



## Complete Code Explanation - Part 2

### Remaining Frontend & Backend Files

---

#### 9. src/services/api.ts

**Purpose:** API client configuration | Backend এর সাথে HTTP communication এর জন্য।

typescript

```
import axios from 'axios';

import type { User, Employee, EmployeeListResponse, AuthResponse } from '../types';

// Create axios instance with base configuration
const api = axios.create({
  baseURL: '/sem',
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: 10000,
});

// Authentication APIs
export const authAPI = {
  signUp: (data: Omit<User, 'id'>) =>
    api.post<AuthResponse>('/signup', {
      inputName: data.Name,
      inputEmail: data.Email,
      inputPassword: data.Password,
    }),
  signIn: (email: string, password: string) =>
    api.post<AuthResponse>('/signin', {
      inputEmail: email,
      inputPassword: password,
    }),
};

// Employee APIs
export const employeeAPI = {
  getAll: () =>
    api.get<EmployeeListResponse>('/employees'),
```

```
getById: (id: string) =>
  api.get<{ data: Employee }>(`/employee/${id}`),

create: (data: Omit<Employee, 'id'>) =>
  api.post('/employee', {
    employeeId: data.EmployeeId,
    name: data.Name,
    kanaName: data.KanaName || '',
    sex: data.Sex,
    postCode: data.PostCode || '',
    address: data.Address || '',
    phoneNumber: data.PhoneNumber || '',
    department: data.Department || '',
    retireFlg: data.RetireFlg,
  }),

update: (id: string, data: Partial<Employee>) =>
  api.put(`/employee/${id}`, {
    employeeId: data.EmployeeId,
    name: data.Name,
    kanaName: data.KanaName || '',
    sex: data.Sex,
    postCode: data.PostCode || '',
    address: data.Address || '',
    phoneNumber: data.PhoneNumber || '',
    department: data.Department || '',
    retireFlg: data.RetireFlg,
  }),

delete: (id: string) =>
  api.delete(`/employee/${id}`),
```

```
};  
  
export default api;
```

### Line-by-line Explanation:

#### Line 1: `import axios from 'axios';`

- **axios**: Popular HTTP client library
- Promise-based
- Works in browser and Node.js
- Better than fetch API (more features)

#### Line 2: `import type { ... } from './types';`

- **import type**: TypeScript-only import (removed at runtime)
- Importing type definitions for type checking
- **Omit, Partial**: TypeScript utility types

### Line 4-11: Axios Instance Creation

#### Line 5: `const api = axios.create({`

- Creating axios instance with custom configuration
- Benefits:
  - Reusable configuration
  - All requests share same settings

- Easier to maintain

**Line 6:** `baseURL: '/sem',`

- **Base URL** for all API requests
- All requests will start with `/sem`
- Example:
  - Request: `api.get('/employees')`
  - Actual URL: `http://localhost:5173/sem/employees`
  - Vite proxy: → `http://localhost:52773/sem/employees`

**Line 7-9:** `headers:`

- Default headers for all requests

**Line 8:** `'Content-Type': 'application/json',`

- Tells server we're sending JSON data
- Server should parse request body as JSON

**Line 10:** `timeout: 10000,`

- Request timeout: 10 seconds (10000 milliseconds)
- If request takes longer → throws timeout error
- Prevents infinite waiting

**Line 13-27: Authentication API Functions**

**Line 14:** `export const authAPI = {`

- Object containing auth-related API functions
- Organized by feature

**Line 15-20: signUp Function**

**Line 15:** `signUp: (data: Omit<User, 'id'>) =>`

- **Parameter type:** `Omit<User, 'id'>`
  - Takes User type
  - Removes 'id' property
  - User can't provide id (database generates it)

**TypeScript Omit Explained:**

typescript

```
// Original User type:  
interface User {  
    id?: number;  
    Email: string;  
    Password: string;  
    Name: string;  
}  
  
// Omit<User, 'id'> becomes:  
{  
    Email: string;  
    Password: string;  
    Name: string;  
}  
// 'id' property removed
```

### Line 16: `api.post<AuthResponse>('/signup', {`

- `api.post`: POST request
- `<AuthResponse>`: Generic type for response
  - TypeScript knows response type
  - Auto-completion works
- `'/signup'`: Endpoint path
  - Full URL: `/sem/signup`

### Line 17-19: Request Body

```
typescript
```

```
{  
  inputName: data.Name,  
  inputEmail: data.Email,  
  inputPassword: data.Password,  
}
```

- Backend expects these exact field names
- **inputName, inputEmail, inputPassword**
- Frontend uses different naming (Name, Email)
- We transform here to match backend expectations

#### Line 22-27: signIn Function

##### Line 22: `signIn: (email: string, password: string) =>`

- Simple parameters: email and password strings
- No complex type needed

##### Line 23: `api.post<AuthResponse>('/signin', {`

- POST request to `/sem/signin`
- Returns AuthResponse type

#### Line 24-26: Request Body

```
typescript
```

```
{  
  inputEmail: email,  
  inputPassword: password,  
}
```

- Transform parameter names for backend

### Line 30-65: Employee API Functions

Line 31: `export const employeeAPI = {`

- Object for employee CRUD operations

### Line 32-33: getAll Function

Line 32: `getAll: ()=>`

- No parameters needed
- Fetches all employees

Line 33: `api.get<EmployeeListResponse>('/employees'),`

- **GET** request
- `<EmployeeListResponse>`: Expected response type

typescript

```
{  
  employees: Employee[]  
}
```

- '/employees': Full URL = `/sem/employees`

### Line 35-36: getById Function

Line 35: `getById: (id: string) =>`

- `id`: Employee's database ID
- Returns single employee

Line 36: `api.get<{ data: Employee }>(`/employee/${id}`),``

- **Template literal:** `'/employee/\${id}'`
  - If `id = "42"` → URL = `/sem/employee/42`
  - Dynamic URL construction
- **Response type:** `{ data: Employee }`

### Line 38-49: create Function

Line 38: `create: (data: Omit<Employee, 'id'>) =>`

- `Omit<Employee, 'id'>`: Employee without id
- New employee doesn't have id yet

### Line 39-48: Request Body Transformation

```
typescript
```

```
api.post('/employee', {
  employeeId: data.EmployeeId,
  name: data.Name,
  kanaName: data.KanaName || '',
  // ... other fields
})
```

## Why transformation?

- Frontend: PascalCase (EmployeeId, KanaName)
- Backend: camelCase (employeeId, kanaName)
- Need to convert naming convention

### Line 42: `kanaName: data.KanaName || '',`

- `|| ''`: Default to empty string if undefined
- Backend expects string, not undefined
- Same for other optional fields

### Line 51-62: update Function

#### Line 51: `update: (id: string, data: Partial<Employee>) =>`

- `Partial<Employee>`: All Employee properties optional
  - Can update any subset of fields
  - Don't need to provide all fields

## TypeScript Partial Explained:

```
typescript

// Original Employee:
interface Employee {
    EmployeeId: string; // required
    Name: string; // required
    Sex: number; // required
}

// Partial<Employee>:
interface PartialEmployee {
    EmployeeId?: string; // optional
    Name?: string; // optional
    Sex?: number; // optional
}
```

**Line 52:** `api.put(`employee/${id}`)`, {

- **PUT** request for update
- Dynamic URL with employee id
- Example: `/sem/employee/42`

**Line 64-65: delete Function**

**Line 64:** `delete: (id: string) =>`

- Simple: just needs id

**Line 65:** `api.delete(`employee/${id}`),`

- **DELETE** request
- Backend performs soft delete (sets deleteFlg = 1)

**Line 67:** `export default api;`

- Export axios instance
- Can be imported for custom requests

### **API Usage Examples:**

typescript

```
// In a component:  
import { authAPI, employeeAPI } from './services/api';  
  
// 1. Sign Up  
try {  
  const response = await authAPI.signUp({  
    Name: "田中太郎",  
    Email: "tanaka@example.com",  
    Password: "password123"  
  });  
  console.log(response.data.message); // "Registration successful."  
} catch (error) {  
  console.error('Signup failed:', error);  
}  
  
// 2. Sign In  
const response = await authAPI.signIn(  
  "tanaka@example.com",  
  "password123"  
);  
  
// 3. Get All Employees  
const response = await employeeAPI.getAll();  
console.log(response.data.employees); // Employee[]  
  
// 4. Create Employee  
await employeeAPI.create({  
  EmployeeId: "12345",  
  Name: "山田花子",  
  Sex: 2,  
  RetireFlg: false  
});
```

```
// 5. Update Employee
await employeeAPI.update("42", {
  Department: "技術部",
  RetireFlg: true
});

// 6. Delete Employee
await employeeAPI.delete("42");
```

### HTTP Request Flow:

```
Component calls API function:  
employeeAPI.getAll()  
↓  
api.get('/employees')  
↓  
Full URL: http://localhost:5173/sem/employees  
↓  
Vite Proxy intercepts  
↓  
Forwards to: http://localhost:52773/sem/employees  
↓  
IRIS Backend processes request  
↓  
Returns JSON response  
↓  
Axios parses response  
↓  
Returns to component as Promise
```

↓  
Component uses data

## Error Handling:

typescript

```
try {
  const response = await employeeAPI.create(data);
  // Success
} catch (error) {
  if (axios.isAxiosError(error)) {
    // Network error or HTTP error
    console.log(error.response?.status); // 404, 500, etc.
    console.log(error.response?.data); // Backend error message
    console.log(error.message); // "Network Error", "timeout"
  }
}
```

## 10. src/components/Layout.tsx

**Purpose:** Common layout wrapper | সব authenticated pages এ navigation bar show করে।

typescript

```
import { ReactNode } from 'react';
import { useNavigate } from 'react-router-dom';
import {
  AppBar,
  Toolbar,
  Typography,
  Button,
  Container,
  Box,
} from '@mui/material';
import { clearAuthData } from '../utils/auth';

interface LayoutProps {
  children: ReactNode;
}

function Layout({ children }: LayoutProps) {
  const navigate = useNavigate();

  const handleLogout = () => {
    clearAuthData();
    navigate('/signin', { replace: true });
  };

  return (
    <Box sx={{ display: 'flex', flexDirection: 'column', minHeight: '100vh' }}>
      {/* Navigation Bar */}
      <AppBar position="static">
        <Toolbar>
          <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
            簡易社員管理システム
          </Typography>
        </Toolbar>
      </AppBar>
    </Box>
  );
}
```

```
<Button color="inherit" onClick={handleLogout}>
  ログアウト
</Button>
</Toolbar>
</AppBar>

/* Main Content */
<Container component="main" sx={{ flexGrow: 1, py: 3 }}>
  {children}
</Container>

/* Footer (Optional) */
<Box component="footer" sx={{ py: 2, textAlign: 'center', bgcolor: '#f5f5f5' }}>
  <Typography variant="body2" color="text.secondary">
    © 2024 Employee Management System
  </Typography>
</Box>
</Box>

);
}

export default Layout;
```

### Line-by-line Explanation:

#### Line 1: import { ReactNode } from 'react';

- **ReactNode**: TypeScript type for React children
- Represents any valid React content:
  - Elements (`<div>`, `<Component />`)
  - Strings, Numbers

- Arrays, Fragments
- null, undefined

**Line 2:** `import { useNavigate } from 'react-router-dom';`

- **useNavigate**: Hook for programmatic navigation
- Returns function to navigate to different routes

### **Line 3-10: Material-UI Imports**

- **AppBar**: Top navigation bar component
- **Toolbar**: Container for AppBar content
- **Typography**: Text component with Material Design styles
- **Button**: Button component
- **Container**: Responsive container (max-width based on breakpoints)
- **Box**: Generic container with sx prop for styling

**Line 11:** `import { clearAuthData } from './utils/auth';`

- Function to clear localStorage and logout

### **Line 13-15: LayoutProps Interface**

**Line 14:** `children: ReactNode;`

- **children**: Content that will be wrapped by Layout
- **ReactNode**: Type for any React content

## Usage:

```
typescript
```

```
<Layout>
  <EmployeeList /> // children
</Layout>
```

### Line 17: `function Layout({ children }: LayoutProps) {}`

- **Destructuring:** Extract children from props
- **Type:** LayoutProps ensures correct prop types

### Line 18: `const navigate = useNavigate();`

- Get navigate function from React Router
- Used for programmatic navigation

### Line 20-23: handleLogout Function

#### Line 20: `const handleLogout = () => {`

- Event handler for logout button click

#### Line 21: `clearAuthData();`

- Remove 'isLoggedIn' and 'userEmail' from localStorage
- User is now logged out

#### Line 22: `navigate('/signin', { replace: true });`

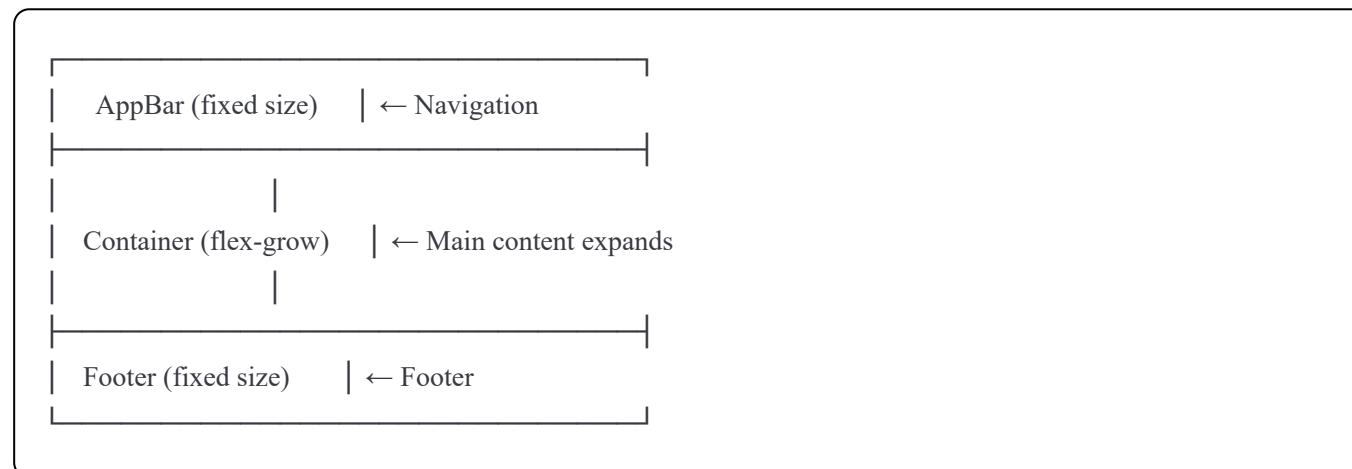
- Navigate to signin page
- **replace: true:** Replace current history entry
  - User can't go back to logged-in page using browser back button
  - Good for logout scenarios

### Line 26-52: JSX Return

Line 26: `<Box sx={{ display: 'flex', flexDirection: 'column', minHeight: '100vh' }}>`

- **Box:** Container component
- **sx prop:** Material-UI styling (similar to CSS)
  - **display: 'flex':** Flexbox layout
  - **flexDirection: 'column':** Vertical stacking
  - **minHeight: '100vh':** Minimum height = viewport height (full screen)

### Why Flexbox?



## Line 28-38: Navigation Bar

### Line 29: `<AppBar position="static">`

- **AppBar**: Material-UI navigation bar
- **position="static"**: Normal flow (not fixed/absolute)
- Alternative: `position="fixed"` (stays at top when scrolling)

### Line 30: `<Toolbar>`

- Container for AppBar content
- Provides padding and alignment

## Line 31-33: Title Typography

### Line 31: `<Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>`

- **variant="h6"**: Heading 6 style
- **component="div"**: Renders as `<div>` (semantic HTML)
- **sx={{ flexGrow: 1 }}**: Takes all available space
  - Pushes logout button to the right

## Line 32: Text Content

簡易社員管理システム  
(Kan'i Shain Kanri Shisutemu)  
Simple Employee Management System

### Line 34-36: Logout Button

Line 34: `<Button color="inherit" onClick={handleLogout}>`

- `color="inherit"`: Inherit color from parent (white on AppBar)
- `onClick={handleLogout}`: Calls logout function on click

Line 35: `ログアウト` (Rogauto)

- "Logout" in Japanese

### Line 40-42: Main Content Container

Line 41: `<Container component="main" sx={{ flexGrow: 1, py: 3 }}>`

- `component="main"`: Semantic HTML (`<main>` tag)
- `sx={{ flexGrow: 1 }}`: Expand to fill available space
  - Footer stays at bottom even if content is short
- `py: 3`: Padding Y-axis (top and bottom)
  - Material-UI spacing scale: 3 = 24px

Line 42: `{children}`

- **Render children here**
- Whatever component is wrapped by Layout appears here
- Example:

```
<Layout>
  <EmployeeList /> // Renders here
</Layout>
```

#### Line 45-50: Footer (Optional)

Line 46: `<Box component="footer" sx={{ py: 2, textAlign: 'center', bgcolor: '#f5f5f5' }}>`

- **component="footer"**: Semantic HTML
- **py: 2**: Padding top and bottom (16px)
- **textAlign: 'center'**: Center text
- **bgcolor: '#f5f5f5'**: Light gray background

#### Line 47-49: Copyright Text

Line 47: `<Typography variant="body2" color="text.secondary">`

- **variant="body2"**: Small body text
- **color="text.secondary"**: Secondary text color (gray)

#### How Layout Works:

```
typescript
```

```
// In EmployeeList.tsx:  
import Layout from '../components/Layout';  
  
function EmployeeList() {  
  return (  
    <Layout>  
      <h1>Employee List</h1>  
      <Table>...</Table>  
    </Layout>  
  );  
}  
  
// Rendered HTML structure:  
<Box> /* Layout wrapper */  
  <AppBar>  
    <Toolbar>  
      <Typography>簡易社員管理システム</Typography>  
      <Button>ログアウト</Button>  
    </Toolbar>  
  </AppBar>  
  
<Container> /* children rendered here */  
  <h1>Employee List</h1>  
  <Table>...</Table>  
</Container>  
  
<Box component="footer">  
  <Typography>© 2024...</Typography>  
</Box>  
</Box>
```

## Logout Flow:

```
User clicks "ログアウト" button  
↓  
handleLogout() called  
↓  
clearAuthData() clears localStorage  
↓  
navigate('/signin', { replace: true })  
↓  
User redirected to signin page  
↓  
Can't go back (history replaced)
```

## Styling Breakdown:

```
typescript  
  
// Material-UI sx prop (shorthand CSS):  
  
sx={{  
  py: 3, //padding-top: 24px; padding-bottom: 24px;  
  flexGrow: 1, //flex-grow: 1;  
  display: 'flex' //display: flex;  
}}  
  
// Spacing scale:  
// 1 = 8px  
// 2 = 16px  
// 3 = 24px  
// 4 = 32px
```

---

## 11. src/components/ProtectedRoute.tsx

**Purpose:** Authentication guard | Login না করলে protected routes access করতে পারবে না।

typescript

```
import { ReactNode } from 'react';
import { Navigate } from 'react-router-dom';
import { isAuthenticated } from '../utils/auth';

interface ProtectedRouteProps {
  children: ReactNode;
}

function ProtectedRoute({ children }: ProtectedRouteProps) {
  if (!isAuthenticated()) {
    // User not logged in, redirect to signin
    return <Navigate to="/signin" replace />;
  }

  // User is authenticated, render children
  return <>{children}</>;
}

export default ProtectedRoute;
```

### Line-by-line Explanation:

**Line 1:** import { ReactNode } from 'react';

- Type for React children content

**Line 2:** `(import { Navigate } from 'react-router-dom');`

- **Navigate:** Component for declarative redirects
- Alternative to `(useNavigate())` hook

**Line 3:** `(import { isAuthenticated } from './utils/auth');`

- Function to check if user is logged in
- Checks localStorage for 'isLoggedIn' === 'true'

### **Line 5-7: ProtectedRouteProps Interface**

**Line 6:** `(children: ReactNode;`

- Content to protect (page components)

**Line 9:** `(function ProtectedRoute({ children }: ProtectedRouteProps) {`

- Wrapper component for protected routes

### **Line 10-13: Authentication Check**

**Line 10:** `(if (!isAuthenticated()) {`

- **Condition:** If user NOT authenticated
- `isAuthenticated()` checks localStorage

**Line 11:** `// Comment explaining redirect`

**Line 12:** `return <Navigate to="/signin" replace />;`

- **Navigate component:** Performs redirect
- **to="/signin":** Destination route
- **replace:** Replace current history entry
  - Prevents back button from returning to protected page
  - Good security practice

### What happens:

```
User tries to access /employees  
↓  
isAuthenticated() returns false  
↓  
<Navigate to="/signin" replace />  
↓  
User redirected to /signin  
↓  
Can't press back to go to /employees
```

### Line 15-16: Render Protected Content

**Line 16:** `return <>{children}</>;`

- **Fragment syntax:** `<>...</>`
- Returns children without extra wrapper
- Only executes if authentication check passes

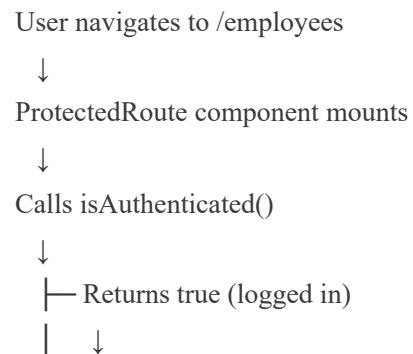
## Usage in App.tsx:

```
typescript

// Protected route:
<Route
  path="/employees"
  element={
    <ProtectedRoute>
      <EmployeeList />
    </ProtectedRoute>
  }
/>

// Flow:
// 1. User navigates to /employees
// 2. React Router tries to render ProtectedRoute
// 3. ProtectedRoute checks authentication:
//   a) If logged in → renders EmployeeList
//   b) If not logged in → redirects to /signin
```

## Complete Flow Diagram:



```
| return <>{children}</>
| ↓
| EmployeeList renders
|
└ Returns false (not logged in)
    ↓
    return <Navigate to="/signin" replace />
    ↓
    Redirect to /signin
    ↓
    SignIn page renders
```

## Why This Pattern?

```
typescript
```

```
// ✗ Without ProtectedRoute (Bad):
<Route path="/employees" element={<EmployeeList />} />
// Problem: Anyone can access by typing URL
// No authentication check!

// ✅ With ProtectedRoute (Good):
<Route
  path="/employees"
  element={
    <ProtectedRoute>
      <EmployeeList />
    </ProtectedRoute>
  }
/>
// ✅ Authentication checked automatically
// ✅ Consistent protection across routes
// ✅ Easy to maintain
```

### Testing Behavior:

typescript

```
// Test 1: Logged out user
localStorage.clear() // No isLoggedIn key
// Navigate to /employees
// → Redirected to /signin ✓

// Test 2: Logged in user
localStorage.setItem('isLoggedIn', 'true')
// Navigate to /employees
// → EmployeeList renders ✓

// Test 3: Invalid login state
localStorage.setItem('isLoggedIn', 'false') // String "false", not boolean
// Navigate to /employees
// → Redirected to /signin ✓
// (isAuthenticated checks === 'true', so any other value fails)
```

### Advanced Pattern (Optional):

typescript

```
// With loading state and role-based access:  
function ProtectedRoute({  
  children,  
  requiredRole  
}: ProtectedRouteProps) {  
  const [loading, setLoading] = useState(true);  
  const [user, setUser] = useState(null);  
  
  useEffect(() => {  
    // Verify token with backend  
    verifyAuth().then(user => {  
      setUser(user);  
      setLoading(false);  
    });  
  }, []);  
  
  if (loading) {  
    return <CircularProgress />;  
  }  
  
  if (!user) {  
    return <Navigate to="/signin" replace />;  
  }  
  
  if (requiredRole && user.role !== requiredRole) {  
    return <Navigate to="/forbidden" replace />;  
  }  
  
  return <>{children}</>;  
}
```

I'll continue with the page components in the next section. The file is getting quite long. Would you like me to continue with SignUp, SignIn, EmployeeList, and EmployeeDetail pages, plus the backend ObjectScript files?