📚 Employee Management System - বাংলা Explanation

(১৫ বছরের ছেলের জন্য সহজ করে বোঝানো)

🎯 Part 1: এই Project টা আসলে কি?

সহজ ভাষায়:

এটা একটা **কর্মচারী ব্যবস্থাপনা সিস্টেম** (Employee Management System)।

কল্পনা করো তুমি একটা অফিসের manager। তোমার কাছে অনেক কর্মচারী আছে। তাদের তথ্য রাখতে হবে - নাম, ID, ফোন নাম্বার, ঠিকানা ইত্যাদি।

আগের দিনে মানুষ কি করতো?

- খাতায় লিখে রাখতো
- Excel sheet এ রাখতো

এখন আমরা কি করলাম?

- একটা ওয়েবসাইট বানিয়েছি
- যেখানে সব কিছু automatically সংরক্ষিত হয়
- সুন্দর interface দিয়ে সহজে ব্যবহার করা যায়

🏗️ Part 2: System Architecture (পুরো সিস্টেম কিভাবে কাজ করে)

এই system টা **তিনটা বড় অংশে** ভাগ:

1️⃣ Frontend (যা তুমি দেখো)

- **Technology**: React.js + TypeScript + Material-UI
- **কাজ**: সুন্দর দেখতে website যেখানে তুমি button click করতে পারো
- **উদাহরণ**: তুমি যখন Facebook খুলো, যা দেখো সেটা Frontend

React.js কি?

- একটা JavaScript library
- ওয়েবসাইট বানানোর tool

- Facebook তৈরি করেছে

TypeScript কি?

- JavaScript এর upgraded version
- বেশি safe, error কম হয়

Material-UI কি?

- Ready-made সুন্দর components
- Button, Form, Table ইত্যাদি already বানানো
- Google এর design system

2️⃣ Backend (যা পর্দার পিছনে কাজ করে)

- **Technology**: InterSystems IRIS
- **Language**: ObjectScript
- **কাজ**: Data save করা, process করা, security দেখা

IRIS কি?

- একটা powerful database
- খুব fast
- Medical, Financial system এ ব্যবহার হয়

3️⃣ Connection (দুইটা কিভাবে কথা বলে)

- **Technology**: REST API
- **কাজ**: Frontend আর Backend এর মধ্যে data পাঠানো

REST API কি?

- দুইটা program এর মধ্যে communication এর নিয়ম
- যেমন: তুমি বন্ধুকে SMS পাঠাও, SMS হলো communication medium

🔍 Part 3: Frontend গভীরভাবে (React.js Explanation)

Project Structure (ফাইল গুলো কিভাবে সাজানো):

```
employee-management-react/
├── public/             # Static files (change হয় না)
│   └── index.html      # Main HTML file
│
├── src/                # Main code folder
│   ├── components/     # Reusable pieces
│   │   ├── Layout.tsx         # Navbar (উপরের menu bar)
│   │   └── ProtectedRoute.tsx   # Security check
│   │
│   ├── pages/          # Different screens
│   │   ├── SignIn.tsx         # Login page
│   │   ├── SignUp.tsx         # Registration page
│   │   ├── EmployeeList.tsx    # Employee দের list
│   │   └── EmployeeDetail.tsx  # Add/Edit form
│   │
│   ├── services/       # Backend এর সাথে কথা বলে
│   │   └── api.ts             # API calls
│   │
│   ├── types/          # Data structure define
│   │   └── index.ts           # TypeScript types
│   │
│   ├── utils/          # Helper functions
│   │   └── auth.ts            # Login related
│   │
│   ├── App.tsx         # Main app (routing)
│   ├── main.tsx        # Entry point
│   └── index.css       # Global styles
│
├── package.json        # Dependencies list
├── vite.config.ts      # Build configuration
└── tsconfig.json       # TypeScript settings
```

React এর মূল ধারণা (Core Concepts):

1. Component কি?

Component মানে হলো একটা **reusable building block।**

উদাহরণ দিয়ে বুঝি: তুমি যদি একটা বাড়ি বানাও:

- ইট একটা component

- দরজা একটা component
- জানালা একটা component

একই ভাবে React এ:

- Button একটা component
- Form একটা component
- Table একটা component

Code Example:

```typescript
// একটা সহজ Button component
function MyButton() {
  return <button>Click Me!</button>;
}
```

2. Props কি?

Props মানে হলো **data pass করা** এক component থেকে আরেক component এ।

Real Life Example: তুমি যখন একটা চিঠি লিখো:

- তুমি = Parent Component
- চিঠি = Props (data)
- তোমার বন্ধু = Child Component

Code Example:

```typescript
// Parent Component
<EmployeeCard name="রহিম" age={25} />

// Child Component
function EmployeeCard(props) {
  return (
    <div>
      <h1>{props.name}</h1>
      <p>বয়স: {props.age}</p>
    </div>
  );
}
```

3. State কি?

State মানে হলো component এর **memory**। যা change হতে পারে।

Real Life Example: তোমার মোবাইলে battery percentage:

- Battery = State
- Charging করলে percentage বাড়ে = State Update

Code Example:

```typescript
const [count, setCount] = useState(0);


// count = current value
// setCount = function to change value
// useState(0) = initial value is 0
```

4. useEffect কি?

useEffect ব্যবহার করা হয় **side effects** এর জন্য।

Side Effect মানে কি?

- API call করা
- Data load করা
- Timer set করা

Real Life Example: তুমি যখন Facebook খুলো:

- Page load হয় (Component mount)
- Automatically newsfeed load হয় (useEffect)

Code Example:

```typescript
useEffect(() => {
  // এই code component load হলে run হবে
  loadEmployees();
}, []); // [] মানে শুধু একবার
```

📁 Part 4: প্রতিটা File এর কাজ (Detailed Explanation)

1. `src/types/index.ts` - Data Structure

কেন দরকার? TypeScript এ আমরা define করে দেই data কেমন হবে।

উদাহরণ:

```typescript
// Employee এর structure
export interface Employee {
  id?: number;              // Optional (নতুন employee এ নাই)
  EmployeeId: string;       // Required - 5 digit ID
  Name: string;             // Required - নাম
  Sex: number;              // 1 = Male, 2 = Female
  PhoneNumber?: string;     // Optional - phone
  RetireFlg: boolean;       // Retired কিনা
}
```

Real Life Example: এটা অনেকটা একটা **Form এর template** এর মত।

- যেমন ভর্তির ফর্মে লেখা থাকে কি কি তথ্য দিতে হবে
- কোনটা required, কোনটা optional

2. `src/services/api.ts` - Backend Connection

কাজ কি? Backend (IRIS database) এর সাথে communication করা।

Structure:

```typescript
```

```
// Axios = HTTP request library
import axios from 'axios';

// Base URL set করা
const api = axios.create({
  baseURL: '/sem',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Authentication API
export const authAPI = {
  signIn: (email, password) =>
    api.post('/signin', { inputEmail: email, inputPassword: password }),

  signUp: (data) =>
    api.post('/signup', data),
};

// Employee API
export const employeeAPI = {
  getAll: () => api.get('/employees'),
  getById: (id) => api.get(`/employee/${id}`),
  create: (data) => api.post('/employee', data),
  update: (id, data) => api.put(`/employee/${id}`, data),
  delete: (id) => api.delete(`/employee/${id}`),
};
```

Real Life Example: ধরো তুমি একটা restaurant এ গেছো:

- তুমি = Frontend
- Waiter = API
- Chef = Backend

তুমি waiter কে order দাও (API call) → Waiter chef এর কাছে যায় → Chef রান্না করে → Waiter তোমার কাছে খাবার নিয়ে আসে (Response)

3. `src/utils/auth.ts` - Authentication Helper

কাজ কি? User login করা আছে কিনা check করা।

Functions:

```
typescript
```

```
// 1. Login check করা
export const isAuthenticated = (): boolean => {
  const isLoggedIn = localStorage.getItem('isLoggedIn');
  return isLoggedIn === 'true';
};

// 2. Login data save করা
export const setAuthData = (email: string): void => {
  localStorage.setItem('isLoggedIn', 'true');
  localStorage.setItem('userEmail', email);
};

// 3. Logout করা
export const clearAuthData = (): void => {
  localStorage.removeItem('isLoggedIn');
  localStorage.removeItem('userEmail');
};
```

localStorage কি?

- Browser এর memory
- Data save করে রাখে
- Page refresh করলেও data থাকে

Real Life Example: তুমি একটা building এ ঢুকছো:

- Security guard = isAuthenticated()
- ID card = localStorage data
- Guard check করে তোমার কাছে ID card আছে কিনা

4. `src/components/Layout.tsx` - Common Layout

কাজ কি? সব page এ common যা আছে সেটা এখানে।

Structure:

```
typescript
```

```
const Layout = ({ children }) => {
  const navigate = useNavigate();

  const handleLogout = () => {
    clearAuthData();
    navigate('/signin');
  };

  return (
    <Box>
      {/* Navbar - সব page এ থাকবে */}
      <AppBar position="static">
        <Toolbar>
          <Typography>簡易社員管理システム</Typography>
          <Button onClick={handleLogout}>ログアウト</Button>
        </Toolbar>
      </AppBar>

      {/* Page content - বদলাবে */}
      <Box component="main">
        {children}
      </Box>
    </Box>
  );
};
```

Real Life Example: তুমি যখন বিভিন্ন channel এ TV দেখো:

- TV এর frame = Layout
- Channel content = children
- Frame same থাকে, content বদলায়

5. `src/components/ProtectedRoute.tsx` - Security Guard

কাজ কি? Login না করলে employee page এ যেতে দিবে না।

Logic:

```
typescript
```

```
const ProtectedRoute = ({ children }) => {
  // Check: User login করা আছে?
  if (!isAuthenticated()) {
    // না থাকলে login page এ পাঠাও
    return <Navigate to="/signin" replace />;
  }

  // হ্যাঁ থাকলে requested page দেখাও
  return <>{children}</>;
};
```

Real Life Example: School এ class room:

- ProtectedRoute = Class teacher
- Login = Roll call
- Roll call এ নাম না থাকলে class এ ঢুকতে পারবে না

📄 Part 5: প্রতিটা Page এর বিস্তারিত

Page 1: `SignUp.tsx` - Registration

Flow:

```
1. User তথ্য দেয় (Name, Email, Password)
  ↓
2. Frontend validation check করে
  ↓
3. Backend এ পাঠায় (API call)
  ↓
4. Backend database এ save করে
  ↓
5. Success message দেখায়
  ↓
6. Login page এ redirect করে
```

Important Code Parts:

```
typescript
```

```
// 1. State management
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');

// 2. Validation
const validateForm = () => {
  if (!email.includes('@')) {
    setError('Invalid email');
    return false;
  }
  if (password.length < 8) {
    setError('Password too short');
    return false;
  }
  return true;
};

// 3. Submit
const handleSubmit = async (e) => {
  e.preventDefault();

  if (!validateForm()) return;

  try {
    const response = await authAPI.signUp({
      inputName: name,
      inputEmail: email,
      inputPassword: password,
    });

    if (response.data.message === 'Registration successful.') {
      navigate('/signin');
    }
  } catch (error) {
    setError('Registration failed');
  }
};
```

Real Life Example: নতুন Facebook account খোলার মত:

- তথ্য দাও → Check হয় → Save হয় → Login করো

Page 2: `SignIn.tsx` - Login

Flow:

```
1. User email + password দেয়
  ↓
2. Frontend validation
  ↓
3. Backend authentication check
  ↓
4. Success হলে localStorage এ save
  ↓
5. Employee list page এ যায়
```

Important Parts:

```typescript
const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    const response = await authAPI.signIn(email, password);

    if (response.data.message === 'Authentication successful.') {
      // Login info save করো
      setAuthData(email);

      // Employee list এ যাও
      navigate('/employees');
    }
  } catch (error) {
    setError('Login failed');
  }
};
```

Page 3: `EmployeeList.tsx` - Employee List

Features:

1. সব employee দেখানো
2. Search করা (ID, Name, Kana name)
3. Filter করা (retired employee show/hide)
4. Sort করা (ID বা Name দিয়ে)
5. Pagination (page wise data)

Important Code:

typescript

```
// 1. Data load করা
useEffect(() => {
  loadEmployees();
}, []);

const loadEmployees = async () => {
  try {
    const response = await employeeAPI.getAll();
    setEmployees(response.data.employees);
  } catch (error) {
    setError('Failed to load');
  }
};

// 2. Search/Filter
const filteredEmployees = useMemo(() => {
  let filtered = employees;

  // Retired filter
  if (!showRetired) {
    filtered = filtered.filter(emp => !emp.RetireFlg);
  }

  // Search filter
  if (searchKeyword) {
    filtered = filtered.filter(emp =>
      emp.EmployeeId.includes(searchKeyword) ||
      emp.Name.includes(searchKeyword)
    );
  }

  // Sort
  filtered = filtered.sort((a, b) => {
    if (order === 'asc') {
      return a[orderBy] > b[orderBy] ? 1 : -1;
    } else {
      return a[orderBy] < b[orderBy] ? 1 : -1;
    }
  });

  return filtered;
}, [employees, showRetired, searchKeyword, orderBy, order]);

// 3. Pagination
const paginatedEmployees = filteredEmployees.slice(
  page * rowsPerPage,
  page * rowsPerPage + rowsPerPage
);
```

Real Life Example: Phone এর contact list এর মত:

- সব contact দেখা যায়
- Search করা যায়
- Sort করা যায় (Name/Number)
- Scroll করা যায়

Page 4: `EmployeeDetail.tsx` - Add/Edit Employee

Two Modes:

1. **Add Mode**: URL = `/employees/new`
2. **Edit Mode**: URL = `/employees/:id`

Flow:

Add Mode:

```
1. Empty form দেখায়
  ↓
2. User তথ্য fill করে
  ↓
3. Validation check
  ↓
4. Confirmation dialog
  ↓
5. Backend এ POST request
  ↓
6. Success হলে list এ redirect
```

Edit Mode:

```
1. Employee ID দিয়ে data load করে
  ↓
2. Form এ pre-filled data দেখায়
  ↓
3. User change করে
  ↓
4. Validation + Confirmation
  ↓
5. Backend এ PUT request
  ↓
6. Success হলে list এ redirect
```

Important Code:

```
typescript
```

```
// 1. Check: Add or Edit?
const isNewEmployee = id === 'new';

// 2. Load data (Edit mode only)
useEffect(() => {
  if (!isNewEmployee && employeeId) {
    loadEmployeeData(employeeId);
  }
}, [isNewEmployee, employeeId]);

const loadEmployeeData = async (id) => {
  try {
    const response = await employeeAPI.getById(id);

    // Form fill করো
    setFormData({
      employeeId: response.data.EmployeeId,
      name: response.data.Name,
      // ... other fields
    });
  } catch (error) {
    setError('Load failed');
  }
};

// 3. Submit (Add or Update)
const handleSubmit = async (e) => {
  e.preventDefault();

  if (!validateForm()) return;

  try {
    if (isNewEmployee) {
      // Create new
      await employeeAPI.create(formData);
    } else {
      // Update existing
      await employeeAPI.update(employeeId, formData);
    }

    navigate('/employees');
  } catch (error) {
    setError('Save failed');
  }
};
```

```
// 4. Delete (Edit mode only)
const handleDelete = async () => {
  try {
    await employeeAPI.delete(employeeId);
    navigate('/employees');
  } catch (error) {
    setError('Delete failed');
  }
};
```

🔧 Part 6: Backend (IRIS + ObjectScript)

Database Tables:

1. `tblAccount` - User accounts

```objectscript
Class SEM.tblAccount Extends %Persistent
{
  Property Email As %String(MAXLEN = 32) [ Required ];
  Property Password As %String(MAXLEN = 32);
  Property Name As %String(MAXLEN = 64);
}
```

2. `tblEmployee` - Employee data

```objectscript
Class SEM.tblEmployee Extends %Persistent
{
  Property EmployeeId As %String(MAXLEN = 5) [ Required ];
  Property Name As %String(MAXLEN = 64) [ Required ];
  Property KanaName As %String(MAXLEN = 64);
  Property Sex As %Integer;
  Property PostCode As %String(MAXLEN = 8);
  Property Address As %String(MAXLEN = 1024);
  Property PhoneNumber As %String(MAXLEN = 13);
  Property Department As %String(MAXLEN = 64);
  Property RetireFlg As %Boolean [ Required ];
  Property deleteFlg As %Boolean [ Required ];
  Property upDateTime As %DateTime [ Required ];
}
```

REST API Structure:

```objectscript
Class SEM.SEMRESTAPI Extends %CSP.REST
{
 XData UrlMap
 {
  <Routes>
   <!-- Authentication -->
   <Route Url="/signup" Method="POST" Call="AccountRegistration"/>
   <Route Url="/signin" Method="POST" Call="AccountLogin"/>

   <!-- Employee CRUD -->
   <Route Url="/employees" Method="GET" Call="GetAllEmployees"/>
   <Route Url="/employee/:id" Method="GET" Call="GetEmployeeById"/>
   <Route Url="/employee" Method="POST" Call="CreateEmployee"/>
   <Route Url="/employee/:id" Method="PUT" Call="UpdateEmployee"/>
   <Route Url="/employee/:id" Method="DELETE" Call="DeleteEmployee"/>
  </Routes>
 }
}
```

API Endpoints Explained:

1. POST /signup - Registration

```objectscript
```

```
ClassMethod AccountRegistration() As %Status
{
  // 1. Request থেকে data নাও
  Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)
  Set reqName = requestObject.inputName
  Set reqEmail = requestObject.inputEmail
  Set reqPassword = requestObject.inputPassword

  // 2. Validation
  If (reqEmail = "") {
    Set result.message = "Email is required."
    Return status
  }

  // 3. Check: Email already exists?
  Set sqlQuery = "SELECT ID FROM SEM.tblAccount WHERE Email = ?"
  // ... query execute

  If resultSet.%Next() {
    // Already exists
    Return "Email already registered."
  }

  // 4. Save to database
  Set newAccount = ##Class(SEM.tblAccount).%New()
  Set newAccount.Email = reqEmail
  Set newAccount.Name = $ZCONVERT(reqName, "I", "UTF8")
  Set newAccount.Password = reqPassword
  Set status = newAccount.%Save()

  Return "Registration successful."
}
```

Real Life Example: নতুন library card বানানোর মত:

- Name দাও → Check করে duplicate নাকি → Card বানায় → Database এ save

2. POST /signin - Login

```objectscript
```

```
ClassMethod AccountLogin() As %Status
{
 // 1. Email + Password নাও
 Set reqEmail = requestObject.inputEmail
 Set reqPassword = requestObject.inputPassword

 // 2. Database check করো
 Set sqlQuery = "SELECT ID, Password FROM SEM.tblAccount WHERE Email = ?"
 // ... execute query

 If resultSet.%Next() {
  Set storedPassword = resultSet.%Get("Password")

  // 3. Password match?
  If storedPassword = reqPassword {
   Return "Authentication successful."
  } Else {
   Return "Invalid password."
  }
 } Else {
  Return "Email is not registered."
 }
}
```

3. GET /employees - Get all employees

objectscript

```
ClassMethod GetAllEmployees() As %Status
{
  // 1. SQL query
  Set sqlQuery = "SELECT ID, EmployeeId, Name, Sex, ...
                 FROM SEM.tblEmployee
                 WHERE deleteFlg = 0
                 ORDER BY upDateTime DESC"

  // 2. Execute query
  Set statement = ##class(%SQL.Statement).%New()
  Set resultSet = statement.%Execute()

  // 3. Loop through results
  While resultSet.%Next() {
    Set employee = {}
    Set employee.id = resultSet.%Get("ID")
    Set employee.EmployeeId = resultSet.%Get("EmployeeId")
    Set employee.Name = resultSet.%Get("Name")
    // ... other fields

    Do result.employees.%Push(employee)
  }

  Return result
}
```

4. POST /employee - Create employee

objectscript

```
ClassMethod CreateEmployee() As %Status
{
  // 1. Request data
  Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)

  // 2. Validation
  If (requestObject.employeeId = "") {
    Return "Employee ID is required."
  }

  // 3. Check duplicate
  Set sqlQuery = "SELECT ID FROM SEM.tblEmployee
                 WHERE EmployeeId = ? AND deleteFlg = 0"
  // ... execute

  If resultSet.%Next() {
    Return "Employee ID already exists"
  }

  // 4. Create new employee
  Set newEmployee = ##Class(SEM.tblEmployee).%New()
  Set newEmployee.EmployeeId = requestObject.employeeId
  Set newEmployee.Name = $ZCONVERT(requestObject.name, "I", "UTF8")
  // ... other fields
  Set newEmployee.deleteFlg = 0
  Set newEmployee.upDateTime = $ZDATETIME($HOROLOG, 3)

  // 5. Save
  Set status = newEmployee.%Save()

  Return "Employee created successfully"
}
```

5. PUT /employee/:id - Update employee

```
objectscript
```

```
ClassMethod UpdateEmployee(id As %String) As %Status
{
  // 1. Load existing employee
  Set employee = ##class(SEM.tblEmployee).%OpenId(id)

  If '$IsObject(employee) {
    Return "Employee not found"
  }

  // 2. Update fields
  Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)
  Set employee.EmployeeId = requestObject.employeeId
  Set employee.Name = $ZCONVERT(requestObject.name, "I", "UTF8")
  // ... other fields
  Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)

  // 3. Save
  Set status = employee.%Save()

  Return "Employee updated successfully"
}
```

6. DELETE /employee/:id - Delete employee (Soft delete)

```objectscript
ClassMethod DeleteEmployee(id As %String) As %Status
{
  // 1. Load employee
  Set employee = ##class(SEM.tblEmployee).%OpenId(id)

  If '$IsObject(employee) {
    Return "Employee not found"
  }

  // 2. Soft delete (শুধু flag set করো)
  Set employee.deleteFlg = 1
  Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)

  // 3. Save
  Set status = employee.%Save()

  Return "Employee deleted successfully"
}
```

Soft Delete কি?

- Employee actually delete করা হয় না
- শুধু একটা flag set করা হয় (deleteFlg = 1)
- পরে recover করা যায়

Real Life Example: Windows Recycle Bin এর মত:

- File delete করলে Recycle Bin এ যায়
- Actually delete হয় না
- পরে restore করা যায়

🔄 Part 7: Complete Data Flow (পুরো প্রক্রিয়া)

Scenario 1: নতুন Employee যোগ করা

Step-by-step:

```
User Action → Frontend → Backend → Database → Response → UI Update

1. User "新規登録" button এ click করে
  ↓
2. Frontend: Navigate to /employees/new
  ↓
3. User form fill করে (ID, Name, Sex, etc.)
  ↓
4. User "登録" button click করে
  ↓
5. Frontend: Validation check
   - Employee ID 5 digit?
   - Name empty না?
   - Sex selected?
  ↓
6. Frontend: Confirmation dialog দেখায়
  ↓
7. User "はい" click করে
  ↓
8. Frontend: API call
   POST /sem/employee
   Body: { employeeId: "12345", name: "山田太郎", ... }
  ↓
9. Backend: Request receive করে
  ↓
10. Backend: Validation check
    - Required fields আছে?
    - Employee ID duplicate?
   ↓
11. Backend: Database এ save
    INSERT INTO tblEmployee ...
   ↓
12. Database: Row insert করে, ID return করে
   ↓
13. Backend: Response send করে
    { message: "Employee created successfully", id: 42 }
   ↓
14. Frontend: Success message দেখায়
   ↓
15. Frontend: Navigate to /employees (list page)
   ↓
16. Frontend: Employee list reload করে
    GET /sem/employees
   ↓
17. Backend: All employees return করে
```

```
  ↓
18. Frontend: Table update করে, নতুন employee list এ দেখায়
```

Visual Flow:

```
Browser                 Vite Proxy           IRIS Backend
  |                         |                      |
  | POST /sem/employee      |                      |
  |------------------------>|                      |
  |                         | POST /employee       |
  |                         |--------------------->|
  |                         |                      |
  |                         |      [Validation]    |
  |                         |                      |
  |                         |     [Save to DB]     |
  |                         |                      |
  |                         |   { success: true }  |
  |                         |<---------------------|
  | { success: true }       |                      |
  |<------------------------|                      |
  |                         |                      |
  | [Show success message]  |                      |
  | [Redirect to list]      |                      |
```

Scenario 2: Employee Search করা

```
1. User search box এ "山田" type করে
  ↓
2. Frontend: searchKeyword state update হয়
  ↓
3. Frontend: useMemo hook trigger হয়
  ↓
4. Frontend: Filter logic run হয়
   - employees.filter(emp =>
       emp.Name.includes("山田") ||
       emp.EmployeeId.includes("山田")
     )
  ↓
5. Frontend: filteredEmployees state update হয়
  ↓
6. Frontend: Table re-render হয়
  ↓
7. User: শুধু matching employees দেখতে পায়
```

Note: Search locally হয় (no API call)। কারণ সব data already load করা আছে।

🛠️ Part 8: Important Concepts বিস্তারিত

1. React Hooks Deep Dive

useState

কি করে: Component এর data/state manage করে যা change হতে পারে।

Syntax:

```typescript
const [value, setValue] = useState(initialValue);
```

Example:

```typescript
const [count, setCount] = useState(0);

// count read করো
console.log(count); // 0

// count change করো
setCount(5);
console.log(count); // 5

// Previous value use করো
setCount(prev => prev + 1);
console.log(count); // 6
```

Real Life: তোমার wallet এ টাকা:

- count = টাকার পরিমাণ
- setCount = টাকা add/subtract করা
- Re-render = তুমি check করো কত টাকা আছে

useEffect

কি করে: Side effects handle করে (API call, subscriptions, timers)।

Syntax:

```typescript
useEffect(() => {
  // Effect code

  return () => {
    // Cleanup (optional)
  };
}, [dependencies]);
```

Examples:

```typescript
// 1. Run once (component mount এ)
useEffect(() => {
  console.log('Component mounted');
}, []); // Empty dependency array

// 2. Run when specific value changes
useEffect(() => {
  console.log('Count changed:', count);
}, [count]); // Runs when count changes

// 3. Run on every render
useEffect(() => {
  console.log('Component rendered');
}); // No dependency array

// 4. With cleanup
useEffect(() => {
  const timer = setInterval(() => {
    console.log('Tick');
  }, 1000);

  // Cleanup
  return () => {
    clearInterval(timer);
  };
}, []);
```

Real Life: Facebook newsfeed:

- Component mount = Page খুললে
- useEffect = Automatically posts load হয়

- Cleanup = Page close করলে loading stop

useMemo

কি করে: Expensive calculation cache করে। শুধু dependency change হলে re-calculate করে।

Syntax:

```typescript
const memoizedValue = useMemo(() => {
  return expensiveCalculation(a, b);
}, [a, b]);
```

Example:

```typescript
const filteredEmployees = useMemo(() => {
  console.log('Filtering...');

  return employees.filter(emp =>
    emp.Name.includes(searchKeyword)
  );
}, [employees, searchKeyword]);

// employees বা searchKeyword change হলে শুধু তখনই re-filter হবে
// অন্য state change হলে re-filter হবে না
```

Real Life: Calculator:

- তুমি 2 + 3 = 5 calculate করলে
- useMemo এটা মনে রাখে
- আবার 2 + 3 করলে calculate না করে directly 5 বলে দেয়

2. TypeScript Benefits

Type Safety

Without TypeScript:

```javascript
```

```
// Error catch করা কঠিন
function addEmployee(employee) {
  // employee.name আছে? নাকি employee.Name?
  // employee.id number? string?
  // Runtime এ error দিবে
}
```

With TypeScript:

```typescript
interface Employee {
  id: number;
  Name: string;
}

function addEmployee(employee: Employee) {
  // TypeScript compile time এ বলে দিবে
  // employee.name wrong (should be Name)
  // employee.id must be number
}
```

Auto-completion

```typescript
const employee: Employee = {
  // এখানে type করার সময় editor suggest করবে:
  // id, Name, Sex, etc.
};
```

3. Material-UI Components

Why Material-UI?

- Google এর Material Design follow করে
- Professional look
- Responsive (mobile friendly)
- Accessibility built-in

Common Components:

```typescript
```

```
// Button
<Button variant="contained" color="primary">
  Click Me
</Button>

// Text Field
<TextField
  label="Name"
  value={name}
  onChange={(e) => setName(e.target.value)}
  required
/>

// Table
<Table>
  <TableHead>
    <TableRow>
      <TableCell>Name</TableCell>
      <TableCell>Age</TableCell>
    </TableRow>
  </TableHead>
  <TableBody>
    {data.map(row => (
      <TableRow key={row.id}>
        <TableCell>{row.name}</TableCell>
        <TableCell>{row.age}</TableCell>
      </TableRow>
    ))}
  </TableBody>
</Table>

// Dialog
<Dialog open={open} onClose={handleClose}>
  <DialogTitle>Are you sure?</DialogTitle>
  <DialogContent>
    <DialogContentText>
      Do you want to delete this employee?
    </DialogContentText>
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose}>Cancel</Button>
    <Button onClick={handleConfirm}>Confirm</Button>
  </DialogActions>
</Dialog>
```

4. REST API Concepts

REST = Representational State Transfer

HTTP Methods:

- GET: Data read করা
- POST: New data create করা
- PUT: Existing data update করা
- DELETE: Data delete করা

Status Codes:

- 200: Success
- 201: Created
- 400: Bad Request (client error)
- 404: Not Found
- 500: Server Error

Example Flow:

```
Frontend                Backend
  |                        |
  | GET /employees         |
  |----------------------->|
  |                        | [Query database]
  |                        |
  | 200 OK                 |
  | [{emp1}, {emp2}, ...]  |
  |<-----------------------|
  |                        |
```

5. Proxy Configuration

Why Proxy?

- Frontend: `http://localhost:5173`
- Backend: `http://localhost:52773`
- Direct call করলে CORS error

Solution:

```typescript
// vite.config.ts
export default defineConfig({
  server: {
    proxy: {
      '/sem': {
        target: 'http://localhost:52773',
        changeOrigin: true,
      },
    },
  },
});
```

How it works:

```
Frontend              Proxy              Backend
   |                    |                   |
   | GET /sem/employees  |                   |
   |--------------------->|                  |
   |                    | GET /sem/employees |
   |                    |------------------->|
   |                    |                   |
   |                    |     [Response]     |
   |                    |<-------------------|
   |    [Response]       |                   |
   |<---------------------|                  |
```

🎓 Part 9: Key Takeaways (মূল শিক্ষা)

Frontend:

1. **React Components** = Reusable UI building blocks
2. **State Management** = useState for dynamic data
3. **Side Effects** = useEffect for API calls
4. **Performance** = useMemo for optimization
5. **Type Safety** = TypeScript prevents errors

Backend:

1. **IRIS Database** = Fast, reliable data storage

2. **ObjectScript** = Powerful server-side language
3. **REST API** = Standard communication protocol
4. **Soft Delete** = Data recovery possible
5. **UTF-8 Conversion** = Proper Japanese character handling

Architecture:

1. **Separation of Concerns** = Frontend ≠ Backend
2. **API Layer** = Clean communication interface
3. **Authentication** = Security first approach
4. **Validation** = Client + Server side
5. **Error Handling** = Graceful failure management

💡 Part 10: Common Patterns & Best Practices

1. Error Handling Pattern

```typescript
const [loading, setLoading] = useState(false);
const [error, setError] = useState<string | null>(null);

const fetchData = async () => {
  setLoading(true);
  setError(null);

  try {
    const response = await api.getData();
    setData(response.data);
  } catch (err: any) {
    setError(err.message || 'Something went wrong');
  } finally {
    setLoading(false);
  }
};
```

2. Form Handling Pattern

```typescript
```

```
const [formData, setFormData] = useState({
  name: '',
  email: '',
});

const handleChange = (field: string, value: any) => {
  setFormData(prev => ({
    ...prev,
    [field]: value,
  }));
};

const handleSubmit = async (e: FormEvent) => {
  e.preventDefault();

  if (!validate()) return;

  await saveData(formData);
};
```

3. Conditional Rendering Pattern

```typescript
return (
  <div>
    {loading && <CircularProgress />}

    {error && <Alert severity="error">{error}</Alert>}

    {!loading && !error && data.length === 0 && (
      <Typography>No data found</Typography>
    )}

    {!loading && !error && data.length > 0 && (
      <Table data={data} />
    )}
  </div>
);
```

🎯 Conclusion

এই project টা একটা **complete full-stack application**:

- Modern Frontend (React + TypeScript)
- Powerful Backend (IRIS + ObjectScript)
- Professional UI (Material-UI)
- Secure Authentication
- CRUD Operations
- Search/Filter/Sort
- Pagination

এটা real-world production system এ যা ব্যবহার হয় তার একটা miniature version।

তুমি এই project থেকে শিখেছো:

- React development
- State management
- API integration
- Database operations
- REST API design
- TypeScript
- Material-UI

এগুলো হলো modern web development এর **core skills**।

এই explanation পড়ার পর তুমি এখন:

1. Project structure বুঝতে পারবে
2. প্রতিটা file এর কাজ জানো
3. Data flow বুঝতে পারবে
4. Frontend-Backend communication বুঝতে পারবে
5. Presentation এ confidence নিয়ে explain করতে পারবে

শুভকামনা তোমার presentation এর জন্য! 🚀