Multi-Agent RAG System - Frontend Documentation

Table of Contents

Overview

The frontend for the Multi-Agent RAG System is built using Next.js, React, and TailwindCSS. It provides an intuitive user interface for interacting with the backend's document analysis and summarization capabilities. The application features a clean, two-column layout with input controls on the left and output displays on the right.

Key Features

- Query input and crew type selection
- Real-time event logging
- Formatted result display with Markdown support
- Job status tracking and notifications
- Responsive design using TailwindCSS

Setup and Installation

Prerequisites

- Node.js 16.x or higher
- npm or yarn package manager
- Backend server running (see Backend Documentation)

Installation Steps

1. Clone the repository:

```
git clone https://github.com/yourusername/multi-agent-rag-frontend.git

cd multi-agent-rag-frontend
```

2. Install dependencies:

```
npm install

# or

yarn install
```

3. Create a .env.local file with the following variables:

```
NEXT_PUBLIC_API_URL=http://localhost:3001
```

4. Start the development server:

```
npm run dev

# or

yarn dev
```

5. Open http://localhost:3000 in your browser to access the application.

Project Structure

The project follows the standard Next.js structure with additional organization for components and hooks:

```
├── app/                # Next.js App Router files
|   ├── globals.css     # Global styles
|   ├── layout.tsx      # Root layout component
|   └── page.tsx        # Home page component
├── components/         # React components
|   ├── EventLog.tsx    # Event logging display
|   ├── FinalOutput.tsx # Final result display
|   ├── Header.tsx      # Application header
|   └── InputSection.tsx  # Query input and crew selection
```

```
 ├── hooks/              # Custom React hooks
|   └── UseCrewJob.tsx     # Job management hook
 ├── public/             # Static assets
└── package.json         # Project dependencies
```

Component Architecture

Layout (layout.tsx)

The root layout provides the application shell, including:

- Global font (Inter)
- Header component
- Toast notifications

Home Page (page.tsx)

The main page component with a two-column layout:

- Left column for input controls (query and crew type)
- Right column for job controls, results, and event logs

Input Section (InputSection.tsx)

Manages user input with:

- Crew type selection dropdown (Analysis/Summary)
- Query input field with validation
- Input confirmation display

Event Log (EventLog.tsx)

Displays real-time job events with:

- Timestamp formatting
- Event data rendering with Markdown support
- Special handling for document summary arrays

Final Output (FinalOutput.tsx)

Shows the final job result with:

- User query display
- Formatted result using Markdown
- Scrollable container for large outputs

Header (Header.tsx)

Simple application header with title

State Management

UseCrewJob Hook (UseCrewJob.tsx)

The primary state management mechanism using React hooks:

State Variables

- running: Boolean flag indicating if a job is in progress
- user_query: String containing the user's input query
- crew_type: String specifying the crew type ("analysis" or "summary")
- finalResult: Object containing the final job result
- events: Array of event objects from the job
- currentJobId: String ID of the current running job

Functions

- setQuery: Updates the user query
- setCrewType: Changes the selected crew type
- startJob: Initiates a new job by calling the backend API

Side Effects

- Auto-polling for job status updates
- Job status management
- Error handling and notifications

API Integration

The frontend communicates with the backend through a RESTful API:

API Endpoints Used

Start Job

- **URL**: ${API_URL}/api/crew
- **Method**: POST
- **Body**:
- { "user_query": "User's query", "crew_type": "analysis" // or "summary"}
- **Response**:
- { "job_id": "uuid-string"}

Get Job Status

- **URL**: ${API_URL}/api/crew/${jobId}
- **Method**: GET
- **Response**:
- { "status": "STARTED|COMPLETE|ERROR", "result": { "user_query": "...", "result": "..." }, "events": [{ "timestamp": "...", "data": "..." }]}

Polling Logic

The system polls for job status updates every 10 seconds while a job is running. Polling automatically stops when:

- Job completes successfully
- An error occurs
- Component unmounts

UI/UX Design

Layout

- Two-column design for clear separation of input and output
- Responsive sizing with TailwindCSS width classes

Color Scheme

- White background for clean, readable interface
- Green action buttons for starting jobs and submitting queries
- Gray containers for content blocks

- Black header for visual contrast

Typography

- Inter font for optimal readability
- Hierarchical text sizes for clear information structure
- Bold headings to separate content sections

Interactive Elements

- Clear button states (enabled/disabled)
- Toast notifications for job status updates
- Scrollable containers for large content

Extending the Frontend

Adding New Features

Supporting New Crew Types

1. Add new options to the crew type dropdown in InputSection.tsx
2. Update the useCrewJob hook to handle the new crew types
3. Add any specialized rendering for new crew type outputs

Enhanced Input Methods

1. Create new input components in the components/ directory
2. Add the components to the left column in page.tsx
3. Update the state in useCrewJob to handle the new input data

Additional Output Visualizations

1. Create new visualization components in the components/ directory
2. Add the components to the right column in page.tsx
3. Update the state handling in useCrewJob to provide the necessary data

Customization Options

Styling

The project uses TailwindCSS for styling. To modify the visual design:

1. Update class names in component files for minor changes
2. Modify globals.css for global style changes
3. Customize the TailwindCSS configuration in tailwind.config.js for theme changes

Configuration

To change API endpoints or other configuration:

1. Update the .env.local file with new environment variables
2. Access variables in code using process.env.NEXT_PUBLIC_*

This documentation covers the core functionality of the Multi-Agent RAG System frontend. For additional details or specific implementation questions, please refer to the source code or contact the system administrator.