Below is the **Frontend Documentation** for the Employee Management System. It provides an overview of the React-based UI, its folder structure, main components, routing, and various pages for managing Companies, Departments, and Employees. This documentation is intended to help developers or reviewers understand how the frontend is organized and how to work with it.

---

1. Overview

- **Framework**: React (with React Router, Bootstrap, and TypeScript).
- **Purpose**: Provide a user-friendly interface for managing companies, departments, and employees, handling user authentication (JWT), role-based views (Admin, Manager, Employee), and additional features like a chatbot or analytics.
- **Key Features**:
    1. **Login & Authentication**: Using JWT tokens from the backend.
    2. **Role-based Navigation**: Admin, Manager, Employee each sees different menu items.
    3. **CRUD Pages** for Companies, Departments, and Employees.
    4. **Optional Chatbot** integration (floating button).
    5. **Search, filtering, modals** for better UX.
    6. **Dark Mode / Theming** or advanced UI improvements (if implemented).

---

2. Folder Structure

css

Copy

employee_management_frontend/

├— public/

├— src/

│ ├— assets/

│ ├— components/

```
│   │   └── Sidebar.tsx
│   ├── context/
│   ├── pages/
│   │   ├── CompaniesList.tsx
│   │   ├── CompanyForm.tsx
│   │   ├── Dashboard.tsx
│   │   ├── DepartmentForm.tsx
│   │   ├── Departments.tsx
│   │   ├── DepartmentsList.tsx
│   │   ├── EmployeesList.tsx
│   │   ├── Login.tsx
│   │   ├── NotFound.tsx
│   │   ├── Profile.tsx
│   ├── services/
│   │   └── api.ts
│   ├── App.tsx
│   ├── index.css
│   ├── main.tsx
│   └── routes.tsx
├── package.json
└── ...
```

Key Directories

- **src/components/**: Reusable UI components (e.g., Sidebar.tsx).
- **src/pages/**: Each page in the app, such as Login, Dashboard, CompaniesList, etc.

- **src/context/**: Could hold context providers (ThemeContext, NotificationContext, etc.).
- **src/services/**: Usually for API helper functions (api.ts).
- **src/routes.tsx**: React Router configuration.

---

3. Main Files

3.1. main.tsx

tsx

Copy

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App'
import 'bootstrap/dist/css/bootstrap.min.css';


createRoot(document.getElementById('root')!).render(
 <StrictMode>
  <App />
 </StrictMode>,
)
```

- **Entry point** of the React app.
- Renders the <App /> component, imports global styles (index.css, bootstrap.min.css).

3.2. App.tsx

tsx

Copy

```
import React from "react";
```

```
import AppRoutes from "./routes";


const App: React.FC = () => {
 return (
  <div>
   <AppRoutes />
  </div>
 );
};


export default App;
```

- Simple container that includes <AppRoutes />.
- In some setups, context providers (ThemeProvider, NotificationProvider, etc.) may wrap <AppRoutes />.

3.3. routes.tsx

tsx

Copy

```
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";
import Login from "./pages/Login";
import Dashboard from "./pages/Dashboard";
import EmployeesList from "./pages/EmployeesList";
import CompaniesList from "./pages/CompaniesList";
import CompanyForm from "./pages/CompanyForm";
import Departments from "./pages/Departments";
import DepartmentForm from "./pages/DepartmentForm";
import Profile from "./pages/Profile";
```

```
const PrivateRoute = ({ element }: { element: JSX.Element }) => {
  const token = localStorage.getItem("accessToken");
  return token ? element : <Navigate to="/" />;
};


const AppRoutes = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/dashboard" element={<PrivateRoute element={<Dashboard />} />} />
        <Route path="/profile" element={<PrivateRoute element={<Profile />} />} />
        <Route path="/employees" element={<PrivateRoute element={<EmployeesList />} />}
/>
        <Route path="/companies" element={<PrivateRoute element={<CompaniesList />} />}
/>
        <Route path="/companies/add" element={<PrivateRoute element={<CompanyForm />}
/>} />
        <Route path="/companies/edit/:id" element={<PrivateRoute element={<CompanyForm
/>} />} />
        <Route path="/departments" element={<PrivateRoute element={<Departments />} />}
/>
        <Route path="/departments/add" element={<PrivateRoute
element={<DepartmentForm />} />} />
        <Route path="/departments/edit/:id" element={<PrivateRoute
element={<DepartmentForm />} />} />
      </Routes>
```

```
  </Router>
 );
};

export default AppRoutes;
```

- **PrivateRoute**: checks if accessToken is in localStorage; otherwise redirects to "/" (the Login page).
- Each path leads to a specific page (e.g., "/employees" → <EmployeesList />).

---

4. Pages

4.1. Login.tsx

tsx

Copy

```
const Login: React.FC = () => {
 // local states: loginField, password, error
 // onSubmit => POST /api/token/
 // if success => store tokens in localStorage, fetch user info => redirect
 ...
};
```

- Sends credentials to the backend's /api/token/.
- Stores accessToken and refreshToken.
- Retrieves user info from /api/user/ to store role in localStorage.

4.2. Dashboard.tsx

tsx

Copy

```
const Dashboard: React.FC = () => {
```

```
 // Example: fetch employees or show summary
 // Might have a table or analytics
 ...
};
```

- Could display an overview of employees, stats, or a default landing for Admin/Manager.

4.3. EmployeesList.tsx

tsx

Copy

```
const EmployeesList: React.FC = () => {
 // fetch employees from /api/employees/
 // if manager => sees employees in their company
 // if admin => sees all employees
 // table with Delete / Edit
 ...
};
```

- Possibly includes a Spinner while loading.
- Allows manager or admin to remove employees, etc.

4.4. CompaniesList.tsx

tsx

Copy

```
const CompaniesList: React.FC = () => {
 // fetch companies from /api/companies/
 // create / edit / delete if admin or manager
 ...
};
```

4.5. Departments.tsx or DepartmentsList.tsx

tsx

Copy

```
const DepartmentsList: React.FC = () => {
 // fetch from /api/departments/
 // add department (manager/admin)
 // filter by company
 ...
};
```

4.6. DepartmentForm.tsx / CompanyForm.tsx

- Typically used for adding or editing a department or company.
- Submits to /api/departments/ or /api/companies/.

4.7. Profile.tsx

tsx

Copy

```
const Profile: React.FC = () => {
 // fetch /api/user/ for user details
 // if employee, show employee data
 ...
};
```

- Simple page to display personal info.

4.8. NotFound.tsx

- Optional 404 fallback if a route doesn't match.

---

5. Components

5.1. Sidebar.tsx

tsx

Copy

```
const Sidebar: React.FC = () => {
  // shows links to employees, companies, departments
  // might have a "Logout" button
  ...
};
```

- Renders different links based on user’s role stored in localStorage.getItem("userRole").
- Could also have a theme toggle or other features.

5.2. SearchBar.tsx, Notification.tsx, etc.

- Possibly used for searching lists or displaying global notifications.

---

6. Services (services/api.ts)

- Could hold a base fetch or axios instance:

ts

Copy

```
import axios from 'axios';

export const api = axios.create({
  baseURL: 'http://127.0.0.1:8000/',
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) {
```

```
  config.headers.Authorization = `Bearer ${token}`;
 }
 return config;
});
```

- Then used in pages to fetch data, e.g. api.get('/api/employees/').

---

7. State Management / Context (Optional)

- If using React Context or Redux, might see files in src/context/ or src/store/.
- Example: ThemeContext.tsx, NotificationContext.tsx for dark mode or toasts.

---

8. Styling & Theming

- index.css: global styles.
- Bootstrap: imported in main.tsx.
- Additional custom classes or theming can appear in index.css or dedicated .css files per component/page.

---

9. Running & Building

1. **Install dependencies**:

bash

Copy

```
npm install
```

2. **Run in development**:

bash

Copy

```
npm run dev
```

3. **Production build**:

bash

Copy

npm run build

4. **Deployment**:
    - Usually serve the dist folder or integrate with a hosting solution.

---

10. Example Workflow

1. **Login** at /:
    - Enter credentials → calls /api/token/ → on success, store accessToken.
    - Fetch user role from /api/user/.
    - Redirect to /dashboard.
2. **Navigation**:
    - Sidebar or top menu → e.g. “Companies” → loads /companies (CompaniesList).
3. **Create / Edit**:
    - Manager or admin → “Add Company” → /companies/add.
    - Submit form → POST /api/companies/.
4. **Employee Management**:
    - Admin or manager sees employees → can add or remove employees in their company.
    - Employee sees only their own record.
5. **Logout**:
    - Clear localStorage, redirect to /.

---

11. Notes on Security & Permissions

- The frontend checks for a valid accessToken before allowing access to private routes (PrivateRoute).

- Role-based logic is mostly done by the backend. The frontend simply hides or shows links accordingly.

---

12. Possible Enhancements

1. **Dark Mode** or advanced styling.
2. **Search and filtering** on each list page (Employees, Departments, etc.).
3. **Notifications** (global toasts) for success/error.
4. **Chatbot** integration: a floating button that opens a panel to query the chat_with_bot endpoint.
5. **Redux or Context** for global state if the app grows larger.

---

13. Conclusion

The frontend is built with React, React Router, and TypeScript, providing a clean structure for:

- **Authentication** (Login page).
- **Protected routes** (PrivateRoute).
- **CRUD pages** for the main entities (Companies, Departments, Employees).
- **Role-based** display logic (Admin sees all, Manager sees only their company, Employee sees personal data).
- **Optional advanced features** like theming, notifications, chatbot.

By following the routes, pages, and component structure, developers can easily extend or maintain the application, integrate additional features, or customize the UI for different roles and user experiences.