💻 Angular Frontend Deep Dive: Building the HR User Interface

This guide explains the architecture, components, services, and communication protocols used to build the responsive Angular client and connect it to the ASP.NET Core API.

1. Project Setup and Core Concepts

The Angular frontend is built as a **Single-Page Application (SPA)** using modern Standalone Components, ensuring a clean, modular structure.

1.1 Project Creation (Recap)

Action	Command (in HRApp_Project)	Key Angular Tools Used
Create Project	ng new hr-frontend --standalone --routing=false	**Angular CLI** and the **Standalone Component** architecture.
Styling	*Manual install and configuration*	**Bootstrap 5** for global styling and responsive UI components.

1.2 The Communication Bridge (Dependencies)

To communicate with the backend API, the frontend needs specific Angular modules:

Dependency	TypeScript Import	Purpose
HttpClient	import { HttpClient } from '@angular/common/http';	The primary Angular service used for making **HTTP requests** (GET, POST) to external APIs.
HttpClientModule	import { HttpClientModule } from '@angular/common/http';	This module must be imported into the main component (App) to enable the HttpClient service.

FormsModule	import { FormsModule } from '@angular/forms';	Enables Angular's two-way data binding ([(ngModel)]) for input elements like the search box.

2. Frontend Structure: Component and Service

The application uses two distinct classes: the EmployeeApiService (for communication) and the App component (for UI and state management).

2.1 The Data Interface (interface Employee)

This defines the contract for data flowing between the API (C#) and the frontend (TypeScript). It must mirror the C# model exactly.

```
// File: src/app/app.ts (Snippet)
interface Employee {
  id: number;
  fullName: string;
  department: string;
  hireDate: string; // The API sends DateTime as an ISO string
  salary: number;
}
```

2.2 The API Service (EmployeeApiService)

This class encapsulates all the backend communication logic, keeping the main component clean.

```
// File: src/app/app.ts (Snippet)
export class EmployeeApiService {
  private apiUrl = 'http://localhost:5120/api/employees';
  private http = inject(HttpClient); // Modern DI using inject()

  // GET: Fetches data from the API's retrieval endpoint
  getEmployees(searchTerm: string): Observable<Employee[]> {
    // Sends the request to: http://localhost:5120/api/employees?searchTerm=...
    return this.http.get<Employee[]>(`${this.apiUrl}?searchTerm=${searchTerm}`);
  }

  // POST: Sends the Excel file to the API's upload endpoint
  uploadFile(file: File): Observable<any> {
```

```
    const formData = new FormData();
    formData.append('file', file, file.name);
    // Sends the request to: http://localhost:5120/api/employees/upload
    return this.http.post<any>(`${this.apiUrl}/upload`, formData);
  }
}
```

Concept	Explanation
inject(HttpClient)	The modern Angular way to get a service instance. It provides the HttpClient instance needed for API calls.
Observable<Employee[]>	The standard type for asynchronous results in Angular. The component subscribes to this to receive data when it arrives.
FormData	A special JavaScript object used specifically to wrap and send files (binary data) over HTTP POST requests.

2.3 The Main Component (App Component)

The App component handles user interaction and manages the application state.

Concept	Explanation
@Component(...)	Defines the class as an Angular component.
standalone: true	Declares the component as independent, allowing it to manage its own dependencies (like HttpClientModule in the imports array).
providers: [...]	Makes services (EmployeeApiService) and pipes (DatePipe, CurrencyPipe) available for injection and use within this component and its children.

employees = signal(...)	A **Signal** is Angular's new state management tool, holding the current list of employee records and automatically notifying the template when the data changes.
constructor(private apiService: EmployeeApiService)	Gets the service instance via Dependency Injection.

3. Full-Stack Communication Flow

The entire application relies on the interaction between the Angular client (Frontend) and the ASP.NET Core API (Backend).

3.1 Initial Data Load (ngOnInit)

1. **Client Action:** The component loads (ngOnInit runs).
2. **Angular:** Calls apiService.getEmployees().
3. **HTTP Request:** Angular sends an **HTTP GET** request to http://localhost:5120/api/Employees.
4. **Backend (C#):** The GetEmployees method executes: it queries MS SQL Server, sorts the data by ID (OrderByDescending(e => e.Id)), and converts the C# Employee objects into **JSON** format.
5. **Angular:** The component receives the JSON data, updates the employees() signal, and the template instantly re-renders the sorted table rows.

3.2 Excel Upload

1. **Client Action:** User selects an Excel file and clicks "Upload."
2. **Angular:** The uploadFile() method creates a FormData object containing the file.
3. **HTTP Request:** Angular sends an **HTTP POST** request to http://localhost:5120/api/Employees/upload with the FormData.
4. **Backend (C#):** The Upload(IFormFile file) method executes:
   - It uses **EPPlus** to read the file stream.
   - It uses **robust parsing logic** to convert Excel's date/salary values into C# DateTime and decimal.
   - It saves the new records to **MS SQL Server** via EF Core.
5. **Angular:** The component receives the success message and immediately calls fetchEmployees() again to update the table with the new data.

4. Front-end Display Fixes

These steps were critical for a professional, correct UI display:

- **Sorting Fix:** The sorting issue was solved in the **backend controller** by forcing OrderByDescending(e => e.Id).
- **ID Display Fix:** Changed the template to format the ID number: {{ employee.id }}EMP (String interpolation).
- **Currency Fix:** Corrected the salaries to display Sri Lankan Rupees: {{ employee.salary \| currency:'LKR':'symbol':'1.2-2' }} (Angular's built-in CurrencyPipe).
- **Code Organization:** Separated custom styling from the component using styleUrls: ['./app.component.css'] for cleaner maintenance.

By understanding this complete flow, you are fully equipped to build and debug new full-stack applications!

Do you have any other final questions about the project, or would you like to explore adding a feature like a Delete button?