💻 Complete Code Explanation - Line by Line

Employee Management System - All Files Explained

📋 Table of Contents

FRONTEND FILES

---

1. package.json

Purpose: Defines project metadata, dependencies, and scripts.

```json
{
  "name": "employee-management-react",
  "private": true,
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.20.0",
    "@mui/material": "^5.14.0",
    "@mui/icons-material": "^5.14.0",
    "@emotion/react": "^11.11.0",
    "@emotion/styled": "^11.11.0",
    "axios": "^1.6.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.0",
    "@types/react-dom": "^18.2.0",
    "@vitejs/plugin-react": "^4.2.0",
    "typescript": "^5.2.0",
    "vite": "^5.0.0"
  }
}
```

Line-by-line Explanation:

Line 2: `"name": "employee-management-react"`

- Project এর নাম। npm package name হিসেবে ব্যবহার হয়।

Line 3: `"private": true`

- এটি নিশ্চিত করে যে এই package accidentally npm registry তে publish হবে না।

Line 4: `"version": "1.0.0"`

- Project এর version number। Semantic versioning follow করে (MAJOR.MINOR.PATCH)।

Line 5: `"type": "module"`

- ES modules use করার জন্য (import/export syntax)।

Line 6-10: `"scripts"` Scripts যা npm run দিয়ে execute করা যায়:

- `"dev": "vite"` - Development server start করে (hot reload সহ)
- `"build": "tsc && vite build"` - TypeScript compile করে, তারপর production build তৈরি করে
- `"preview": "vite preview"` - Production build locally preview করে

Line 11-19: `"dependencies"` Production এ দরকার এমন packages:

- `react`: Core React library
- `react-dom`: React কে browser DOM এর সাথে connect করে
- `react-router-dom`: Client-side routing এর জন্য
- `@mui/material`: Material-UI components
- `@mui/icons-material`: Material-UI icons
- `@emotion/react` & `@emotion/styled`: Material-UI এর জন্য CSS-in-JS library
- `axios`: HTTP requests এর জন্য

Line 20-26: `"devDependencies"` শুধু development এ দরকার:

- `@types/react` & `@types/react-dom`: TypeScript type definitions
- `@vitejs/plugin-react`: Vite plugin for React support
- `typescript`: TypeScript compiler
- `vite`: Build tool and dev server

---

2. vite.config.ts

Purpose: Vite build tool এর configuration। Development server এবং proxy setup।

```
typescript
```

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      '/sem': {
        target: 'http://localhost:52773',
        changeOrigin: true,
        secure: false,
      },
    },
  },
})
```

Line-by-line Explanation:

Line 1: **`import { defineConfig } from 'vite'`**

- Vite থেকে `defineConfig` function import করছি।
- এই function TypeScript auto-completion provide করে।

Line 2: **`import react from '@vitejs/plugin-react'`**

- React plugin import করছি।
- এটি JSX/TSX files compile করে এবং Fast Refresh enable করে।

Line 4: **`// https://vitejs.dev/config/`**

- Comment: Documentation link।

Line 5: **`export default defineConfig({`**

- Configuration object export করছি।
- `defineConfig` wrap করে type safety পাওয়ার জন্য।

Line 6: **`plugins: [react()],`**

- Vite plugins array।
- `react()` plugin activate করছি।

- এটি React components compile করে এবং hot reload enable করে।

Line 7-16: **`server:`** **Block** Development server configuration:

Line 8: **`port: 5173,`**

- Dev server port number।
- Frontend এই port এ run হবে: `http://localhost:5173`

Line 9-15: **`proxy:`** **Block** API requests proxy করার configuration:

Line 10: **`'/sem': {`**

- যেকোনো request যা `/sem` দিয়ে শুরু হয়, সেটা proxy হবে।
- Example: `http://localhost:5173/sem/employees` → proxy → IRIS backend

Line 11: **`target: 'http://localhost:52773',`**

- Proxy destination: IRIS backend এর URL।
- `/sem/employees` → `http://localhost:52773/sem/employees`

Line 12: **`changeOrigin: true,`**

- HTTP request এর `Origin` header change করে target URL এর সাথে match করায়।
- CORS issues avoid করার জন্য দরকার।

Line 13: **`secure: false,`**

- HTTPS certificate verification disable করে।
- Development এ self-signed certificates এর জন্য।

Why Proxy is Needed:

```
Without Proxy:
Frontend (localhost:5173) → Backend (localhost:52773)
❌ CORS Error! (Different origins)

With Proxy:
Frontend (localhost:5173) → Proxy (localhost:5173) → Backend (localhost:52773)
✅ Same origin! No CORS error.
```

---

3. tsconfig.json

Purpose: TypeScript compiler এর configuration।

```json
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

Line-by-line Explanation:

Line 2: `"compilerOptions": {`

- TypeScript compiler এর options।

Line 3: `"target": "ES2020"`

- TypeScript code কোন JavaScript version এ compile হবে।
- ES2020 = Modern JavaScript features (async/await, optional chaining, etc.)

Line 4: `"useDefineForClassFields": true`

- Class fields এর জন্য ECMAScript standard behavior use করে।

Line 5: `"lib": ["ES2020", "DOM", "DOM.Iterable"]`

- কোন built-in types available থাকবে:
  - **ES2020**: Modern JavaScript APIs
  - **DOM**: Browser APIs (document, window, etc.)
  - **DOM.Iterable**: Array methods on DOM collections

Line 6: `"module": "ESNext"`

- Module system: Latest ES modules (import/export)

Line 7: `"skipLibCheck": true`

- Third-party library এর type checking skip করে (faster compilation)

Line 9-14: Bundler Mode Vite bundler এর জন্য specific settings:

Line 10: `"moduleResolution": "bundler"`

- Modern bundlers (Vite, webpack) এর জন্য module resolution strategy।

Line 11: `"allowImportingTsExtensions": true`

- `.ts` / `.tsx` extension সহ import করা যাবে।

Line 12: `"resolveJsonModule": true`

- JSON files import করা যাবে।

Line 13: `"isolatedModules": true`

- প্রতিটা file আলাদাভাবে compile হবে (Vite এর জন্য দরকার)।

Line 14: `"noEmit": true`

- TypeScript JavaScript file generate করবে না (Vite করবে)।

Line 15: `"jsx": "react-jsx"`

- JSX কিভাবে transform হবে।
- React 17+ এর new JSX transform use করে।

Line 17-20: Linting Options Code quality check এর জন্য:

Line 18: `"strict": true`

- সব strict type-checking options enable।
- Prevents: `any` types, null checks, etc.

Line 19: `"noUnusedLocals": true`

- Unused variables থাকলে error দেখাবে।

Line 20: `"noUnusedParameters": true`

- Unused function parameters থাকলে error।

Line 21: `"noFallthroughCasesInSwitch": true`

- Switch case এ `break` missing থাকলে error।

Line 23: `"include": ["src"]`

- কোন directory TypeScript compile করবে।
- শুধু `src/` folder।

Line 24: `"references": [{ "path": "./tsconfig.node.json" }]`

- Node.js scripts এর জন্য আলাদা TypeScript config reference।

---

4. index.html

Purpose: Application এর entry HTML file। React app এখানে mount হয়।

```html
```

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>簡易社員管理システム</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Line-by-line Explanation:

Line 1: `<!DOCTYPE html>`

- HTML5 document type declaration।
- Browser কে বলে এটি modern HTML document।

Line 2: `<html lang="ja">`

- HTML element with language attribute।
- `lang="ja"` = Japanese language (screen readers এর জন্য)

Line 4: `<meta charset="UTF-8" />`

- Character encoding set করছি UTF-8 তে।
- Japanese characters properly display করার জন্য দরকার।

Line 5: `<link rel="icon" type="image/svg+xml" href="/vite.svg" />`

- Favicon (browser tab এর icon)।
- Vite এর default icon।

Line 6: `<meta name="viewport" content="width=device-width, initial-scale=1.0" />`

- Mobile responsive এর জন্য viewport settings।
- **width=device-width**: Screen width অনুযায়ী width set হবে
- **initial-scale=1.0**: Zoom level 100%

Line 7: `<title>簡易社員管理システム</title>`

- Browser tab এ দেখা যায় এই title।
- "Kan'i Shain Kanri Shisutemu" = Simple Employee Management System

Line 10: `<div id="root"></div>`

- **সবচেয়ে গুরুত্বপূর্ণ element!**
- React application এই `div` এর মধ্যে mount/render হয়।
- Empty থাকে initially, JavaScript load হলে React components দিয়ে fill হয়।

Line 11: `<script type="module" src="/src/main.tsx"></script>`

- Main JavaScript/TypeScript file load করছি।
- **type="module"**: ES6 modules support করবে
- **src="/src/main.tsx"**: Entry point file
- Vite এই file process করে এবং সব dependencies bundle করে

How it Works:

```
1. Browser index.html load করে
   ↓
2. <div id="root"></div> খালি থাকে
   ↓
3. main.tsx script execute হয়
   ↓
4. React app root div এর মধ্যে render হয়
   ↓
5. User interface দেখতে পায়
```

---

5. src/main.tsx

Purpose: Application এর entry point। React app কে DOM এ mount করে।

```
typescript
```

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import './index.css'

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

Line-by-line Explanation:

Line 1: `import React from 'react'`

- React library import করছি।
- JSX use করার জন্য দরকার (modern React এ optional)।

Line 2: `import ReactDOM from 'react-dom/client'`

- ReactDOM library import করছি।
- React 18+ এর new rendering API।
- React components কে browser DOM এ render করার জন্য।

Line 3: `import App from './App.tsx'`

- Main App component import করছি।
- './App.tsx' = same directory তে App.tsx file আছে।

Line 4: `import './index.css'`

- Global CSS file import করছি।
- এই CSS সব components এ apply হবে।

Line 6: `ReactDOM.createRoot(document.getElementById('root')!)` এই line এ অনেক কিছু হচ্ছে:

- `document.getElementById('root')`:
    - HTML file থেকে `id="root"` এর element খুঁজছি
    - Returns: `<div id="root"></div>`
- `!` **(Non-null assertion operator)**:
    - TypeScript কে বলছি element definitely পাওয়া যাবে

    - Without `!`: TypeScript error দেবে (element null হতে পারে)

- **`ReactDOM.createRoot(...)`**:
    - React 18 এর new concurrent rendering API
    - Root element তৈরি করছি যেখানে React app render হবে

Line 6-9: `.render(...)` Root element এ React app render করছি:

Line 7: `<React.StrictMode>`

- Development mode এ extra checks enable করে:
    - Warns about deprecated APIs
    - Detects side effects
    - Checks for unsafe lifecycle methods
- **শুধু development এ run হয়, production build এ removed**

Line 8: `<App />`

- Main App component render করছি।
- এটাই application এর root component।

Execution Flow:

```
1. Browser loads index.html
  ↓
2. Loads main.tsx script
  ↓
3. Imports React, ReactDOM, App, CSS
  ↓
4. Finds <div id="root"></div>
  ↓
5. Creates React root at that div
  ↓
6. Renders <App /> component inside StrictMode
  ↓
7. App component এর সব child components render হয়
  ↓
8. User interface complete!
```

Why StrictMode?

```
typescript
```

```
// With StrictMode (Development):
<React.StrictMode>
  <App />
</React.StrictMode>
// ✅ Catches potential problems
// ✅ Warns about bad practices
// ✅ Helps write better code

// Without StrictMode:
<App />
// ❌ Problems may go unnoticed
// ❌ Harder to debug later
```

---

6. src/App.tsx

Purpose: Main routing component। Application এর সব routes define করা আছে।

```
typescript
```

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
import SignIn from './pages/SignIn'
import SignUp from './pages/SignUp'
import EmployeeList from './pages/EmployeeList'
import EmployeeDetail from './pages/EmployeeDetail'
import ProtectedRoute from './components/ProtectedRoute'

function App() {
  return (
    <BrowserRouter>
      <Routes>
        {/* Public Routes */}
        <Route path="/signin" element={<SignIn />} />
        <Route path="/signup" element={<SignUp />} />

        {/* Protected Routes */}
        <Route
          path="/employees"
          element={
            <ProtectedRoute>
              <EmployeeList />
            </ProtectedRoute>
          }
        />
        <Route
          path="/employees/:id"
          element={
            <ProtectedRoute>
              <EmployeeDetail />
            </ProtectedRoute>
          }
        />

        {/* Default Redirect */}
        <Route path="/" element={<Navigate to="/signin" replace />} />
      </Routes>
    </BrowserRouter>
  )
}

export default App
```

Line-by-line Explanation:

Line 1: `import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'` React Router থেকে routing components import করছি:

- **BrowserRouter**: URL based routing enable করে
- **Routes**: সব Route elements এর container
- **Route**: Individual route define করে
- **Navigate**: Programmatic redirect এর জন্য

Line 2-5: Page Components Import সব page components import করছি:

- **SignIn**: Login page
- **SignUp**: Registration page
- **EmployeeList**: Employee list with search/filter
- **EmployeeDetail**: Add/Edit/Delete employee form

Line 6: **`import ProtectedRoute from './components/ProtectedRoute'`**

- Authentication guard component
- Login না করলে protected pages access করতে পারবে না

Line 8: **`function App() {`**

- Main App component definition
- Functional component (not class component)

Line 10: **`<BrowserRouter>`**

- Routing enable করে application এ
- HTML5 history API use করে (clean URLs)
- Example URLs: `/signin`, `/employees` (no `#` in URL)

Line 11: **`<Routes>`**

- সব Route components এর wrapper
- React Router v6 এ Switch এর replacement

Line 13: Public Routes Section

Line 13: **`<Route path="/signin" element={<SignIn />} />`**

- **path="/signin"**: URL path যা match করবে
- **element={<SignIn />}**: এই path এ কোন component render হবে

- Public route: কেউই access করতে পারবে

Line 14: `<Route path="/signup" element={<SignUp />} />`

- Registration page route
- Public route: কেউই access করতে পারবে

Line 16: Protected Routes Section

Line 17-24: Employee List Route

```typescript
<Route
  path="/employees"
  element={
    <ProtectedRoute>
      <EmployeeList />
    </ProtectedRoute>
  }
/>
```

- **path="/employees"**: Employee list page URL
- **ProtectedRoute wrapper**: Authentication check করবে
- **EmployeeList**: Actual component যা render হবে
- Login না থাকলে → Redirect to `/signin`
- Login থাকলে → Show EmployeeList

Line 25-32: Employee Detail Route

```typescript
<Route
  path="/employees/:id"
  element={
    <ProtectedRoute>
      <EmployeeDetail />
    </ProtectedRoute>
  }
/>
```

- **path="/employees/:id"**: Dynamic route parameter
- `:id`: Variable part of URL

- Examples:
  - `/employees/new` → Add new employee (id = "new")
  - `/employees/42` → Edit employee with ID 42
- **ProtectedRoute**: Same authentication check
- **EmployeeDetail**: Handles both add and edit modes

Line 34: Default Redirect

Line 35: `<Route path="/" element={<Navigate to="/signin" replace />} />`

- **path="/"**: Root URL (http://localhost:5173/)
- **Navigate to="/signin"**: Redirect করবে signin page এ
- **replace**: Browser history replace করে (back button work করে)

How Routing Works:

```
User navigates to URL:
├─ /              → Redirects to /signin
├─ /signin        → Shows SignIn component (public)
├─ /signup        → Shows SignUp component (public)
├─ /employees     → Checks auth → Shows EmployeeList (protected)
├─ /employees/new → Checks auth → Shows EmployeeDetail in ADD mode (protected)
├─ /employees/42  → Checks auth → Shows EmployeeDetail in EDIT mode (protected)
└─ /other         → 404 (no route matches)
```

Protected Route Flow:

```
User tries to access /employees:
  ↓
ProtectedRoute checks localStorage
  ├─ isLoggedIn = true?
  │   ↓
  │  ✅ Render EmployeeList
  │
  └─ isLoggedIn = false?
     ↓
    ❌ Redirect to /signin
```

Line 40: `export default App`

- App component export করছি

- main.tsx file এ import করা হবে

---

7. src/types/index.ts

Purpose: TypeScript type definitions। Application এর সব data structures define করা।

typescript

```
// User Account Interface
export interface User {
  id?: number;
  Email: string;
  Password: string;
  Name: string;
}

// Employee Interface
export interface Employee {
  id?: number;
  EmployeeId: string;
  Name: string;
  KanaName?: string;
  Sex: number;  // 1 = Male, 2 = Female
  PostCode?: string;
  Address?: string;
  PhoneNumber?: string;
  Department?: string;
  RetireFlg: boolean;
  deleteFlg?: boolean;
  upDateTime?: string;
}

// API Response Types
export interface ApiResponse<T> {
  message: string;
  data?: T;
}

export interface EmployeeListResponse {
  employees: Employee[];
}

export interface AuthResponse {
  message: string;
  id?: number;
}
```

Line-by-line Explanation:

Line 1-6: User Interface

Line 2: `export interface User {`

- **export**: অন্য files এ import করা যাবে

- **interface**: TypeScript type definition
- **User**: Account/User data structure

Line 3: `id?: number;`

- **id**: Database এর auto-generated ID
- ?: Optional property (নাও থাকতে পারে)
- **number**: TypeScript type
- New user তৈরি করার সময় id থাকে না, database generate করে

Line 4: `Email: string;`

- **Email**: Required property (? নাই)
- User এর email address
- **string**: TypeScript type

Line 5: `Password: string;`

- User password
- Required field
- Production এ hashed হওয়া উচিত

Line 6: `Name: string;`

- User এর নাম
- Required field

Line 9-22: Employee Interface

Line 10: `export interface Employee {`

- Employee data structure definition

Line 11: `id?: number;`

- Database auto-generated ID
- Optional: New employee add করার সময় থাকে না

Line 12: `EmployeeId: string;`

- 5-digit employee ID (e.g., "12345")

- Required: Must be unique
- String type because leading zeros possible

Line 13: `Name: string;`

- Employee এর নাম (e.g., "山田太郎")
- Required field
- Japanese characters support

Line 14: `KanaName?: string;`

- Katakana name (e.g., "ヤマダタロウ")
- Optional: সব employee এর নাও থাকতে পারে

Line 15: `Sex: number;`

- Gender field
- **1 = Male (男性)**
- **2 = Female (女性)**
- Number type: easier for database storage and comparison

Line 16: `PostCode?: string;`

- Japanese postal code (e.g., "100-0001")
- Optional field
- String type: contains hyphen

Line 17: `Address?: string;`

- Full address (e.g., "東京都千代田区...")
- Optional field

Line 18: `PhoneNumber?: string;`

- Phone number (e.g., "090-1234-5678")
- Optional field
- String type: contains hyphens and may have leading zeros

Line 19: `Department?: string;`

- Department name (e.g., "営業部", "技術部")
- Optional field

Line 20: `RetireFlg: boolean;`

- Retirement flag
- **true**: Retired (退職済み)
- **false**: Currently employed (在職中)
- Required field

Line 21: `deleteFlg?: boolean;`

- Soft delete flag
- **true**: Deleted (but still in database)
- **false**: Active record
- Optional: Frontend may not always need this

Line 22: `upDateTime?: string;`

- Last update timestamp
- Optional: Frontend may not display this
- String format: "2024-12-25 14:30:45"

Line 25-28: Generic API Response Type

Line 26: `export interface ApiResponse<T> {`

- **Generic type**: `<T>` can be any type
- Reusable response structure

Line 27: `message: string;`

- Success/error message from backend
- Examples: "Employee created successfully", "Login failed"

Line 28: `data?: T;`

- **T**: Generic type parameter
- Optional data payload

- Type depends on API endpoint

Usage Example:

```typescript
// When fetching employee:
ApiResponse<Employee>
// data will be Employee type

// When fetching list:
ApiResponse<Employee[]>
// data will be Employee array
```

Line 31-33: Employee List Response

Line 32: `export interface EmployeeListResponse {`

- Specific response type for employee list API

Line 33: `employees: Employee[];`

- **Employee[]**: Array of Employee objects
- Required field
- Backend returns list of all employees

Line 36-39: Authentication Response

Line 37: `export interface AuthResponse {`

- Response type for signup/signin APIs

Line 38: `message: string;`

- Success or error message
- Examples:
  - "Authentication successful."
  - "Invalid password."
  - "Email already exists."

Line 39: `id?: number;`

- User ID after successful registration

- Optional: Only present on success

Why TypeScript Types?

```typescript
// Without Types (JavaScript):
function createEmployee(emp) {
  // emp.name? emp.Name? emp.employeeName?
  // Easy to make mistakes!
  api.post('/employee', emp)
}

// With Types (TypeScript):
function createEmployee(emp: Employee) {
  // ✅ IDE suggests: emp.Name, emp.EmployeeId, etc.
  // ✅ Compile error if wrong property
  // ✅ Type safety!
  api.post('/employee', emp)
}
```

Type Safety Example:

```typescript
```

```
// ✅ Correct:
const employee: Employee = {
  EmployeeId: "12345",
  Name: "山田太郎",
  Sex: 1,
  RetireFlg: false
};

// ❌ Compile Error: Missing required fields
const employee: Employee = {
  EmployeeId: "12345"
  // Error: Name is required!
  // Error: Sex is required!
};

// ❌ Compile Error: Wrong type
const employee: Employee = {
  EmployeeId: 12345,  // Error: Should be string!
  Name: "山田太郎",
  Sex: "Male",        // Error: Should be number!
  RetireFlg: "no"     // Error: Should be boolean!
};
```

---

8. src/utils/auth.ts

Purpose: Authentication helper functions। Login state management localStorage use করে।

```
typescript
```

```
// Check if user is authenticated
export const isAuthenticated = (): boolean => {
  const isLoggedIn = localStorage.getItem('isLoggedIn');
  return isLoggedIn === 'true';
};

// Save authentication data
export const setAuthData = (email: string): void => {
  localStorage.setItem('isLoggedIn', 'true');
  localStorage.setItem('userEmail', email);
};

// Get current user email
export const getUserEmail = (): string | null => {
  return localStorage.getItem('userEmail');
};

// Clear authentication data (logout)
export const clearAuthData = (): void => {
  localStorage.removeItem('isLoggedIn');
  localStorage.removeItem('userEmail');
};
```

Line-by-line Explanation:

Line 1-5: isAuthenticated Function

Line 2: `export const isAuthenticated = (): boolean => {`

- **export const**: Export করা arrow function
- **isAuthenticated**: Function name
- **(): boolean**:
  - Empty parameters `()`
  - Returns boolean type

Line 3: `const isLoggedIn = localStorage.getItem('isLoggedIn');`

- **localStorage.getItem()**: Browser storage থেকে value পড়ছি
- **'isLoggedIn'**: Key name
- **Returns**: string | null
  - If key exists → returns the value as string
  - If key doesn't exist → returns null

Line 4: `return isLoggedIn === 'true';`

- String comparison করছি
- **Returns**:
  - `true` if isLoggedIn === 'true'
  - `false` if isLoggedIn is null or any other value

Why string comparison?

```typescript
// localStorage only stores strings!
localStorage.setItem('key', true)   // Stored as "true" (string)
localStorage.getItem('key')         // Returns "true" (string)

// So we compare with string:
isLoggedIn === 'true'  // ✅ Correct
isLoggedIn === true    // ❌ Wrong! (comparing string with boolean)
```

Line 7-11: setAuthData Function

Line 8: `export const setAuthData = (email: string): void => {`

- **email: string**: Parameter type
- **: void**: Returns nothing

Line 9: `localStorage.setItem('isLoggedIn', 'true');`

- **localStorage.setItem()**: Browser storage এ value save করছি
- **'isLoggedIn'**: Key name
- **'true'**: Value (string)
- এটি mark করে user logged in

Line 10: `localStorage.setItem('userEmail', email);`

- User এর email save করছি
- Later use করা যাবে (display name, etc.)

Line 13-16: getUserEmail Function

Line 14: `export const getUserEmail = (): string | null => {`

- **: string | null**: Union type
  - Returns string if email exists
  - Returns null if not found

Line 15: `return localStorage.getItem('userEmail');`

- Directly return করছি localStorage থেকে
- No need for extra checks

Line 18-22: clearAuthData Function

Line 19: `export const clearAuthData = (): void => {`

- Logout করার জন্য function
- সব authentication data remove করে

Line 20: `localStorage.removeItem('isLoggedIn');`

- Login flag remove করছি
- After this, isAuthenticated() will return false

Line 21: `localStorage.removeItem('userEmail');`

- Email remove করছি
- Clean up সব data

localStorage Explained:

```typescript
```

```
// localStorage = Browser's permanent storage

// Save data:
localStorage.setItem('key', 'value')

// Get data:
localStorage.getItem('key')  // Returns: 'value'

// Remove data:
localStorage.removeItem('key')

// Clear all:
localStorage.clear()

// Properties:
// ✅ Data persists after page refresh
// ✅ Data persists after browser close
// ✅ Data specific to domain
// ❌ Only stores strings (must convert objects)
// ❌ Synchronous (blocks main thread)
// ❌ ~5-10MB storage limit
```

Usage Example in Application:

```
typescript
```

```
// After successful login:
setAuthData('user@example.com')
// localStorage now has:
// {
//   'isLoggedIn': 'true',
//   'userEmail': 'user@example.com'
// }

// Check if logged in:
if (isAuthenticated()) {
  // ✅ User is logged in
  // Show employee list
} else {
  // ❌ User not logged in
  // Redirect to signin
}

// Get current user:
const email = getUserEmail()
// Returns: 'user@example.com'

// Logout:
clearAuthData()
// localStorage now empty
// isAuthenticated() returns false
```

Security Note:

```
typescript
```

```
// ⚠️ Current Implementation (Not Production-Ready):
// - Stores login state in localStorage
// - No expiration time
// - No token validation
// - Anyone can manually set isLoggedIn = 'true'

// ✅ Production Implementation Should Have:
// - JWT tokens with expiration
// - Token refresh mechanism
// - Server-side validation
// - HttpOnly cookies (more secure)
// - CSRF protection

// Example with JWT:
export const setAuthData = (token: string): void => {
  localStorage.setItem('authToken', token);
  // Token contains: user info, expiry, signature
};

export const isAuthenticated = (): boolean => {
  const token = localStorage.getItem('authToken');
  if (!token) return false;

  // Decode and check expiry
  const decoded = decodeJWT(token);
  const isExpired = decoded.exp < Date.now() / 1000;

  return !isExpired;
};
```

I'll continue with the remaining files. Would you like me to proceed with the next files (api.ts, Layout.tsx, etc.)?