🎯 Test Case Scenario: Complete Demo Flow

Employee Management System - Live Demonstration

📋 Overview

Demo Duration: 5-6 minutes
Flow: Signup → Login → Add Employee → Edit Employee → Delete Employee
Audience: Senior Engineers (Technical)
Your Role: IRIS & React.js Intern

🔄 Complete Test Case Flow

Pre-Demo Setup Checklist

- [ ] Backend IRIS server running (localhost:52773)
- [ ] Frontend dev server running (localhost:5173)
- [ ] Browser opened to `http://localhost:5173`
- [ ] Browser developer console closed (cleaner view)
- [ ] Test data prepared (see below)

📝 Test Data Preparation

Test Account for Signup/Login

> Name: 田中太郎 (Tanaka Taro)
> Email: tanaka.taro@example.com
> Password: password123

Test Employee Data

Employee 1 (New Addition)

Employee ID: 12345
Name: 山田花子 (Yamada Hanako)
Kana Name: ヤマダハナコ
Sex: Female (女性)
Post Code: 100-0001
Address: 東京都千代田区千代田1-1
Phone Number: 090-1234-5678
Department: 営業部
Retire Flag: No (在職中)

Employee 2 (For Editing)

Employee ID: 67890
Name: 佐藤次郎 (Sato Jiro)
Kana Name: サトウジロウ
Sex: Male (男性)
Post Code: 150-0001
Address: 東京都渋谷区神宮前1-1
Phone Number: 080-9876-5432
Department: 開発部
Retire Flag: No (在職中)

↓ Will be updated to ↓

Name: 佐藤次郎 (Sato Jiro) [SAME]
Department: 技術部 [CHANGED]
Retire Flag: Yes (退職済み) [CHANGED]

🎬 Step-by-Step Demo Script

PHASE 1: User Signup (60 seconds)

Actions:

1. Navigate to: http://localhost:5173
2. Click: 「新規登録」(Sign Up) link
3. Fill the signup form:
   - **氏名 (Name):** 田中太郎
   - **メールアドレス (Email):** tanaka.taro@example.com
   - **パスワード (Password):** password123

4. Click: 「登録」(Register) button
5. Observe: Success message and redirect to Sign In page

What to Say (English): "First, I'll create a new user account. This is the signup page where we collect basic user information. After registration, the system stores the account in IRIS database and redirects to the login page."

Technical Notes to Mention:

- Frontend validates email format and password length
- Backend API endpoint: `POST /sem/signup`
- Password stored in IRIS `tblAccount` table
- Duplicate email check implemented

Expected Result:

- ✅ Account created successfully
- ✅ Redirect to Sign In page
- ✅ Success message displayed

PHASE 2: User Login (40 seconds)

Actions:

1. On Sign In page, enter credentials:
   - **Email:** tanaka.taro@example.com
   - **Password:** password123
2. Click: 「ログイン」(Login) button
3. Observe: Authentication success and redirect to Employee List

What to Say (English): "Now I'll authenticate using the account we just created. The system validates credentials against the IRIS database. Upon successful authentication, a session token is stored in localStorage, and we're redirected to the employee management dashboard."

Technical Notes to Mention:

- Frontend sends credentials to backend
- Backend API endpoint: `POST /sem/signin`
- IRIS validates email and password match

- Client-side session management using localStorage
- Protected route implementation (ProtectedRoute component)

Expected Result:

✅ Login successful
✅ Redirect to /employees page
✅ Navigation bar appears with logout option
✅ Employee list loads (may be empty initially)

PHASE 3: Add New Employee (90 seconds)

Actions:

1. On Employee List page, click: 「新規登録」(New Registration) button
2. Fill the employee form:
    - **社員番号 (Employee ID):** 12345
    - **氏名 (Name):** 山田花子
    - **カナ氏名 (Kana Name):** ヤマダハナコ
    - **性別 (Sex):** Select 「女性」(Female)
    - **郵便番号 (Post Code):** 100-0001
    - **住所 (Address):** 東京都千代田区千代田1-1
    - **電話番号 (Phone):** 090-1234-5678
    - **部署 (Department):** 営業部
    - **退職フラグ (Retire Flag):** Uncheck (在職中)
3. Click: 「登録」(Register) button
4. In confirmation dialog, click: 「はい」(Yes)
5. Observe: Success message and redirect to list with new employee

What to Say (English): "Let me demonstrate adding a new employee. This form captures all essential employee information. Notice the validation—for example, Employee ID must be exactly 5 digits. The Sex field uses a dropdown for data consistency. When I submit, the system shows a confirmation dialog to prevent accidental submissions."

Technical Notes to Mention:

- **Frontend:**

- React controlled components with useState
- Form validation before submission
- Material-UI components (TextField, Select, Dialog)
- Confirmation dialog pattern

- **Backend:**
  - API endpoint: `POST /sem/employee`
  - Duplicate Employee ID check in IRIS
  - UTF-8 conversion for Japanese characters using `$ZCONVERT`
  - Timestamp stored with `$ZDATETIME($HOROLOG, 3)`
- **Database:**
  - Data saved in `tblEmployee` table
  - `deleteFlg` set to 0 (not deleted)
  - Auto-generated ID field

Expected Result:

- ✅ Form validation passes
- ✅ Confirmation dialog appears
- ✅ Employee created successfully
- ✅ Redirect to employee list
- ✅ New employee visible in table (Employee ID: 12345, Name: 山田花子)

PHASE 4: Search and View Employee (30 seconds)

Actions:

1. In the search box, type: 「山田」
2. Observe: Table filters to show only matching employees
3. Clear search box
4. Observe: All employees displayed again

What to Say (English): "The system includes real-time search functionality. As I type, the list filters immediately without making API calls—this is efficient client-side filtering using React's useMemo hook. The search works across Employee ID, Name, and Kana Name fields."

Technical Notes to Mention:

- Client-side filtering (no backend call)
- useMemo hook for performance optimization
- Debounced search (optional improvement to mention)
- Case-insensitive matching

Expected Result:

✅ Search filters results instantly
✅ Shows only matching employees
✅ Clearing search shows all employees

PHASE 5: Edit Existing Employee (90 seconds)

Actions:

1. First, add another employee (Employee ID: 67890) to demonstrate editing
   - **社員番号:** 67890
   - **氏名:** 佐藤次郎
   - **カナ氏名:** サトウジロウ
   - **性別:** 男性
   - **郵便番号:** 150-0001
   - **住所:** 東京都渋谷区神宮前1-1
   - **電話番号:** 080-9876-5432
   - **部署:** 開発部
   - **退職フラグ:** Uncheck
2. After adding, click on 「編集」(Edit) icon for Employee 67890
3. Modify fields:
   - **部署 (Department):** Change from 「開発部」 to 「技術部」
   - **退職フラグ (Retire Flag):** Check it (mark as retired)
4. Click: 「更新」(Update) button
5. In confirmation dialog, click: 「はい」(Yes)
6. Observe: Success message and updated data in list

What to Say (English): "Now let's edit an existing employee. When I click the edit icon, the system fetches the employee details from IRIS and pre-fills the form. This demonstrates the difference between ADD and EDIT modes—same component, different behavior based on the route parameter. Notice that Employee ID is read-only during editing to maintain data integrity."

Technical Notes to Mention:

- **Frontend:**
  - Same EmployeeDetail component for add/edit
  - Route parameter determines mode: `/employees/new` vs `/employees/:id`
  - useEffect loads existing data in edit mode
  - Employee ID field disabled during edit
- **Backend:**
  - API endpoint: `GET /sem/employee/:id` (fetch)
  - API endpoint: `PUT /sem/employee/:id` (update)
  - Validation ensures required fields present
  - Update timestamp: `upDateTime = $ZDATETIME($HOROLOG, 3)`
- **Database:**
  - ObjectScript `%OpenId(id)` loads existing record
  - Modifies object properties
  - `%Save()` commits changes

Expected Result:

- ✅ Edit form loads with pre-filled data
- ✅ Employee ID field is read-only
- ✅ Changes saved successfully
- ✅ Updated data visible in list
- ✅ Department changed to 技術部
- ✅ Retire flag checked (退職済み)

PHASE 6: Delete Employee (40 seconds)

Actions:

1. In edit mode for Employee 67890, click: 「削除」(Delete) button at the bottom
2. In delete confirmation dialog, click: 「はい」(Yes)

3. Observe: Success message and employee removed from list

What to Say (English): "Finally, let's delete an employee. This system implements soft delete—the record isn't physically removed from the database. Instead, we set a deleteFlg to 1, which filters it out from the list. This approach allows data recovery if needed and maintains referential integrity."

Technical Notes to Mention:

- **Soft Delete Pattern:**
  - Record not physically deleted
  - `deleteFlg` set to 1
  - Employee still exists in database but hidden
  - Can be recovered by changing flag back to 0
- **API:**
  - Endpoint: `DELETE /sem/employee/:id`
  - Backend sets `deleteFlg = 1` and updates timestamp
- **Benefits:**
  - Data recovery possible
  - Audit trail maintained
  - Referential integrity preserved
- **Query Filtering:**
  - List query: `WHERE deleteFlg = 0`
  - Automatically excludes deleted records

Expected Result:

- ✅ Delete confirmation dialog appears
- ✅ Employee removed from list
- ✅ Success message displayed
- ✅ Record still in database with deleteFlg = 1

PHASE 7: Logout (20 seconds)

Actions:

1. Click: 「ログアウト」(Logout) button in navigation bar
2. Observe: Redirect to Sign In page

What to Say (English): "When logging out, the system clears localStorage, removing the session token. This ensures security—if I try to access /employees directly, the ProtectedRoute component will redirect me back to login."

Technical Notes to Mention:

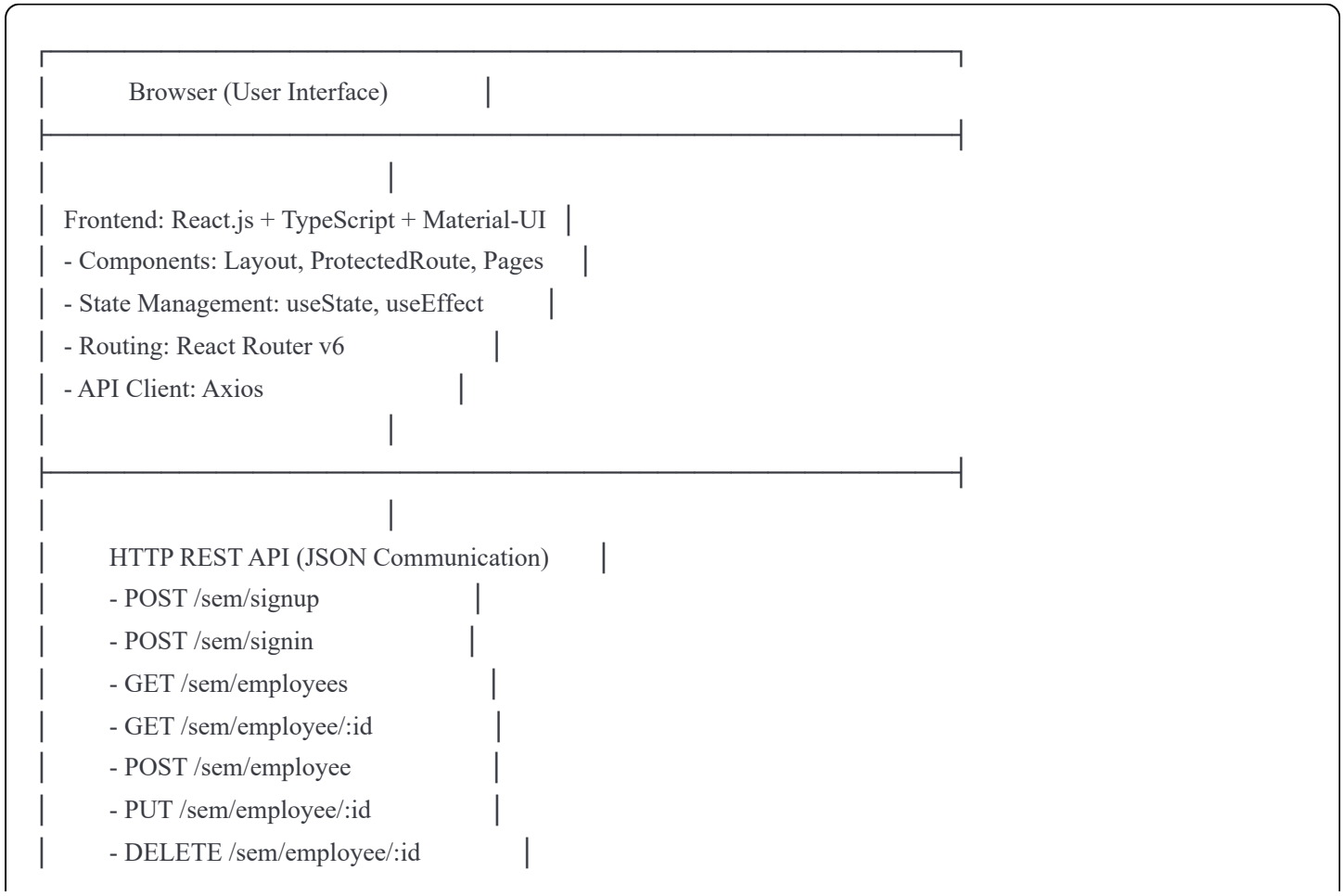
- localStorage cleared on logout
- Session management client-side
- Protected routes enforce authentication
- Clean state reset

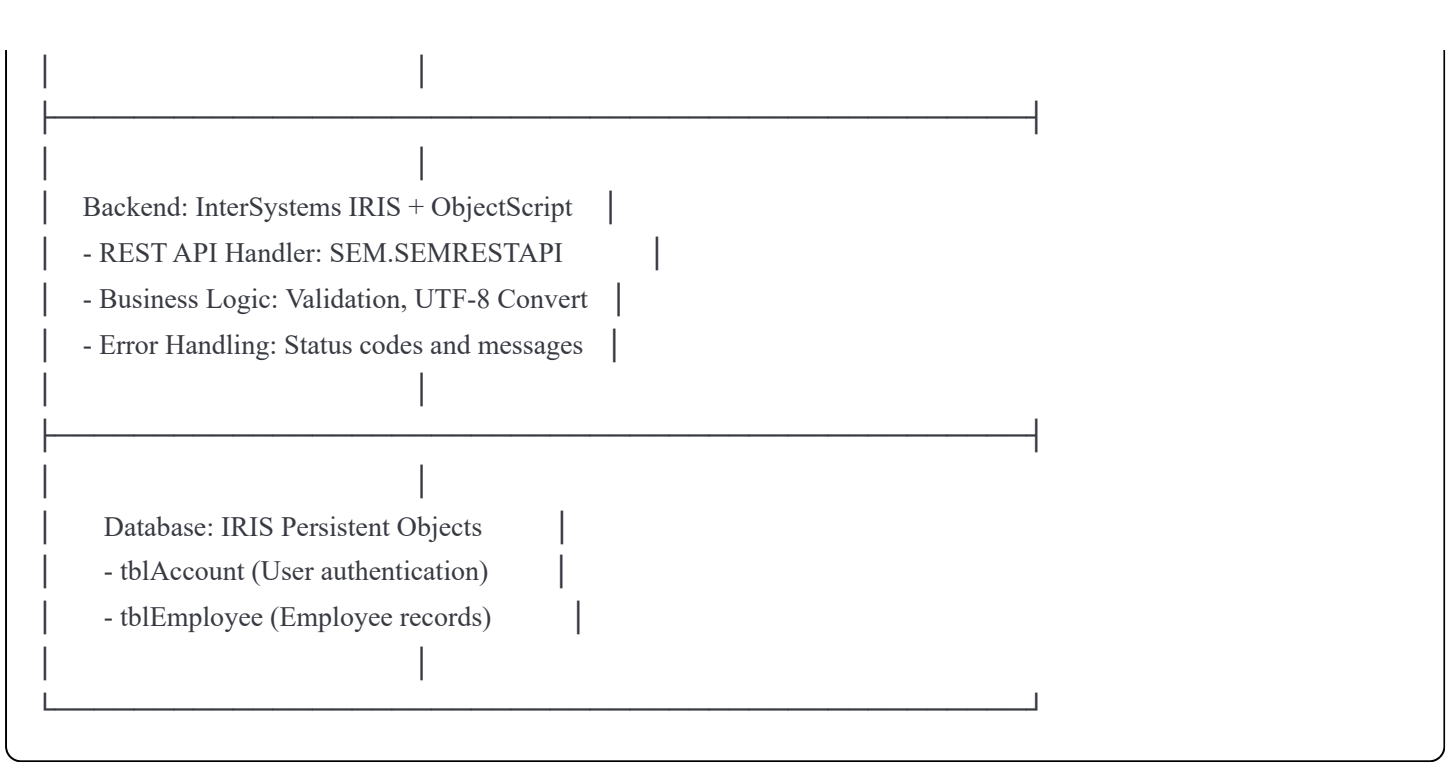
Expected Result:

✅ Logout successful
✅ Redirect to Sign In page
✅ Cannot access /employees without login

🏗️ Architecture Overview (For Technical Discussion)

System Architecture Layers





Frontend Technology Stack

Core:

- **React 18**: Component-based UI library
- **TypeScript**: Type-safe JavaScript superset
- **Vite**: Fast build tool and dev server

UI Framework:

- **Material-UI (MUI)**: Google Material Design components
- **React Router v6**: Client-side routing

State & Data:

- **useState**: Component state management
- **useEffect**: Side effects (API calls, lifecycle)
- **useMemo**: Performance optimization (memoization)
- **Axios**: HTTP client for API communication

Project Structure:

```
src/
├── components/       # Reusable components
│   ├── Layout.tsx          # Navigation bar & layout
│   └── ProtectedRoute.tsx  # Auth guard for routes
├── pages/            # Page components
│   ├── SignIn.tsx          # Login page
│   ├── SignUp.tsx          # Registration page
│   ├── EmployeeList.tsx    # List with search/filter/sort
│   └── EmployeeDetail.tsx  # Add/Edit/Delete form
├── services/         # API communication
│   └── api.ts              # Axios instance & endpoints
├── types/            # TypeScript type definitions
│   └── index.ts            # Employee, User interfaces
└── utils/            # Helper functions
    └── auth.ts             # Authentication helpers
```

Backend Technology Stack

Core:

- **InterSystems IRIS**: Multi-model database
- **ObjectScript**: IRIS native programming language
- **REST API**: %CSP.REST framework

Key Classes:

```
SEM Package
├── tblAccount        # User accounts (Persistent class)
├── tblEmployee       # Employee records (Persistent class)
└── SEMRESTAPI        # REST API handler (extends %CSP.REST)
```

ObjectScript Features Used:

- **Object Persistence**: `%Persistent` classes
- **SQL Queries**: `%SQL.Statement`
- **JSON Handling**: `%DynamicAbstractObject`
- **UTF-8 Conversion**: `$ZCONVERT(text, "I", "UTF8")`
- **DateTime**: `$ZDATETIME($HOROLOG, 3)`

API Communication Flow

Example: Adding Employee

1. User fills form → Click "Register"
   ↓
2. Frontend Validation (React)
   - Employee ID: 5 digits?
   - Name: not empty?
   - Sex: selected?
   ↓
3. Confirmation Dialog
   ↓
4. API Call (Axios)
   POST /sem/employee
   Body: { employeeId: "12345", name: "山田花子", ... }
   ↓
5. Vite Proxy
   http://localhost:5173/sem/employee
   → http://localhost:52773/sem/employee
   ↓
6. IRIS REST Handler
   SEM.SEMRESTAPI:CreateEmployee()
   ↓
7. Backend Validation
   - Required fields present?
   - Employee ID duplicate?
   ↓
8. UTF-8 Conversion
   $ZCONVERT(name, "I", "UTF8")
   ↓
9. Create Object
   newEmployee = ##Class(SEM.tblEmployee).%New()
   Set properties
   ↓
10. Save to Database
    status = newEmployee.%Save()
    ↓
11. Return Response
    { message: "Employee created successfully", id: 42 }
    ↓
12. Frontend Updates
    - Show success message
    - Navigate to employee list
    - Refresh table data

🎓 Key Technical Concepts to Explain

1. React Concepts

Component-Based Architecture:

- UI broken into reusable components
- Each component manages its own state
- Props for parent-child communication

Hooks:

- **useState**: Component state (e.g., form data, loading state)
- **useEffect**: Side effects (API calls, subscriptions)
- **useMemo**: Performance optimization (expensive calculations)

Routing:

- React Router for client-side navigation
- ProtectedRoute for authentication guard
- Dynamic routes: `/employees/:id`

2. TypeScript Benefits

Type Safety:

```typescript
interface Employee {
  id?: number;
  EmployeeId: string;
  Name: string;
  Sex: number;
  // ...
}

// Compile-time error prevention
function addEmployee(emp: Employee) {
  // TypeScript ensures emp has correct structure
}
```

Auto-completion:

- IDE suggests available properties

- Reduces runtime errors
- Improves code maintainability

3. IRIS Database Features

Multi-Model Database:

- Object database (persistent objects)
- Relational (SQL queries)
- Key-value, document, etc.

ObjectScript:

- Native IRIS language
- Object-oriented
- Direct database access

Performance:

- Fast data retrieval
- Transaction support
- Built-in web services

4. Soft Delete Pattern

Why Soft Delete?

- Data recovery possible
- Audit trail maintained
- Referential integrity preserved
- Regulatory compliance (data retention)

Implementation:

```objectscript
```

```
// Instead of physical delete
Do %DeleteId(id)

// Set flag
Set employee.deleteFlg = 1
Do employee.%Save()

// Query excludes deleted
WHERE deleteFlg = 0
```

5. Security Considerations

Authentication:

- Password stored (should be hashed in production)
- Session management via localStorage
- Protected routes on frontend

Validation:

- Client-side (UX improvement)
- Server-side (security requirement)
- Never trust client input

Recommendations for Production:

- Use bcrypt for password hashing
- Implement JWT tokens
- Add HTTPS
- CSRF protection
- Input sanitization

📊 Demo Success Criteria

Functional Requirements

- [ ] User can sign up with valid data
- [ ] User can log in with credentials
- [ ] User can add new employee

- [ ] User can search employees
- [ ] User can edit existing employee
- [ ] User can delete employee (soft delete)
- [ ] User can log out

Technical Requirements

- [ ] Frontend validation works correctly
- [ ] Backend validation prevents invalid data
- [ ] API communication successful
- [ ] Database operations execute correctly
- [ ] State management working properly
- [ ] Navigation and routing functional
- [ ] Error handling demonstrates gracefully

Presentation Requirements

- [ ] Explain architecture clearly
- [ ] Demonstrate each CRUD operation
- [ ] Highlight key technical concepts
- [ ] Answer questions confidently
- [ ] Stay within 5-6 minute timeframe

🚀 Troubleshooting Common Issues

Issue 1: Backend not running

Symptom: API calls fail, network errors
Solution: Start IRIS and ensure REST API accessible at localhost:52773

Issue 2: Frontend not connecting

Symptom: CORS errors, connection refused
Solution: Check vite.config.ts proxy settings, ensure backend URL correct

Issue 3: Duplicate Employee ID error

Symptom: Cannot add employee with existing ID
Solution: This is expected behavior! Change Employee ID or delete existing

Issue 4: Japanese characters not displaying

Symptom: Garbled text in database
Solution: Check UTF-8 conversion in backend: `$ZCONVERT(text, "I", "UTF8")`

Issue 5: Logout doesn't work

Symptom: Can still access protected routes
Solution: Ensure localStorage.clear() is called, check ProtectedRoute logic

---

💡 Additional Questions to Prepare For

Architecture Questions:

1. **"Why did you choose React for frontend?"**
   - Component reusability
   - Large ecosystem and community
   - Virtual DOM for performance
   - TypeScript support
2. **"Why IRIS instead of traditional databases?"**
   - Multi-model database capabilities
   - Fast performance for transaction systems
   - Built-in web services
   - Good for healthcare/financial applications
3. **"What are the security vulnerabilities?"**
   - Passwords not hashed (should use bcrypt)
   - No HTTPS in development
   - Client-side session (should use JWT)
   - No rate limiting on API

Implementation Questions:

1. **"How does the proxy work?"**
   - Vite dev server proxies /sem to IRIS backend
   - Avoids CORS issues in development
   - In production, use reverse proxy (nginx)
2. **"Why soft delete instead of hard delete?"**
   - Data recovery possible
   - Audit trail for compliance

- Maintains referential integrity

3. **"How would you scale this system?"**
    - Add Redis for session management
    - Implement caching layer
    - Database read replicas
    - API rate limiting
    - CDN for static assets

📝 Final Checklist Before Demo

Technical Setup:

- [ ] IRIS backend running and accessible
- [ ] Frontend dev server running (npm run dev)
- [ ] Browser opened to localhost:5173
- [ ] Database has some test data (optional)
- [ ] No console errors in browser

Presentation Materials:

- [ ] Test data prepared and accessible
- [ ] Architecture diagram ready (if showing slides)
- [ ] Code editor open (if showing code)
- [ ] Calm and confident mindset 😊

Backup Plans:

- [ ] Screenshots prepared (if demo fails)
- [ ] Video recording of working demo (backup)
- [ ] Written explanation ready

🎯 Summary: What Makes This Demo Strong

Technical Strengths:

✅ Full-stack implementation (Frontend + Backend + Database)
✅ Modern tech stack (React 18, TypeScript, IRIS)
✅ Complete CRUD operations
✅ Authentication system

✅ Soft delete pattern
✅ Client-side optimization (useMemo)
✅ Proper error handling

Demonstration Strengths:

✅ Clear flow: Signup → Login → Add → Edit → Delete
✅ Real-world scenario (employee management)
✅ Shows both add and edit in same component
✅ Search/filter functionality
✅ Confirmation dialogs (UX consideration)

Learning Outcomes:

✅ Understanding React component lifecycle
✅ State management with hooks
✅ REST API design and implementation
✅ Database operations with ObjectScript
✅ Full-stack data flow

Good luck with your presentation! You've got this! 🚀 💪