# RAGLens Frontend Documentation

## Table of Contents

1. [Introduction](#introduction)
2. [Frontend Architecture](#frontend-architecture)
3. [DocHub Module](#dochub-module)
   * [Component Structure](#component-structure)
   * [State Management](#state-management)
   * [API Integration](#api-integration)
   * [File Operations](#file-operations)
4. [User Interface](#user-interface)
   * [Layout and Styling](#layout-and-styling)
   * [Responsive Design](#responsive-design)
   * [Accessibility](#accessibility)
5. [Development Workflow](#development-workflow)
   * [Project Setup](#project-setup)
   * [Development Server](#development-server)
   * [Building for Production](#building-for-production)
6. [Testing](#testing)
7. [Troubleshooting](#troubleshooting)
8. [References](#references)

## Introduction

RAGLens is a document management and knowledge extraction system with a modern React-based frontend. The application allows users to upload, organize, and process documents while extracting knowledge and building semantic connections between content.

This documentation focuses on the frontend implementation, particularly the DocHub module which handles document management functionality.

## Frontend Architecture

The RAGLens frontend is built using modern React with TypeScript, following a component-based architecture. It uses hooks for state management and side effects, and communicates with the backend through a RESTful API.

### Technology Stack

* **React 18**: UI library
* **TypeScript**: Static typing
* **Axios**: API client
* **React Router**: Navigation
* **Tailwind CSS**: Styling
* **Lucide React**: Icons
* **React Dropzone**: File upload handling

### Architectural Pattern

The frontend follows a modified Model-View-Controller (MVC) pattern adapted for React:

┌───────────────────┐ ┌───────────────────┐ ┌───────────────────┐  
│ │ │ │ │ │  
│ Components │◄────►│ Hooks │◄────►│ API Client │  
│ (View) │ │ (Controller) │ │ (Model) │  
│ │ │ │ │ │  
└───────────────────┘ └───────────────────┘ └───────────────────┘

### Directory Structure

src/  
├── api/  
│ └── axiosInstance.ts # Configured Axios instance  
├── components/  
│ ├── dochub/ # DocHub specific components  
│ │ ├── ActionBar.tsx # Top action buttons  
│ │ ├── DocHubGrid.tsx # Files/folders grid display  
│ │ ├── FileIcon.tsx # File type icons  
│ │ ├── FileItem.tsx # Individual file/folder item  
│ │ ├── TopNavigation.tsx # Breadcrumb navigation  
│ │ ├── CreateFolderModal.tsx # New folder creation modal  
│ │ └── UploadDocumentModal.tsx # Document upload modal  
│ └── shared/ # Shared UI components  
├── contexts/ # React contexts  
│ └── ThemeContext.tsx # Theme management  
├── hooks/ # Custom React hooks  
│ ├── useDocHubActions.ts # DocHub API actions  
│ └── useDocHubStructure.ts # DocHub state management  
├── pages/ # Page components  
│ └── DocHubPage.tsx # Main DocHub page  
├── styles/ # Global styles  
│ └── themes.css # Theme definitions  
├── types/ # TypeScript type definitions  
│ └── dochub.ts # DocHub related types  
├── utils/ # Utility functions  
│ ├── fileUtils.ts # File handling utilities  
│ └── nameUtils.ts # Name generation utilities  
├── App.tsx # Main application component  
└── main.tsx # Application entry point

## DocHub Module

The DocHub module is responsible for document management, including uploading, organizing, and processing documents and folders. It provides a user-friendly interface for interacting with the document repository.

### Component Structure

The DocHub module follows a hierarchical component structure:

┌─────────────────────────────────────────────────────────────┐  
│ DocHubPage │  
├─────────────────────────────────────────────────────────────┤  
│ │  
│ ┌───────────────┐ ┌───────────────┐ ┌───────────────┐ │  
│ │ ActionBar │ │ TopNavigation │ │ DocHubGrid │ │  
│ └───────────────┘ └───────────────┘ └───────────────┘ │  
│ │ │  
│ ┌───────────────┐ ┌───────────────┐ │ │  
│ │CreateFolderMod│ │UploadDocument │ │ │  
│ │ al │ │ Modal │ │ │  
│ └───────────────┘ └───────────────┘ ▼ │  
│ ┌───────────────┐ │  
│ │ FileItem │ │  
│ └───────────────┘ │  
│ │ │  
│ ▼ │  
│ ┌───────────────┐ │  
│ │ FileIcon │ │  
│ └───────────────┘ │  
│ │  
└─────────────────────────────────────────────────────────────┘

### Key Components

#### DocHubPage

Main container component that integrates all DocHub functionality and manages the overall state.

function DocHubPage() {  
 const {  
 folders, documents, currentPath, isLoading, error,  
 getCurrentFolder, getChildFolders, getFolderDocuments,  
 navigateToFolder, createFolderInCurrent, uploadDocumentsToCurrent,  
 deleteDocumentById, deleteFolderById, renameDocumentById, renameFolderById  
 } = useDocHubStructure();  
  
 // Component implementation  
}

#### ActionBar

Provides primary actions like uploading documents and creating folders.

function ActionBar({ onUpload, onCreateFolder }) {  
 return (  
 <div className="flex items-center justify-between h-16 px-4">  
 {/\* Action buttons \*/}  
 </div>  
 );  
}

#### TopNavigation

Displays breadcrumb navigation and allows navigation between folders.

function TopNavigation({ currentFolder, folderPath, onBack, onNavigateToFolder }) {  
 return (  
 <div className="flex items-center h-12 px-4 border-b border-gray-700">  
 {/\* Breadcrumb navigation \*/}  
 </div>  
 );  
}

#### DocHubGrid

Displays a grid of files and folders with their metadata.

function DocHubGrid({  
 folders, documents, selectedItems, onSelectionChange,  
 onNavigateToFolder, onRename, onDelete, onDownload  
}) {  
 return (  
 <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 xl:grid-cols-6 gap-4 p-4">  
 {/\* File and folder items \*/}  
 </div>  
 );  
}

#### FileItem

Represents an individual file or folder with associated actions.

function FileItem({  
 item, isFolder, isSelected, onClick,  
 onRename, onDelete, onDownload  
}) {  
 return (  
 <div className="group flex flex-col items-center p-4 rounded-xl transition-all duration-200 cursor-pointer">  
 {/\* File/folder representation and actions \*/}  
 </div>  
 );  
}

#### Modals

* **CreateFolderModal**: Modal dialog for creating new folders.
* **UploadDocumentModal**: Modal dialog for uploading documents with progress tracking.

### State Management

The DocHub module uses custom React hooks for state management:

#### useDocHubStructure

Manages the overall state of the DocHub module, including folders, documents, current path, and loading states.

function useDocHubStructure() {  
 const [folders, setFolders] = useState<Folder[]>([]);  
 const [documents, setDocuments] = useState<Document[]>([]);  
 const [currentPath, setCurrentPath] = useState<string[]>([]);  
 const [isLoading, setIsLoading] = useState(false);  
 const [error, setError] = useState<string | null>(null);  
   
 // Document processing state  
 const processingDocuments = useRef<DocumentStatus>({});  
   
 // Methods for folder/document operations  
 // ...  
   
 return {  
 // State  
 folders,  
 documents,  
 currentPath,  
 isLoading,  
 error,  
   
 // Getters and actions  
 // ...  
 };  
}

#### useDocHubActions

Handles API interactions for DocHub operations.

function useDocHubActions() {  
 // Folder operations  
 const listFolders = useCallback(async () => { /\* ... \*/ }, []);  
 const createFolder = useCallback(async (data) => { /\* ... \*/ }, []);  
 const renameFolder = useCallback(async (folderId, newName) => { /\* ... \*/ }, []);  
 const deleteFolder = useCallback(async (folderId) => { /\* ... \*/ }, []);  
   
 // Document operations  
 const listDocuments = useCallback(async () => { /\* ... \*/ }, []);  
 const uploadDocument = useCallback(async (file, folderId) => { /\* ... \*/ }, []);  
 const uploadDocuments = useCallback(async (files, folderId) => { /\* ... \*/ }, []);  
 const deleteDocument = useCallback(async (documentId) => { /\* ... \*/ }, []);  
 const renameDocument = useCallback(async (documentId, newName) => { /\* ... \*/ }, []);  
 const getDocumentStatus = useCallback(async (documentId) => { /\* ... \*/ }, []);  
   
 return {  
 // Return all actions  
 // ...  
 };  
}

### Data Flow

The data flow in the DocHub module follows this pattern:

┌──────────────┐ API Calls ┌────────────┐ Updates ┌──────────────┐  
│ │ ─────────────►│ │◄────────────┤ │  
│ Backend API │ │ Hooks │ │ Components │  
│ │◄──────────────┤ │─────────────►│ │  
└──────────────┘ Results └────────────┘ Render └──────────────┘

1. Components trigger actions through hooks
2. Hooks make API calls and update state
3. State changes trigger component re-renders
4. Components display updated data

### API Integration

The DocHub module communicates with the backend through a RESTful API using Axios. The API client is configured in src/api/axiosInstance.ts:

const axiosInstance = axios.create({  
 baseURL: import.meta.env.VITE\_API\_BASE\_URL || 'http://localhost:8000',  
});

Key API endpoints:

| Endpoint | Method | Description |
| --- | --- | --- |
| /api/dochub/folders/ | GET | List all folders |
| /api/dochub/folders/ | POST | Create a new folder |
| /api/dochub/folders/{id}/ | PUT | Update a folder |
| /api/dochub/folders/{id}/ | DELETE | Delete a folder |
| /api/dochub/documents/ | GET | List all documents |
| /api/dochub/documents/ | POST | Upload a document |
| /api/dochub/documents/bulk\_upload/ | POST | Upload multiple documents |
| /api/dochub/documents/{id}/ | PUT | Update a document |
| /api/dochub/documents/{id}/ | DELETE | Delete a document |

### File Operations

#### Document Upload Flow

┌───────────┐ ┌───────────┐ ┌───────────┐ ┌───────────┐  
│ Select │ │ Upload │ │ Process │ │ Ready │  
│ Files │────────►│ Files │────────►│ Documents │────────►│ State │  
│ │ │ │ │ │ │ │  
└───────────┘ └───────────┘ └───────────┘ └───────────┘  
 │ ▲  
 │ │  
 ▼ │  
 ┌───────────┐ │  
 │ Update │ │  
 │ Progress │───────────────┘  
 │ │  
 └───────────┘

1. User selects files through the UploadDocumentModal
2. Files are uploaded to the backend with progress tracking
3. Backend processes documents asynchronously
4. Frontend polls for document status until ready
5. Document icons reflect processing status (processing, ready, error)

#### Folder Navigation Flow

┌───────────┐ ┌───────────┐ ┌───────────┐  
│ Click │ │ Update │ │ Display │  
│ Folder │────────►│ Path │────────►│ Folder │  
│ │ │ │ │ Contents │  
└───────────┘ └───────────┘ └───────────┘  
 ▲ │  
 │ │  
 └────────────────────────────────────────────┘

1. User clicks on a folder in the DocHubGrid
2. The current path is updated in the useDocHubStructure hook
3. The folder contents are fetched and displayed
4. The TopNavigation component updates to show the current path

## User Interface

The DocHub UI is designed to be intuitive and responsive, following modern design principles.

### Layout and Styling

The application uses a combination of Tailwind CSS and custom styling to create a clean, professional interface.

┌─────────────────────────────────────────────────────────────┐  
│ Action Bar │  
├─────────────────────────────────────────────────────────────┤  
│ Top Navigation │  
├─────────────────────────────────────────────────────────────┤  
│ │  
│ │  
│ │  
│ DocHub Grid │  
│ │  
│ │  
│ │  
│ │  
└─────────────────────────────────────────────────────────────┘

#### Key UI Elements

* **ActionBar**: Fixed at the top, providing primary actions
* **TopNavigation**: Shows breadcrumb navigation and allows easy folder traversal
* **DocHubGrid**: Main content area displaying files and folders in a responsive grid
* **Modals**: Overlay dialogs for actions like creating folders and uploading files

### Responsive Design

The UI is designed to be responsive across different screen sizes:

* **Large screens**: Grid displays 6 items per row
* **Medium screens**: Grid adapts to 3-4 items per row
* **Small screens**: Grid reduces to 1-2 items per row

.grid {  
 grid-template-columns: repeat(auto-fill, minmax(170px, 1fr));  
}

### Accessibility

The DocHub UI implements several accessibility features:

* Proper focus management for keyboard navigation
* Descriptive labels for all interactive elements
* ARIA attributes for screen readers
* Sufficient color contrast for text readability
* Keyboard shortcuts for common actions

## Development Workflow

### Project Setup

1. Clone the repository
2. Install dependencies

npm install

1. Set up environment variables in .env:

VITE\_API\_BASE\_URL=http://localhost:8000

### Development Server

Run the development server:

npm run dev

### Building for Production

Build the application for production:

npm run build

## Testing

The application can be tested using:

npm run test

Key testing areas:

1. **Component rendering**: Ensure components render correctly
2. **API interactions**: Test API calls and response handling
3. **User interactions**: Simulate user actions and verify results
4. **Error handling**: Test error scenarios and recovery

## Troubleshooting

### Common Issues

#### CORS Errors

* Ensure backend has proper CORS headers
* Check that API base URL is correctly configured

#### 404 Not Found Errors

* Verify API endpoint paths are correct
* Check that backend server is running

#### File Upload Issues

* Check file size limits
* Verify form data construction
* Ensure correct content type headers

#### Status Tracking Loop

* Verify Celery worker is running
* Check that document processing tasks are executing
* Implement timeout or automatic status resolution for testing

## References

* [React Documentation](https://react.dev/)
* [Tailwind CSS Documentation](https://tailwindcss.com/docs)
* [Axios Documentation](https://axios-http.com/docs/intro)
* [TypeScript Documentation](https://www.typescriptlang.org/docs/)