Students Affairs System

This document explains the **Students Affairs System** in a very simple way, written as **use cases and requirements** for a **junior JavaScript developer**.

The system is a small **web application** that manages data for:

- Students
- Courses
- Instructors
- Employees

The app uses:

- **JavaScript (ES6)**
- **Modules (import / export)**
- **json-server** as a fake backend
- **Fetch API** to get and send data
- **Grid (Table view)** with pagination and basic datatable features

Primary Actor

- **Employee (Student Affairs Staff)**

Managed Data (Entities)

- Student
- Course
- Instructor
- Employee

Note: Students, Courses, Instructors, and Employees are **data records**, not system users.

2. General System Description

The Students Affairs Staff uses the system to:

- View data in a table (grid view)
- Add new records
- Edit existing records
- Delete records
- Search, sort, and paginate data

All data is stored in a **JSON file** and accessed through **json-server REST APIs**.

3. Use Case List (High Level)

Use Case ID	Use Case Name
UC-01	View list of records
UC-02	Add new record
UC-03	Edit record
UC-04	Delete record
UC-05	Paginate records
UC-06	Search records
UC-07	Sort records

4. Use Case Details

UC-01: View List of Records

Actor: Student Affairs Staff

Description: View Students / Courses / Instructors / Employees in a grid (table).

Precondition:

- json-server is running
- Frontend application is loaded

Main Flow:

1. User opens a page (e.g. Students page).
2. System sends GET request using Fetch API.
3. System receives data from json-server.
4. System displays data in a table (grid view).

Postcondition:

- Data is visible in the table.

UC-02: Add New Record

Actor: Student Affairs Staff

Description: Add a new Student / Course / Instructor / Employee.

Main Flow:

1. User clicks **Add New** button.
2. System shows a form.
3. User fills the form and clicks **Save**.
4. System sends POST request to json-server.
5. System refreshes the table.

Postcondition:

- New record appears in the table.

UC-03: Edit Record

Actor: Student Affairs Staff

Description: Update existing data.

Main Flow:

1. User clicks **Edit** on a row.
2. System shows the record data in a form.
3. User updates values and clicks **Save**.
4. System sends PUT or PATCH request.
5. System updates the table.

Postcondition:

- Record data is updated.

UC-04: Delete Record

Actor: Student Affairs Staff

Description: Remove a record from the system.

Main Flow:

1. User clicks **Delete**.
2. System asks for confirmation.
3. User confirms.
4. System sends DELETE request.
5. System removes the row from table.

Postcondition:

- Record no longer exists.

UC-05: Paginate Records

Actor: Student Affairs Staff

Description: View data page by page.

Main Flow:

1. System loads first page of data.
2. User clicks **Next / Previous**.
3. System sends `GET` request with `_page` and `_limit`.
4. System displays new page.

Postcondition:

- Only one page of data is shown at a time.

UC-06: Search Records

Actor: Students Affairs Staff

Description: Search data using a keyword.

Main Flow:

1. User types in search box.
2. System sends `GET` request with `q=keyword`.
3. System displays filtered results.

Postcondition:

- Table shows matching records only.

UC-07: Sort Records

Actor: Student Affairs Staff

Description: Sort data by column.

Main Flow:

1. User clicks column header.
2. System sends `GET` request with `_sort` and `_order`.
3. System displays sorted data.

5. Functional Requirements (Simple)

- System must display data in a table.
- System must support CRUD operations.
- System must support pagination.
- System must support search and sort.
- System must fetch data using Fetch API.
- System must use ES6 modules.

6. Non-Functional Requirements (Simple)

- Code should be easy to read.
- Each feature should be in a separate JS file.
- Basic error handling (show message if request fails).

7. Technical Requirements

Frontend

- Class based ,OOP with JavaScript ES6
- HTML + CSS
- Modules (`import` / `export`)

Backend (Mock)

- json-server
- db.json file