Study Group Management System

Project by: Angela Apostolska
Course: Internet Technologies
Framework: ASP.NET MVC 5 (.NET Framework 4.7.2)
Date: October 2025

Introduction

Overview

The Study Group Management System is a web-based application designed to facilitate collaborative learning among students. It provides a centralized platform where students can create, join, and manage study groups, schedule study sessions, and rate the effectiveness of completed sessions.

Purpose

The primary motivation behind this project is to address the challenge students face in organizing effective study sessions. Traditional methods of coordinating study groups through messaging apps or social media lack structure and often result in poor attendance and disorganized sessions. This system provides:

- **Centralized Group Management**: Students can easily discover and join study groups based on their subjects of interest
- **Session Scheduling**: Organized scheduling with capacity management to ensure productive sessions
- **Quality Tracking**: Rating system to evaluate session effectiveness and encourage quality participation
- **Academic Organization**: Subject-based categorization for better academic structure

Technology Stack

- **Backend Framework**: ASP.NET MVC 5 (.NET Framework 4.7.2)
- **ORM**: Entity Framework 6.4.4 (Code First approach)
- **Database**: SQL Server LocalDB
- **Frontend**: Razor Views, Bootstrap 5, jQuery
- **Language**: C# 7.3
- **Authentication**: Custom session-based authentication with role management
- **Validation**: Data Annotations and server-side validation

System Architecture and Design

Architecture overview

The application follows the **Model-View-Controller (MVC)** architectural pattern, which separates concerns into three interconnected components:

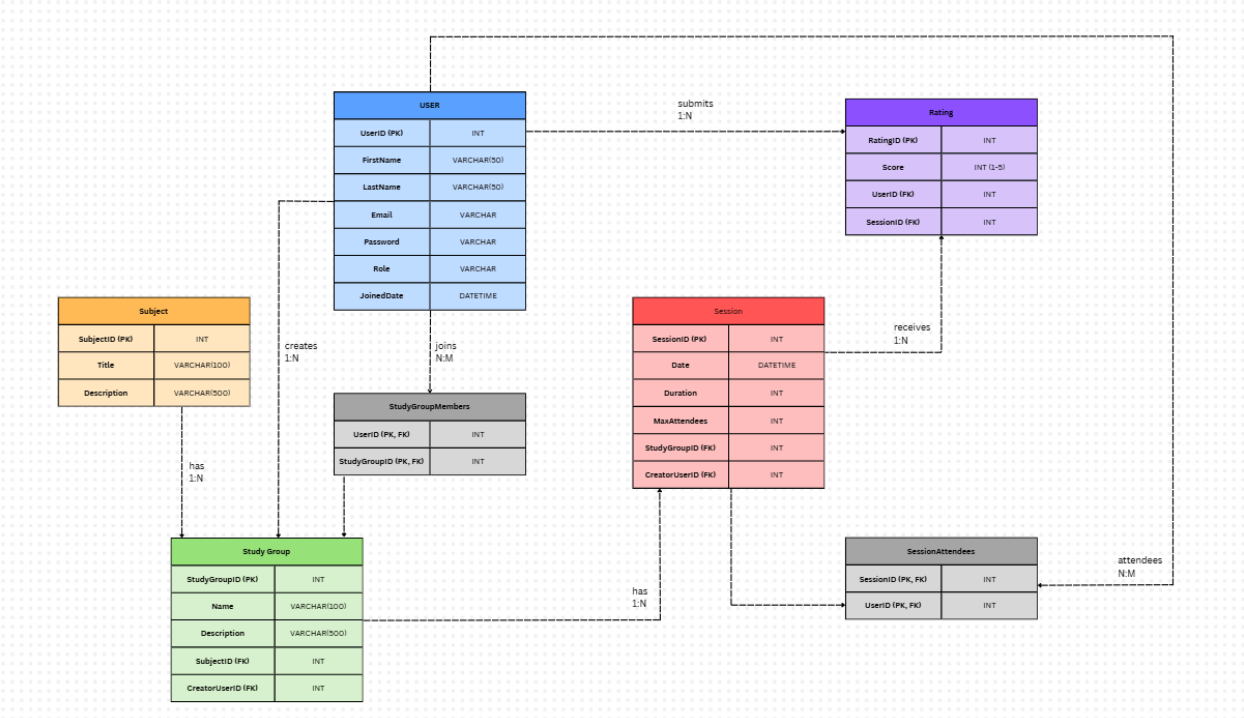
1. **Models**: Represent the application data and business logic (5 core entities + 3 view models)
2. **Views**: Handle the presentation layer and user interface (50+ razor views)
3. **Controllers**: Process incoming requests, manipulate models, and return views (8 controllers)

Project Structure

```
StudyGroups/
├── Controllers/          # Request handling and business logic coordination
│   ├── AccountController.cs      # Authentication and registration
│   ├── HomeController.cs         # Dashboard and main views
│   ├── StudyGroupsController.cs  # Study group management
│   ├── SessionsController.cs     # Session management
│   ├── SubjectsController.cs     # Subject CRUD operations
│   ├── RatingsController.cs      # Session rating system
│   ├── UsersController.cs        # User management (Admin)
│   └── SeederController.cs       # Database seeding utility
│
├── Models/               # Data entities
│   ├── User.cs
│   ├── StudyGroup.cs
│   ├── Subject.cs
│   ├── Session.cs
│   ├── Rating.cs
│   ├── UserDashboardViewModel.cs      // ViewModel
│   ├── AdminDashboardViewModel.cs  //ViewModel
│   └── UserListViewModel.cs              //ViewModel
│
├── Views/                # Razor view templates
│   ├── Account/          # Login and registration views
│   ├── Home/             # Dashboard views
│   ├── StudyGroups/      # Study group views
│   ├── Sessions/         # Session management views
│   ├── Subjects/         # Subject views
│   ├── Ratings/          # Rating views
│   ├── Users/            # User management views
│   └── Shared/           # Layout and shared components
```

```
│
├── DAL/                    # Data Access Layer
│   ├── StudyGroupContext.cs      # DbContext configuration
│
│
├── Filters/                # Custom action filters
│   └── SessionAuthorizeAttribute.cs  # Authorization filter
│
└── Migrations/             # EF Code First migrations
```

ER Diagram



Database Design

Tables overview

The project has 5 core tables in total, corresponding to the Model classes.

Table	**Purpose**
Users	Student and Admin accounts

Subjects	Topics for studygroups
StudyGroups	Collaborative learning groups
Sessions	Scheduled study meets
Ratings	Session feedback

Additionally there are 2 Junction Tables from the M:N relationships.

Table	**Purpose**
StudyGroupMembers	Links users to groups they've joined
SessionAttenees	Links users to sessions they're attending

Relationships

#	**Relationship**	**Type**	**Cardinality**	**Junction Table**	**Purpose**
1	User → StudyGroup (Creator)	Direct	1:N	-	Ownership tracking
2	User ←→Study Group (Member)	Indirect	N:M	StudyGroup Members	Membership Tracking
3	Subject → StudyGroup	Direct	1:N	-	Subject Categorizatio n
4	StudyGroup → Session	Direct	1:N	-	Group Activities
5	User → Session (Creator)	Direct	1:N	-	Session organization
6	User ←→Session (Attendee)	Indirect	N:M	SessionAtten dees	Attendance tracking
7	User → Rating	Direct	1:N	-	Feedback authorship

8	Session → Rating	Direct	1:N	-	Feedback collection

Database Configuration (StudyGroupContext)

The *OnModelCreating* method configures relationships and prevents circular cascade paths:

```
modelBuilder.Entity<User>()
    .HasMany(u => u.CreatedStudyGroups)
    .WithRequired(sg => sg.Creator)
    .HasForeignKey(sg => sg.CreatorUserID)
    .WillCascadeOnDelete(false);
```

This configuration ensures that:

- Deleting a user doesn't delete their created groups/sessions
- Deleting a study group removes all associated sessions
- Membership tables cascade properly

Features and Functionalities

Core Functionalities

As a Guest (Not Logged In), You Can:

- **Register** a new account with your first name, last name, email, and password
- **Login** to access the system with your email and password

As a Regular User, You Can:

Manage Your Profile

- View your personal dashboard with statistics about:
  - Number of subjects you're studying
  - Total study sessions you've participated in
  - Total time spent studying
  - Your average rating across sessions

Work with Subjects

- **Browse** all available subjects
- **View** detailed information about any subject
- **Search** for subjects by title or description

Manage Study Groups

- **Create** new study groups by choosing a subject, name, and description

- **View** all study groups (all groups, ones you created, ones you joined, or available groups)
- **Edit** study groups that you created
- **Delete** study groups that you created
- **Join** study groups created by other users
- **Leave** study groups that you're a member of (but didn't create)
- **Search** for study groups by name, description, or subject
- **Filter** study groups by subject

Manage Study Sessions

- **Create** new study sessions for groups you created or joined
    - Set the date, time, duration, and maximum number of attendees (1-10)
- **View** upcoming sessions from your groups
- **Edit** sessions that you created (only before they finish)
- **Delete** sessions that you created
- **Join** sessions as an attendee (if not full and you're not the creator)
- **Leave** sessions you've joined
- **Search** for sessions by study group, subject, or creator name
- **Filter** sessions by study group

Rate Sessions

- **Rate** completed sessions that you created or attended
    - Give a score from 1 to 5 stars
    - Can only rate after the session has ended
    - Can rate each session only once
- **View** ratings given by other participants
- **Edit** your own ratings
- **Delete** your own ratings

As an Admin, You Can:

Manage Subjects

- **Create** new subjects with title and description
- **View** all subjects
- **Edit** any subject's information
- **Delete** subjects from the system

View Platform Statistics

- Access the admin dashboard showing:
    - Total number of subjects
    - Total number of registered users
    - Total number of study groups
    - Total number of sessions

Manage Users

- **View** a list of all registered users with their:

    - Name and email
    - Number of study groups they're involved in
    - Number of sessions they've participated in
    - Join date

Note: Admins cannot create, join, or manage study groups and sessions—these features are exclusively for regular users.

Advanced Functionalities

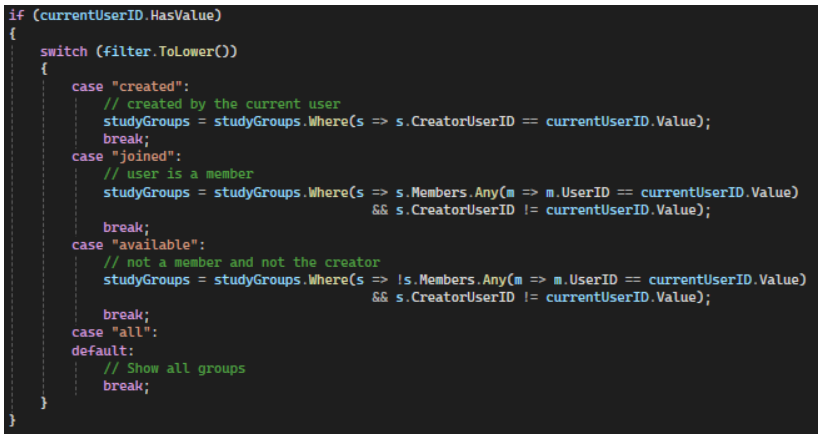
Search and Filtering System

Study Groups Index

- **Text Search**: Search by group name, description, or subject title
- **Subject Filter**: Dropdown to filter by specific subject

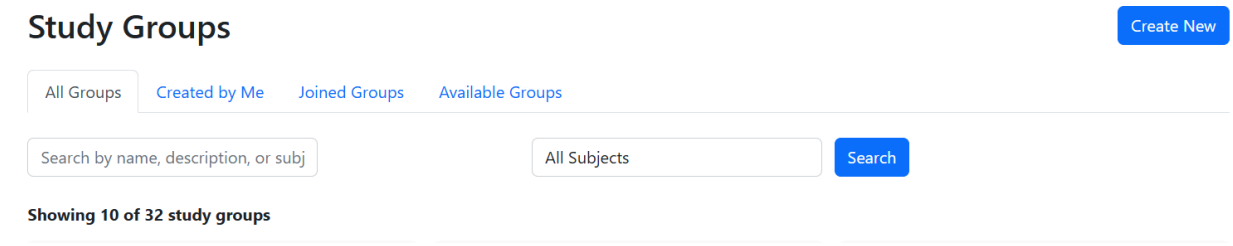
```
if (subjectId.HasValue && subjectId.Value > 0)
{
    studyGroups = studyGroups.Where(s => s.SubjectID == subjectId.Value);
}
```

- **Category Filters** (Tab-based navigation):
    - **All Groups**: Shows all available study groups
    - **Created by Me**: Groups where current user is the creator
    - **Joined Groups**: Groups where user is a member (not creator)
    - **Available Groups**: Groups user hasn't joined yet



```
if (currentUserID.HasValue)
{
    switch (filter.ToLower())
    {
        case "created":
            // created by the current user
            studyGroups = studyGroups.Where(s => s.CreatorUserID == currentUserID.Value);
            break;
        case "joined":
            // user is a member
            studyGroups = studyGroups.Where(s => s.Members.Any(m => m.UserID == currentUserID.Value)
                                                 && s.CreatorUserID != currentUserID.Value);
            break;
        case "available":
            // not a member and not the creator
            studyGroups = studyGroups.Where(s => !s.Members.Any(m => m.UserID == currentUserID.Value)
                                                 && s.CreatorUserID != currentUserID.Value);
            break;
        case "all":
        default:
            // Show all groups
            break;
    }
}
```

- **Pagination**: 10 items per page with page navigation
- **Filter Persistence**: Search and filter parameters maintained during pagination



Sessions Index

- **Text Search**: Search by study group name, subject title, or creator name

```
// search filtering
if (!String.IsNullOrEmpty(searchString))
{
    sessions = sessions.Where(s =>
        s.StudyGroup.Name.Contains(searchString) ||
        s.StudyGroup.Subject.Title.Contains(searchString) ||
        s.Creator.FirstName.Contains(searchString) ||
        s.Creator.LastName.Contains(searchString)
    );
}
```

- **Study Group Filter**: Dropdown showing only groups where user is creator or member

```
if (currentUserID.HasValue)
{
    var userStudyGroups = db.StudyGroups
    .Include(sg => sg.Members)
    .Where(sg => sg.CreatorUserID == currentUserID.Value ||
                 sg.Members.Any(m => m.UserID == currentUserID.Value))
    .OrderBy(sg => sg.Name)
    .ToList();

    ViewBag.StudyGroups = new SelectList(userStudyGroups, "StudyGroupID", "Name", studyGroupId);
}
```

- **Status Filter**: Automatically shows only upcoming (unfinished) sessions

```
var upcomingSessions = sessions.Where(s => !s.IsFinished).OrderBy(s => s.Date).ToList();
```

- **Pagination**: 10 items per page

```
// pagination
int pageSize = 10;
int pageNumber = (page ?? 1);

var totalItems = upcomingSessions.Count();
var totalPages = (int)Math.Ceiling((double)totalItems / pageSize);

var pagedSessions = upcomingSessions
    .Skip((pageNumber - 1) * pageSize)
    .Take(pageSize)
    .ToList();
```

- **Filter Persistence**: Maintains search parameters across pages

Subjects Index

- Text search across subject title and description
- Instant filtering with search button
- Clear button to reset search

Pre-filled forms

Study Group Creation from Subject → Create study group with pre-selected subject

User flow:

1. User views subject details page
2. Clicks "Create Study Group for this Subject"
3. Redirected to create form with subject pre-filled and locked

```
// GET: StudyGroups/CreateForSubject/5
[SessionAuthorize(Roles = "User")]
0 references
public ActionResult CreateForSubject(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

    }

    Subject subject = db.Subjects.Find(id);
    if (subject == null)
    {
        return HttpNotFound();
    }

    // Create a new StudyGroup with pre-filled subject
    var studyGroup = new StudyGroup
    {
        SubjectID = subject.SubjectID
    };

    // Pass the subject details to the view
    ViewBag.SubjectTitle = subject.Title;
    ViewBag.SubjectID = subject.SubjectID;
    /// flag so that we can use the same create view
    ViewBag.IsSubjectLocked = true;

    return View("Create", studyGroup);

}
```

Session Creation from Study Group → Create session with pre-selected study group

User flow:

1. User views study group details page (as creator or member)
2. Clicks "Create Session" button
3. Form opens with study group pre-filled and locked

Access Control:

1. Only creator or members can access this feature

```
// GET: Sessions/CreateForStudyGroup/5
[SessionAuthorize(Roles = "User")]
0 references
public ActionResult CreateForStudyGroup(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    StudyGroup studyGroup = db.StudyGroups
        .Include(sg => sg.Subject)
        .Include(sg => sg.Members)
        .FirstOrDefault(sg => sg.StudyGroupID == id);

    if (studyGroup == null)
    {
        return HttpNotFound();
    }

    // check if the current user is creator or member
    int currentUserID = (int)Session["UserID"];
    bool isCreator = studyGroup.CreatorUserID == currentUserID;
    bool isMember = studyGroup.Members != null && studyGroup.Members.Any(m => m.UserID == currentUserID);

    if (!isCreator && !isMember)
    {
        TempData["Error"] = "You must be the creator or a member of this study group to create sessions.";
        return RedirectToAction("Details", "StudyGroups", new { id = id });
    }
```

2. Additional checks to check whether the new session that gets added overlaps with any other session created for that same study group

```
int currentUserID = (int)Session["UserID"];
bool isCreator = studyGroup.CreatorUserID == currentUserID;
bool isMember = studyGroup.Members != null && studyGroup.Members.Any(m => m.UserID == currentUserID);

if (!isCreator && !isMember)
{
    TempData["Error"] = "You must be the creator or a member of this study group to create sessions.";
    return RedirectToAction("Details", "StudyGroups", new { id = session.StudyGroupID });
}

// validation check for time overlapping between sessions
DateTime sessionEndTime = session.Date.AddMinutes(session.Duration);

var existingSessions = db.Sessions
    .Where(s => s.StudyGroupID == session.StudyGroupID)
    .ToList();

bool isOverlapping = existingSessions.Any(s =>
{
    DateTime existingEndTime = s.Date.AddMinutes(s.Duration);
    return (session.Date >= s.Date && session.Date < existingEndTime) ||
           (sessionEndTime > s.Date && sessionEndTime <= existingEndTime) ||
           (session.Date <= s.Date && sessionEndTime >= existingEndTime);
});


if (isOverlapping)
{
    ModelState.AddModelError("Date", "A session already exists at this time for this study group. Sessions cannot overlap.");
}
```

Session and Group Membership Management

Join Study Group → users can join available study groups

Process:

1. Button is visible only on groups where the user is not a creator or a member
2. POST action method that adds the user to the members collection
3. Success/Error messages feedback via TempData
4. Redirect Back to group details page

```
// POST: StudyGroups/Join/5
[HttpPost]
0 references
public ActionResult Join(int id)
{
    if (Session["UserID"] == null)
    {
        return RedirectToAction("Login", "Account");
    }

    int currentUserID = (int)Session["UserID"];
    var studyGroup = db.StudyGroups
        .Include(sg => sg.Members)
        .FirstOrDefault(sg => sg.StudyGroupID == id);

    if (studyGroup == null)
    {
        return HttpNotFound();
    }

    // check if the user is already a member or the creator
    if (studyGroup.CreatorUserID == currentUserID ||
        (studyGroup.Members != null && studyGroup.Members.Any(m => m.UserID == currentUserID)))
    {
        TempData["Error"] = "You are already a member or the creator of this study group.";
        return RedirectToAction("Details", new { id = id });
    }

    var user = db.Users.Find(currentUserID);
    studyGroup.Members.Add(user);
    db.SaveChanges();
    TempData["SuccessMessage"] = $"You joined '{studyGroup.Name}' successfully!";
    return RedirectToAction("Details", new { id });
}
```

Validation:

1. If the user is already a member or a creator, he doesn't have access to this action.

```
// check if the user is already a member or the creator
if (studyGroup.CreatorUserID == currentUserID ||
    (studyGroup.Members != null && studyGroup.Members.Any(m => m.UserID == currentUserID)))
{
    TempData["Error"] = "You are already a member or the creator of this study group.";
    return RedirectToAction("Details", new { id = id });
}
```

Leave Study Group → members can leave study groups that they've joined

Process:

1. Available only to members (users that joined, not creators)
2. Removes users from Members collection

```
// POST: StudyGroups/Leave/5
[HttpPost]
0 references
public ActionResult Leave(int id)
{
    if (Session["UserID"] == null)
    {
        return RedirectToAction("Login", "Account");
    }

    int currentUserID = (int)Session["UserID"];
    var studyGroup = db.StudyGroups
        .Include(sg => sg.Members)
        .FirstOrDefault(sg => sg.StudyGroupID == id);

    if (studyGroup == null)
    {
        return HttpNotFound();
    }

    // check if the user is a member
    var user = studyGroup.Members.FirstOrDefault(m => m.UserID == currentUserID);
    if (user != null)
    {
        studyGroup.Members.Remove(user);
        db.SaveChanges();
        TempData["SuccessMessage"] = $"You left '{studyGroup.Name}' successfully!";
    }
    return RedirectToAction("Index");
}
```

Join Sessions → users can join upcoming sessions with capacity

Validation:

1. Creator cannot join its own session
2. You cannot join the session if you already are an attendee
3. You cannot join the session if it's max capacity is reached

```
// Check if user is the creator
if (session.CreatorUserID == currentUserID)
{
    TempData["Error"] = "You cannot join your own session.";
    return RedirectToAction("Details", new { id });
}

// Check if already attending
if (session.Attendees.Any(a => a.UserID == currentUserID))
{
    TempData["Error"] = "You are already attending this session.";
    return RedirectToAction("Details", new { id });
}

// Check if session is full
if (session.Attendees.Count >= session.MaxAttendees)
{
    TempData["Error"] = "This session is full.";
    return RedirectToAction("Details", new { id });
}
```

Leave Sessions → Attendees can leave sessions before they start
The users can leave sessions before they start, only if they were ever an attendee (added in the collection of attendees of that session).

```
// POST: Sessions/Leave/5
[HttpPost]
0 references
public ActionResult Leave(int id)
{
    if (Session["UserID"] == null)
    {
        return RedirectToAction("Login", "Account");
    }

    int currentUserID = (int)Session["UserID"];
    var session = db.Sessions
        .Include(s => s.Attendees)
        .FirstOrDefault(s => s.SessionID == id);

    if (session == null)
    {
        return HttpNotFound();
    }

    var user = session.Attendees.FirstOrDefault(a => a.UserID == currentUserID);
    if (user != null)
    {
        session.Attendees.Remove(user);
        db.SaveChanges();
        TempData["Success"] = "Successfully left the session.";
    }

    return RedirectToAction("Index");
}
```

Rating System

Leave rating from Sessions Details → Quick rating submission directly from a sessions details page
Validation:

1. Session must be finished (calculated)

```
// verifying if the session has ended
var session = db.Sessions
    .Include(s => s.Attendees)
    .FirstOrDefault(s => s.SessionID == sessionId);

if (session == null)
{
    TempData["Error"] = "Session not found.";
    return RedirectToAction("Index", "Sessions");
}
DateTime sessionEndTime = session.Date.AddMinutes(session.Duration);
if (DateTime.Now <= sessionEndTime)
{
    TempData["Error"] = "You can only rate a session after it has ended.";
    return RedirectToAction("Details", "Sessions", new { id = sessionId });
}
```

2. User must be creator or attendee

```
// check if the logged in user is creator or attendee
bool isCreator = session.CreatorUserID == currentUserID;
bool isAttendee = session.Attendees.Any(a => a.UserID == currentUserID);

if (!isCreator && !isAttendee)
{
    TempData["Error"] = "Only the creator and attendees can rate this session.";
    return RedirectToAction("Details", "Sessions", new { id = sessionId });
}
```

3. User hasn't already rated the session

```
// check if the logged in user has already voted
if (db.Ratings.Any(r => r.SessionID == sessionId && r.UserID == currentUserID))
{
    TempData["Error"] = "You have already rated this session.";
    return RedirectToAction("Details", "Sessions", new { id = sessionId });
}
```

Dashboards

User Dashboard

Purpose:

- The User Dashboard provides personalized statistics and information for individual users about their study activity within the platform.

Statistics Displayed:

- User Profile: Their initials, full name, email address, joined date
- Study Statistics: their subjects studied, total sessions, time spent studying based on the sessions they have attended, average rating they left on sessions that they were attending.

The main functions are written in the Index() of the HomeController.cs. Using the UserDashboardViewModal we aggregate data from multiple database tables, such as the user table, study groups table, sessions table and rating table, in order to show statistics tailored to that user at an end view point.

Examples of queries:

```
TotalSubjects = db.StudyGroups
    .Where(sg => sg.CreatorUserID == currentUserID)
    .Select(sg => sg.SubjectID)
    .Union(db.StudyGroups
        .Where(sg => sg.Members.Any(m => m.UserID == currentUserID))
        .Select(sg => sg.SubjectID))
    .Distinct()
    .Count(),

TotalSessions = db.Sessions.Count(s => s.CreatorUserID == currentUserID)
                + db.Sessions.Count(s => s.Attendees.Any(a => a.UserID == currentUserID) && s.CreatorUserID != currentUserID),

TotalTimeStudied = (db.Sessions
            .Where(s => s.CreatorUserID == currentUserID)
            .Sum(s => (int?)s.Duration) ?? 0) +
            (db.Sessions
            .Where(s => s.Attendees.Any(a => a.UserID == currentUserID) && s.CreatorUserID != currentUserID)
            .Sum(s => (int?)s.Duration) ?? 0),

AverageRating = db.Ratings
                .Where(r => r.UserID == currentUserID)
                .Select(r => (double?)r.Score)
                .DefaultIfEmpty(0)
                .Average() ?? 0
```

Admin Dashboard

Purpose:

- The Admin Dashboard provides system-wide statistics for administrators to monitor overall platform usage and growth.

Statistics Displayed:

- Total Subjects
- Total Users
- Total Study groups created
- Total sessions

The main functions are written in the AdminDashboard() action result method inside the HomeController.cs. Using the AdminDashboardViewModel we aggregate platform-wide counts from the users table (total users), the studygroups table (total study groups), the sessions table (total sessions) and the subjects table (total subjects), which we show in the end view for the user.

Database Seeder

The Database Seeder is a utility that populates the database with realistic fake data for testing and development purposes. It uses the **Bogus** library to generate random but realistic data while maintaining referential integrity with existing database records.

Architecture Components:

1. SeederController.cs → The MVC controller that provides the user interface for triggering the seeding process.
2. DatabaseSeeder.cs → The core seeding logic that generates and inserts data into the database.
3. Index.cshtml → The view that displays the seeder interface and feedback messages.

Flow of seeding the DB:

1. Click on the “Seed Database” button on the View/Seeder/Index file
2. SeederController.SeedData() [POST] method gets called
3. Creates a DatabaseSeeder instance with connection string
4. DatabaseSeeder.SeedDatabase()
    a. Phase 1: Loads existing data (needed for loading the initial users i had created and the existing subjects that i manually entered with a sql query)
    b. Phase 2: Generate and insert study groups
    c. Phase 3: Generate and insert sessions
    d. Phase 4: Generate and insert ratings

    *The process is divided into phases since in order to create a study group or a session, you must have created users and subjects beforehand.

Authorization

SessionAuthorizeAttribute is a custom action filter that provides session-based authentication and role-based authorization for controller actions. It extends ActionFilterAttribute to intercept action execution and verify user credentials before allowing access.
Purpose:

- Ensures that users are authenticated before accessing protected resources
- Enforce role-based access control for specific actions
- Provide automatic redirects for unauthorized access attempts

Usage examples:

1. [SessionAuthorize] → requires any authenticated user

```
[SessionAuthorize]
public ActionResult Index()
{
// Only logged-in users can access
}
```

2. [SessionAuthorize=(Roles = “Admin)]

```
[SessionAuthorize(Roles=”Admin”)]
public ActionResult Index()
{
// Only admin role can access
}
```

Role-Based Authorization Visual

Feature	**Anonymus**	**User**	**Admin**
Authentication	✅ Login/Register	✅ Logout	✅ Logout
Dashboard	❌	✅ User Dashboard	✅ Admin Dashboard
Subjects	❌	✅ View Only	✅ Full CRUD
StudyGroups	❌	✅ Full CRUD*	✅ View Only
Sessions	❌	✅ Full CRUD*	✅ View Only
Ratings	❌	✅ Full CRUD*	✅ View Only
User Management	❌	❌	✅ View Only
Join/Leave Groups	❌	✅	❌
Join/Leave Sessions	❌	✅	❌

***CRUD with ownership restrictions** - Users can only edit/delete resources they created

Controller-Level Authorization

Controller	**Filter Applied**
HomeController	[SessionAuthorize]
SubjectsController	[SessionAuthorize]
StudyGroupsController	[SessionAuthorize]
SessionsController	[SessionAuthorize]
RatingsController	[SessionAuthorize]
UsersController	[SessionAuthorize(Roles = "Admin")]
AccountController	No filter (public)

Additional Access Control

1. **Ownership Validation**
   Applied to Edit/Delete operations for study groups, sessions and ratings

   ```
   if (resource.CreatorUserID != currentUserID) {
   TempData["Error"] = "You can only edit/delete items you created.";
   return RedirectToAction("Details", new { id });
   }
   ```
2. **Membership Validation**
   Only members and creators can edit/delete sessions or study groups that they've been a part of.
3. **Business Rules**

Rule	**Applies To**
Cannot edit finished sessions	Sessions.Edit
Can only rate after session ends	Rating.CreateFromSession
Cannot rate twice	Rating.CreateFromSession
Cannot join own session	Session.Join
Session capacity limits	Session.Join

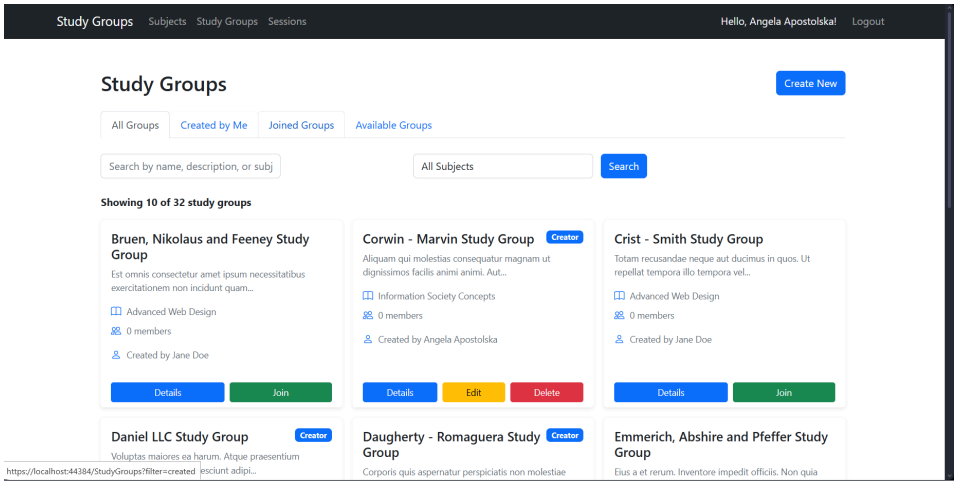
Unauthorized Access Handling

- Not Authenticated → redirects to Account/Login
- Wrong Role → redirects to Home/Unauthorized
- Not owner → redirects to Details/Index, and shows error message

Design Approach

The application follows a modern, card-based design system built on **Bootstrap 5**, emphasizing clarity, consistency, and user-friendly interactions across all views.

1. Consistent layout structure using container my-5 classes
2. Card Based UI providing clear UI and better visual hierarchy
3. Visual Feedback and Status Indicatiors
    a. Badges —> for status like creator, member, full, finished, upcoming
    b. Alert components
    c. Toast notifications for CRUD feedback actions
    d. Hover states



Future Enhancements

1. Real-Time Communication
    - Live chat with study groups
    - Real time notifications when someone new joins, new sessions posting, session reminders
    - Online/Offline User status

2. Sharing System
    - File upload system for materials
    - File preview functionality
3. Calendar Integration
    - Visual calendar representation of sessions slots
    - Export options to personal calendars
    - Drag and drop session scheduling
4. Achievement System
    - Badges → user could collect badges by completing some activities like creating their first session, or joining 10 groups, completing x number of sessions etc..
5. OAuth 2.0 authentication using google or microsoft accounts for the users which will ease out the registration process, and provide additional verification check whether the user is actually an user

Conclusion

The Study Group Management System successfully addresses the challenge of organizing collaborative learning by providing a comprehensive platform for students to create, join, and manage study sessions. Throughout this project, I have implemented a full-stack web application using ASP.NET MVC 5 that demonstrates proficiency in:

Technical Implementation:

- Complex database design with multiple many-to-many relationships
- Custom authentication and role-based authorization system
- Advanced filtering and search functionality
- Real-time capacity management
- Comprehensive CRUD operations across five main entities

User-Centric Features:

- Intuitive dashboards for both users and administrators
- Context-aware UI that adapts to user roles and relationships
- Seamless join/leave mechanisms for groups and sessions
- Session rating system for quality assurance
- Pagination and filtering for improved usability

Code Quality:

- Separation of concerns through MVC pattern
- Eager loading to prevent performance issues
- Comprehensive validation at model and business logic levels
- Reusable components and DRY principles
- Clear error messaging and user feedback