

* We have a web application for students so they can use different features.
This is implemented with the visual studio code project named webapp

*We want to introduce a new feature which allows users to create and join study groups for different subjects.

Implementing the Feature:

1. Domain Model / Business Logic

- *Solution/TestApp/StudyGroup.cs*
 - Defines the *StudyGroup* entity with validation rules.
 - Implements *AddUser()* and *RemoveUser()* methods with duplicate/non-existent handling.
 - Validates: name length (5-30 chars), valid subjects (Math, Chemistry, Physics).
 - Stores creation date and user list (many-to-many navigation).
 - Added parameterless constructor and private setters for EF Core compatibility.
- *Solution/TestApp/User.cs*
 - Defines the *User* entity with *Id*, *Name*, and *StudyGroups* navigation properties.

2. Data Layer / Repository

- *Solution/TestAppAPI/AppDbContext.cs*
 - EF Core DbContext for MS SQL database.
 - Configures entities, unique index on *Subject*, many-to-many junction table "StudyGroupUsers".
- *Solution/TestAppAPI/IStudyGroupRepository.cs*
 - Interface defining the contract for data operations.
 - Methods: *CreateStudyGroup*, *GetStudyGroups* (with sort), *SearchStudyGroups* (with subject and sort), *JoinStudyGroup*, *LeaveStudyGroup*.
- *Solution/TestAppAPI/StudyGroupRepository.cs*
 - EF Core implementation (not in-memory; uses *AppDbContext*).
 - Enforces "one study group per subject" rule via unique index and check.
 - Handles sorting by creation date (ASC/DESC).
 - Manages user joins/leaves with existence and duplicate checks.

3. API / Presentation Layer

- *Solution/TestAppAPI/StudyGroupController.cs*
 - REST API endpoints:

- *POST /api/StudyGroup - Create new study group (validates input, handles duplicates/invalids).*
- *GET /api/StudyGroup?sort=asc/desc - Get all groups with sorting by creation date.*
- *GET /api/StudyGroup/search?subject=Math&sort=desc - Search/filter by subject with sorting.*
- *POST /api/StudyGroup/join?studyGroupId=1&userId=100 - Join a group (validates existence, prevents duplicates).*
- *POST /api/StudyGroup/leave?studyGroupId=1&userId=100 - Leave a group (validates membership).*
- *Solution/TestAppAPI/Program.cs*
 - *Configures dependency injection, EF Core, and API setup (not modified in changes).*

4. Test Layer

- *Solution/TestApp.Tests/StudyGroupUnitTests.cs*
 - *9 unit tests (NUnit) for StudyGroup domain logic: constructor validations (name, subject, date), add/remove users, null handling.*
- *Solution/TestAppAPI.Tests/StudyGroupControllerTests.cs*
 - *15 component tests (NUnit with Moq) covering API scenarios: creation (valid/duplicate/invalid), get all (with sorts), search (valid/invalid/with sort), join/leave (valid/duplicate/non-existent).*
- *Solution/TestAppAPI.Tests/StudyGroupRepositoryQueryTests.cs*
 - *1 component test (NUnit with EF Core SQLite in-memory) verifying the custom SQL query execution and results.*
- *Solution/ManualTests/StudyGroupE2ETests.md*
 - *Manual E2E test cases: create/view, join/leave/filter, sort in UI (with steps, inputs, expectations, regression flags).*

*This feature has an API which is part of the code.

API Implementation Files:

1. API Controller (HTTP Endpoints)

- *Solution/TestAppAPI/StudyGroupController.cs*
 - *Contains all REST API endpoints.*
 - *Handles HTTP requests/responses.*
 - *Maps routes and HTTP verbs (GET, POST).*
 - *Returns proper status codes (Ok, BadRequest, NotFound).*
 - *Uses dependency injection for the repository.*

2. API Project Configuration

- *Solution/TestAppAPI/Program.cs*
 - Configures the ASP.NET Core web API.
 - Sets up dependency injection (e.g., registers *IStudyGroupRepository*).
 - Registers services (controllers, Swagger if present).
 - Configures the HTTP pipeline (middleware, routing).

3. API Project File

- *Solution/TestAppAPI/TestAppAPI.csproj*
 - Defines the API project configuration.
 - References ASP.NET Core Web API framework (net8.0).
 - Includes NuGet packages (e.g., Microsoft.EntityFrameworkCore.SqlServer for DB, Swashbuckle.AspNetCore for Swagger/OpenAPI).

4. Repository Interface (API Contract)

- *Solution/TestAppAPI/IStudyGroupRepository.cs*
 - Defines the contract that the API controller uses.
 - Acts as abstraction layer between API and data.

5. Repository Implementation (API Backend)

- *Solution/TestAppAPI/StudyGroupRepository.cs*
 - Implements the business logic called by the API.
 - Handles data operations for API requests using EF Core.

API Endpoints Summary: All endpoints are defined in *StudyGroupController.cs* under the base route "api/StudyGroup" (case-sensitive):

* The system stores data in the repository, which is implemented as a MS SQL database. Although Subjects and Users are already existing in our data model, we want to introduce a new entity for StudyGroups, so you can also check below part of the code introduced for it.

The output goes here:

<http://localhost:5000/studygroup/debug/query-m-users>

The task

We need you to do the following as part of this task:

1. Write a list of test cases to check that this new feature satisfies the acceptance criteria provided bellow. In this list you must:
 - a. Describe some high-level steps and expectations (you can make assumptions of how the app works - just explain it at the beginning)
 - b. Highlight what are the inputs you will be using on each test case
 - c. Define testing level of these test cases: unit testing, component testing or e2e testing (manual) - considering that:
 - i. We have a unit test framework in TestApp using Nunit framework
 - ii. We have a component test framework in our TestAppAPI using Nunit framework
 - iii. We don't have any automation to test the UI, so manual testing will be required on e2e level
 - d. Indicate which test cases you would add to regression suite and which not
2. Write the code for all automated tests you described on the different frameworks
3. Write a SQL query that will return "all the StudyGroups which have at least one user with 'name' starting on 'M' sorted by 'creation date'" like "Miguel" or "Manuel"

There are multiple ways to approach this task, so as a first thing, explain us what approach you selected, the reasoning behind this decision and if you already had past experience implementing a similar approach or not

Acceptance criteria

1. There can be only one Study Group for a single Subject
 - a. The name for the Study Group with size between 5-30 characters
 - b. The only valid Subjects are: Math, Chemistry, Physics
 - c. We want to record when Study Groups were created
2. Users can join Study Groups for different Subjects
3. Users can check the list of all existing Study Groups
 - a. Users can also filter Study Groups by a given Subject
 - b. Users can sort to see most recently created Study Groups or oldest ones
4. Users can leave Study Groups they joined

API Controller class

```
using Microsoft.AspNetCore.Mvc;

namespace TestAppAPI
{
    public class StudyGroupController
    {
        private readonly IStudyGroupRepository _studyGroupRepository;

        public StudyGroupController(IStudyGroupRepository studyGroupRepository)
        {
            _studyGroupRepository = studyGroupRepository;
        }

        public async Task<IActionResult> CreateStudyGroup(StudyGroup studyGroup)
        {
            await _studyGroupRepository.CreateStudyGroup(studyGroup);
            return new OkResult();
        }

        public async Task<IActionResult> GetStudyGroups()
        {
            var studyGroups = await _studyGroupRepository.GetStudyGroups();
            return new OkObjectResult(studyGroups);
        }

        public async Task<IActionResult> SearchStudyGroups(string subject)
        {
            var studyGroups = await _studyGroupRepository.SearchStudyGroups(subject);
            return new OkObjectResult(studyGroups);
        }

        public async Task<IActionResult> JoinStudyGroup(int studyGroupId, int userId)
        {
            await _studyGroupRepository.JoinStudyGroup(studyGroupId, userId);
            return new OkResult();
        }

        public async Task<IActionResult> LeaveStudyGroup(int studyGroupId, int userId)
        {
            await _studyGroupRepository.LeaveStudyGroup(studyGroupId, userId);
            return new OkResult();
        }
    }
}
```

```
}
```

StudyGroup class

```
using System;
using System.Collections.Generic;

namespace TestApp
{
    public class StudyGroup
    {
        public StudyGroup(int studyGroupId, string name, Subject subject, DateTime createDate, List<User> users)
        {
            StudyGroupId = studyGroupId;
            Name = name;
            Subject = subject;
            CreateDate = createDate;
            Users = users;
        }

        //Some logic will be missing to validate values according to acceptance criteria, but imagine it is existing or do it
        public int StudyGroupId { get; }

        public string Name { get; }

        public Subject Subject { get; }

        public DateTime CreateDate { get; }

        public List<User> Users { get; private set; }

        public void AddUser(User user)
        {
            Users.Add(user);
        }

        public void RemoveUser(User user)
        {
            Users.Remove(user);
        }
    }

    public enum Subject
    {
        Math,
        Chemistry,
        Physics
    }
}
```

}

}