RDLab - Comprehensive Technical Documentation

Contents

1 Introduction

1.1 Project Overview

The RDLab Project is a web-based platform designed to enhance research assistance and knowledge discovery in the Oil and Gas domain. Leveraging Retrieval-Augmented Generation (RAG) with GraphRAG, it streamlines literature reviews by automatically retrieving, organizing, and cross-referencing large volumes of technical documents, regulatory texts, and incident reports. Unlike traditional research tools, it enables researchers to efficiently identify trends, gaps, and connections within industry data. The system features:

- React-based frontend with role-specific interfaces
- FastAPI backend with RESTful APIs
- Secure user authentication and authorization
- Document management system
- Retrieval-Augmented Generation capabilities

1.2 System Architecture

The system consists of two main components:

- Frontend: React-based web application
- Backend: FastAPI-based services handling business logic and RAG functionality

2 Project Structure

2.1 Root Directory Structure

- `/backend` - Backend FastAPI application
- `/src` - Frontend React application
- `/public` - Static assets

3 Backend Documentation

3.1 Core Structure

- `/app` - Main application package
  - `main.py` - Application entry point
- `run.py` - Application runner script

3.2 Module Breakdown

3.2.1 API Endpoints

- `/api`
  - `admin.py` - Admin-specific endpoints
  - `user.py` - User authentication endpoints

3.2.2 Core Business Logic

- `/core`
  - `/vector_dbs` - Vector database storage
  - `/dataset` - Document datasets
  - `config.py` - Configuration settings
  - `rag_api.py` - RAG implementation
  - `security.py` - Security utilities

3.2.3 Database Layer

- `/db`
  - `database.py` - Database connection
  - `crud.py` - CRUD operations
  - `models.py` - SQLAlchemy models

3.2.4 Supporting Components

- `/dependencies` - Dependency injections
- `/schemas` - Pydantic schemas
- `/uploads` - Uploaded documents storage

3.3 Detailed Component Documentation

3.3.1 Core Application

- `main.py`
  - FastAPI app initialization
  - CORS configuration
  - Router inclusions
  - Database connection setup

- `run.py`
  - Uvicorn server configuration
  - Hot-reloading setup

3.3.2 API Endpoints

- `user.py`
  - `/signup` - User registration
  - `/login` - User authentication
  - `/logout` - Session termination
  - `/role` - Role retrieval
- `admin.py`
  - `/users` - User management
  - `/documents` - Document operations

3.3.3 Core Services

- `rag_api.py`
  - Document processing pipeline
  - Vector store management
  - Gemini API integration
  - `/rag` endpoint implementation
- `security.py`
  - Password hashing/verification
  - Cryptographic utilities

3.3.4 Database Layer

- `database.py`
  - Session factory
  - Engine configuration
  - Connection management
- `models.py`
  - User model
  - Document model

  - Role definitions
- `crud.py`
  - User operations
  - Document operations

4 Frontend Documentation

4.1 Core Structure

4.1.1 Main Files

- `App.js` - Root application component

4.1.2 Directory Structure

- `/assets` - Static assets (images, styles)
- `/constants` - Configuration constants
- `/contexts` - React contexts
- `/views` - Page components
- `/components` - Reusable UI components

4.2 Component Documentation

4.2.1 Contexts

- `chatContext.js`
  - Chat state management
  - Message history
  - Current conversation
  - API interaction

4.2.2 Views (Pages)

- `LandingScreen.js` - Welcome/entry point
- `Login.js/Signup.js` - Authentication
- `Dashboard.js` - Admin overview
- `ChatPage.js` - Main chat interface
- `FileManagementPage.js` - Document admin
- `RolesPage.js` - Role management

- `UsersPage.js` - User administration

4.2.3 UI Components

- **Buttons & Inputs**
  - `BtnStyleOne/Two.js` - Primary/secondary buttons
  - `InputField.js` - Form input
  - `FilterBtn.js` - Filter toggle
  - `DropDownBtn.js` - Dropdown menu
- **File Management**
  - `DragDrop.js` - File upload
  - `FileCard/Grid/Table.js` - Document views
  - `AddedFile.js` - Upload queue item
- **Chat Interface**
  - `ChatHistory.js` - Conversation list
  - `ModelResponse.js` - AI messages
  - `UserMessage.js` - User messages
  - `ModelTyping.js` - Loading indicator
- **Data Visualization**
  - `BarGraph.js` - Bar charts
  - `LineGraph.js` - Line charts
- **Modals & Dialogs**
  - `DeletePopUp.js` - Deletion confirmation
  - `EditRoleModal.js` - Role assignment
  - `Logout.js` - Logout confirmation
  - `NotificationPanel.js` - Admin alerts
- **Branding**
  - `ClientLogo.js` - Client branding
  - `LogoSlogan.js` - App branding

5 System Workflows

5.1 User Authentication

1. Signup process
2. Login flow
3. Session management
4. Role-based access

5.2 Document Management

1. File upload processing
2. Metadata extraction
3. Thumbnail generation
4. Database storage

5.3 RAG Process

1. Document ingestion
2. Text processing
3. Vector embedding
4. Query handling
5. Response generation

6 Detailed Component Specifications

6.1 Backend Components

6.1.1 Core Application

- `main.py` - Main FastAPI application entry point
  - Sets up primary FastAPI instance with metadata
  - Configures CORS middleware
  - Includes routers for user and admin modules
  - Database connection setup
  - Root endpoint with welcome message
- `run.py` - Application runner

    - Starts FastAPI application using Uvicorn
    - Runs on `127.0.0.1:8000` with hot-reloading

6.1.2 User Management

- `user.py` - User API endpoints
    - Routes for signup, login, logout, and role retrieval
    - Session-based authentication
    - Password hashing and verification
- `crud.py` - Database operations
    - User creation and retrieval
    - SQLAlchemy ORM integration
- `security.py` - Security utilities
    - SHA-256 password hashing with salt
    - Password verification

6.1.3 Admin Features

- `admin.py` - Admin API endpoints
    - User management (view, modify roles, delete)
    - Document management (upload, list)
    - Admin authentication middleware

6.1.4 Database Configuration

- `database.py` - Database setup
    - Environment variable loading
    - SQLAlchemy engine and session factory
    - Table creation
    - Session dependency injection

6.1.5 RAG Functionality

- `rag_api.py` - Retrieval-Augmented Generation service
    - Separate FastAPI application (typically runs on port 8888)
    - Document processing pipeline:
        * PDF/text parsing using Docling library

        * Text cleaning and chunking
        * Embedding generation (sentence-transformers/all-MiniLM-L6-v2)
        * FAISS vector store management
    - Gemini API integration
    - Strict context-based answering
    - `/rag` endpoint for question answering

7 Frontend Components

7.1 Core Components

- `App.js` - Root application component
    - Sets up client-side routing
    - Wraps application in `BrowserRouter`
    - Defines various routes for public and private pages
    - Uses `useChatContext` for global chat state

7.2 Authentication Components

- `Login.js` - Login page
    - Manages email and password state
    - Implements login functionality
    - Includes error handling
- `Signup.js` - Registration page
    - Manages form data state
    - Implements signup functionality
    - Includes password confirmation

7.3 Main Application Components

7.3.1 Dashboard Components

- `Dashboard.js` - Admin dashboard
    - Displays various metrics and visualizations
    - Includes static data for queries and documents
    - Manages time frame filter state

  - Renders `BarGraph` and `LineGraph` components
- `BarGraph.js` - Data visualization
  - Uses Chart.js components
  - Customizes chart options and y-axis formatting
  - Retrieves data from JSON file
- `LineGraph.js` - Data visualization
  - Uses Chart.js components
  - Configures chart options
  - Retrieves data from JSON file

7.3.2 Chat Interface Components

- `ChatPage.js` - Main chat interface
  - Uses `useChatContext` for chat data
  - Renders `UserMessage` and `ModelResponse` components
  - Conditionally renders `ModelTyping` when loading
  - Includes text input for user messages
- `ChatHistory.js` - Conversation sidebar
  - Displays recent chats
  - Uses `useChatContext` for chat data
  - Includes toggle between regular and saved chats
- `UserMessage.js` - User chat messages
  - Implements message editing
  - Displays user avatar and message content
- `ModelResponse.js` - AI responses
  - Manages liked and copied states
  - Includes interaction buttons (copy, like, regenerate)
- `ModelTyping.js` - Loading indicator
  - Displays typing animation during AI response generation

7.3.3 Document Management Components

- `FileManagementPage.js` - Document management
  - Manages document list and view state
  - Fetches documents from backend
  - Implements sorting and view switching
  - Renders `FileTable` or `FileGrid` based on view
  - Includes `DropFileInput` for uploads
- `DragDrop.js` - File upload interface
  - Implements drag-and-drop functionality
  - Manages file lists and upload status
  - Includes metadata collection workflow
- `AddedFile.js` - Upload queue item
  - Displays file name, type, and size
  - Shows options dropdown menu
  - Uses file type icons from `imgConfig`
- `FileCard.js` - Grid view item
  - Displays file thumbnail and metadata
  - Manages dropdown and popup states
  - Conditionally renders `DeletePopup`
- `FileGrid.js` - Grid layout
  - Displays files in card-based layout
  - Responsive grid design
- `FileTable.js` - Table view
  - Displays files in tabular format
  - Includes sorting functionality
- `DeletePopUp.js` - Deletion confirmation
  - Modal dialog for delete confirmation
  - Uses `BtnStyle1` and `BtnStyle2` components

7.3.4 User Management Components

- `UsersPage.js` - User administration
  - Manages users and notifications
  - Implements filtering by user status
  - Includes role assignment functionality
  - Renders `UsersTable` component
- `UsersTable.js` - User data table
  - Fetches users from backend
  - Implements inline role editing
  - Handles user activation/deletion
  - Includes search functionality
- `NotificationPanel.js` - Admin notifications
  - Displays pending account approvals
  - Manages notification and user lists
  - Includes approval workflow with role selection
- `EditRoleModal.js` - Role assignment
  - Modal dialog for role changes
  - Dropdown for role selection
  - Confirmation/cancel actions

7.3.5 Role Management Components

- `RolesPage.js` - Role administration
  - Manages roles and privileges
  - Includes modal for new role creation
  - Displays role table with permissions
  - Handles role updates and deletions
- `RoleTable.js` - Role data display
  - Tabular display of role information
  - Shows assigned permissions
  - Includes edit/delete actions
- `CreateRoleModal.js` - New role creation

  - Form for new role definition
  - Permission selection interface
  - Validation and submission handling

7.3.6 UI Utility Components

- `BtnStyleOne.js` - Primary button
  - Solid color styling
  - Consistent hover/focus states
- `BtnStyleTwo.js` - Secondary button
  - Outlined styling
  - Complementary to primary button
- `InputField.js` - Form input
  - Labeled input field
  - Supports various input types
  - Validation states
- `SearchBar.js` - Search interface
  - Live search functionality
  - Clean, minimalist design
- `DropDownBtn.js` - Dropdown menu
  - Toggleable dropdown
  - Click-outside detection
  - Customizable menu items
- `FilterBtn.js` - Filter toggle
  - Active/inactive states
  - Customizable filter options
- `ToggleView.js` - View switcher
  - Toggles between grid and table views
  - Visual indicator of current view

7.3.7 Branding Components

- `ClientLogo.js` - Client branding
  - Displays client logo and name
  - Consistent placement across views
- `LogoSlogan.js` - Application branding
  - Displays app logo and tagline
  - Used in auth screens and landing page
- `SplashScreen.js` - Introductory screen
  - Animated typing effect
  - Automatic navigation after delay
  - Uses `framer-motion` for animations

8 Conclusion

This documentation provides a comprehensive overview of the RDLab system architecture, components, and workflows.