIcon />
</button>

// ❌ Bad: No accessibility
<div onClick={handleDelete}>
  <TrashIcon />
</div>
```

8. State Management

```
// ☑ Good: React Query for server state
const { data } = useGetFiles();

// ❌ Bad: useState for server data
const [files, setFiles] = useState([]);
useEffect(() => {
  fetchFiles().then(setFiles);
}, []);
```

Additional Resources

Documentation

- Next.js Docs
- React Query Docs
- shadcn/ui Docs
- Tailwind CSS Docs
- Zod Docs

Useful Commands

```
# Development
pnpm dev                    # Start dev server
pnpm build                  # Production build
pnpm start                  # Run production build

# Code Quality
pnpm lint                   # Run linter
pnpm lint:fix               # Fix linting issues

# Dependencies
pnpm add [package]          # Add dependency
pnpm add -D [package]       # Add dev dependency
pnpm remove [package]       # Remove dependency
pnpm update                 # Update all dependencies

# shadcn/ui
npx shadcn@latest add [component]   # Add component
npx shadcn@latest diff              # Check for updates
```

Environment Checklist

Before deployment, ensure:

- ☐ All environment variables set
- ☐ Backend API accessible
- ☐ Firebase configured (if using notifications)
- ☐ CORS properly configured
- ☐ Build completes without errors
- ☐ Type checking passes
- ☐ Linting passes
- ☐ All tests pass (if configured)

Conclusion

This guide covers the essential aspects of developing with the File Management System. For specific implementation details, refer to the existing code examples throughout the project. When in doubt, check

similar implementations in the codebase or consult the system architecture documentation.

Quick Tips

1. Always use TypeScript types
2. Validate forms with Zod
3. Use React Query for API calls
4. Follow the established folder structure
5. Write accessible components
6. Test on both light and dark themes
7. Check mobile responsiveness
8. Handle loading and error states

Happy coding! 🚀