**Query Management System Frontend Documentation**

**Overview**

The frontend of the Employee Query Management System is a single-page application (SPA) built with React. It uses Material-UI for the user interface components and Axios for handling API requests. The application is designed to provide a seamless user experience for managing employee queries, including viewing, updating, and resolving queries.

**Project Structure**

The frontend project is organized into several key directories and files:

* **src/**: Contains the main source code of the application.
  + **components/**: Reusable UI components.
  + **pages/**: Page components representing different views in the application.
  + **api/**: API request functions.
  + **App.js**: The main application component that sets up routing.
  + **index.js**: Entry point of the application.

**Main Components**

**1. App Component**

The App component is the root component of the application. It sets up routing using react-router-dom to navigate between different pages of the application. It also provides the global theming using Material-UI's ThemeProvider.

**2. QueryInbox Component**

The QueryInbox component is the main interface for viewing and managing employee queries. It includes:

* **Tabs**: Allows switching between open and resolved queries.
* **Sidebar**: Provides navigation links to different parts of the application.
* **Query List and Detail View**: Displays a list of queries and the details of a selected query.

**3. Sidebar Component**

The Sidebar component contains navigation links to different pages of the application. It ensures easy access to various sections such as the query inbox and settings.

**4. QueryList Component**

The QueryList component displays a list of employee queries. Each query item in the list includes basic information such as the query title and status. Users can click on a query item to view its details.

**5. QueryDetail Component**

The QueryDetail component shows detailed information about a selected query. It allows users to view the full query content, update the query status, and add comments or notes.

**API Integration**

The frontend communicates with the backend using Axios. API calls are centralized in the api directory, which includes functions for:

* Fetching all queries.
* Fetching query details by ID.
* Updating query status.

This separation of concerns ensures that all API interactions are handled in a consistent manner.

**State Management**

The application uses React's built-in state management features, such as useState and useEffect, to manage component state and side effects. This approach keeps the application simple and easy to maintain.

**Routing**

Routing is handled by react-router-dom. The main routes are defined in the App component, which include:

* /: The home page, which typically redirects to the query inbox.
* /queries: The query inbox, displaying the list of queries.
* /queries/:id: The query detail view, showing details of a specific query.

These routes allow users to navigate between different views in the application.

**Theming**

Global theming is applied using Material-UI's ThemeProvider and createTheme. The theme configuration includes:

* **Palette**: Defines primary and secondary colors.
* **Typography**: Sets default font styles and sizes.
* **Spacing**: Configures default spacing for margins and padding.

The theme is provided to the entire application, ensuring a consistent look and feel.

**Styling**

Styling is primarily handled using Material-UI's styling solution. Components are styled using the makeStyles and withStyles hooks provided by Material-UI. This approach allows for:

* **Scoped CSS**: Styles are scoped to components, preventing global style conflicts.
* **Dynamic Styles**: Styles can be dynamically adjusted based on component props and state.

**Error Handling**

Error handling is implemented at both the component and API levels. Common error handling strategies include:

* **API Errors**: API request functions catch and handle errors, providing feedback to the user.
* **Component Errors**: Components include error boundaries to catch and display errors that occur during rendering.

**Responsive Design**

The application is designed to be responsive, ensuring a good user experience across different devices and screen sizes. Material-UI's responsive utilities and grid system are used to achieve this.

**Build and Deployment**

**Local Deployment**

For local development, run the development server using:

1. **From the project root where you see both query\_management and query\_management\_frontend.**

cd query\_management\_frontend

cd queries

1. **Install dependancies**

npm install

1. **Run the frontend server**

npm start

**Docker Deployment**

For deploying the application using Docker, follow these steps:

1. **Make sure you are in the project root.**

Where you see both query\_management and querymanagement\_frontend folders.

cd query\_management\_frontend

cd queries

1. **Build the Docker Image**

Docker-compose build

1. **Run the Docker Container**

Docker-compose up

The frontend is built using a Dockerfile that performs the following steps:

1. **Install Dependencies**: Installs the necessary npm packages.
2. **Build the Application**: Runs the build script to create a production build of the application.
3. **Serve the Application**: Uses the serve package to serve the built application.

This Dockerfile ensures that the frontend can be easily built and deployed in a consistent environment.

**Summary**

This documentation provides an extensive overview of the frontend of the Employee Query Management System. It covers the project structure, main components, API integration, state management, routing, theming, styling, error handling, responsive design, and build and deployment processes. The application is designed to be modular, maintainable, and user-friendly, leveraging modern web development practices and tools.