Contents

[**------Search and Sort ( New )--------** 1](#_Toc164638524)

[**ApplicationDbContext:** 1](#_Toc164638525)

[**Controller:** 2](#_Toc164638526)

[**Index:** 7](#_Toc164638527)

[**1. bỏ dòng này vào phía trước tất cả các Index** 7](#_Toc164638528)

[**2. Bỏ tiếp cái này vào và nhớ chỉ cần chỉnh cái tên asp-action thôi để nó chạy. Tên thì có trong controller như trên đó** 7](#_Toc164638529)

[**3. Bỏ cái này để sử dụng sort. Nút sort có sẵn bị hư và không bấm được** 10](#_Toc164638530)

[**SEARCH** 14](#_Toc164638531)

[--------------------------**Comment--**----------------------------- 16](#_Toc164638532)

[**--------------------------Like** ------------------------------- 18](#_Toc164638533)

[**--------------------------Sort** ------------------------------- 19](#_Toc164638534)

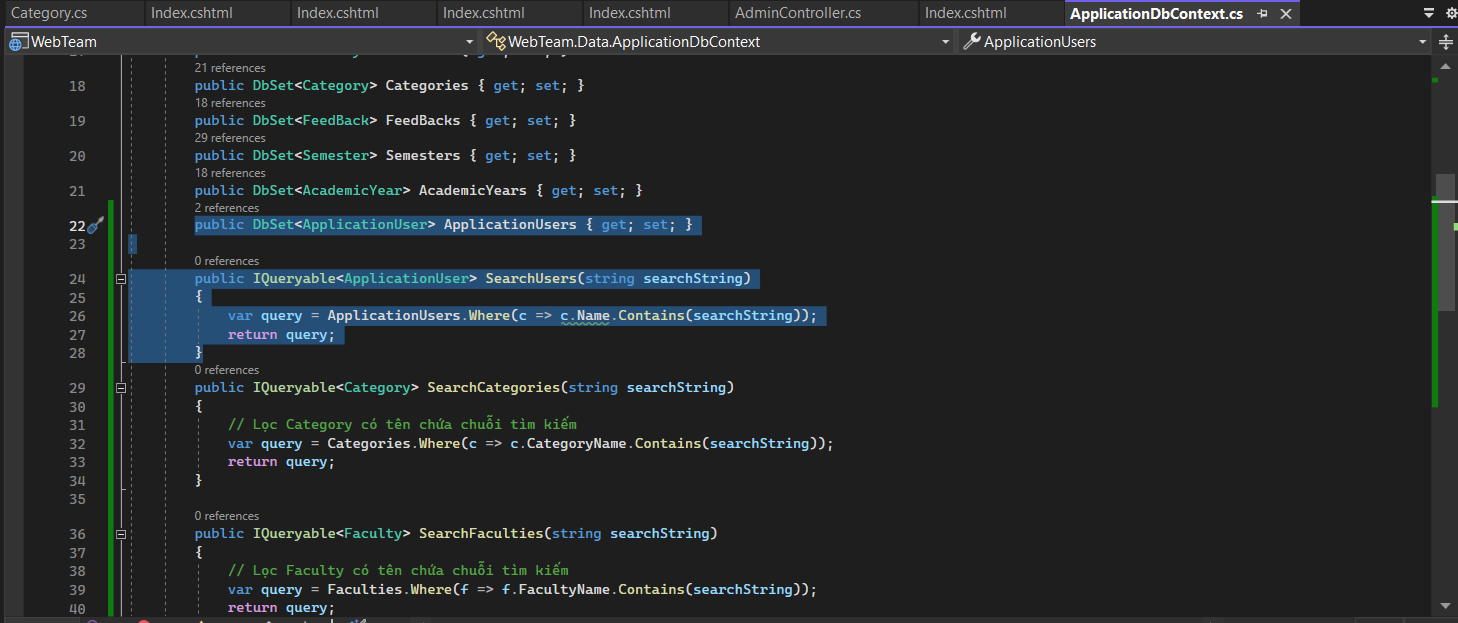
[--------------------------**Set up limit 14days to Feedback and lock -------------------------------** 20](#_Toc164638535)

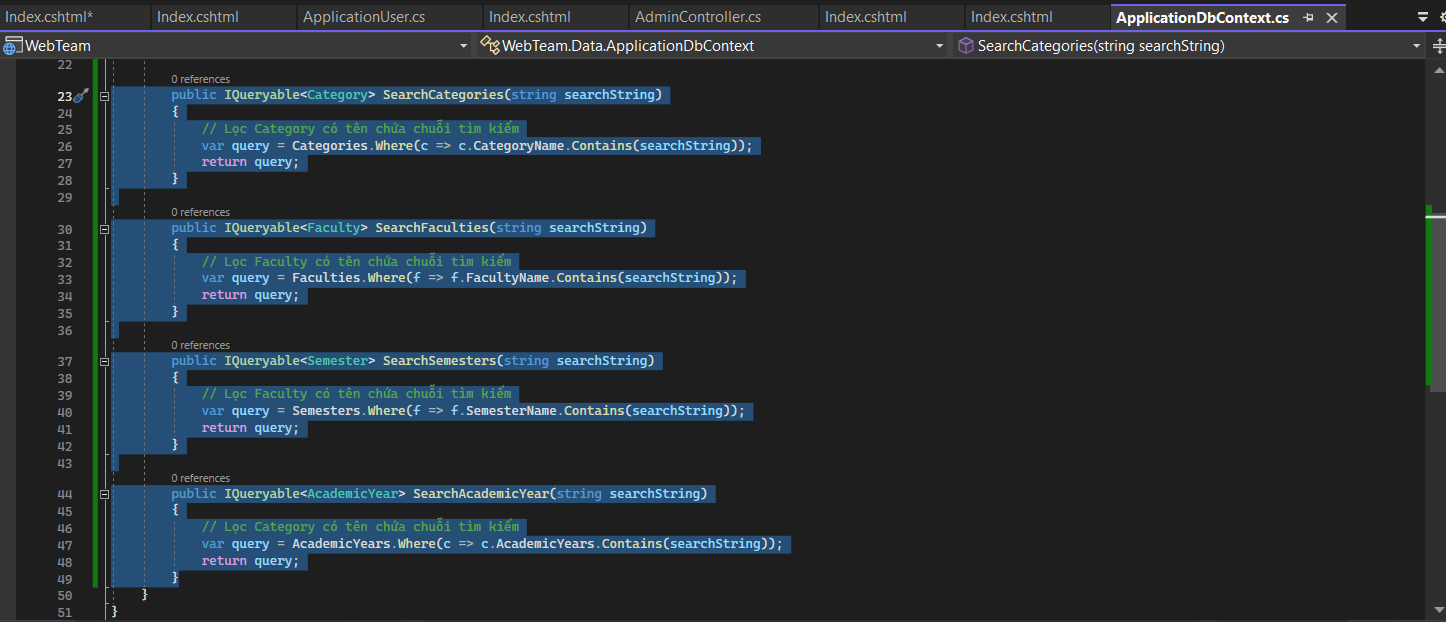
[**--------------------------Upload Image ( + Limit 5MB) -------------------------------** 21](#_Toc164638536)

[**--------------------------Limit day for dl1 and dl2**------------------------------- 22](#_Toc164638537)

# **------Search and Sort ( New )--------**

## **ApplicationDbContext:**

****

****

**Code:**

public DbSet<ApplicationUser> ApplicationUsers { get; set; }

public IQueryable<ApplicationUser> SearchUsers(string searchString)

{

var query = ApplicationUsers.Where(c => c.Name.Contains(searchString));

return query;

}

public IQueryable<Category> SearchCategories(string searchString)

{

// Lọc Category có tên chứa chuỗi tìm kiếm

var query = Categories.Where(c => c.CategoryName.Contains(searchString));

return query;

}

public IQueryable<Faculty> SearchFaculties(string searchString)

{

// Lọc Faculty có tên chứa chuỗi tìm kiếm

var query = Faculties.Where(f => f.FacultyName.Contains(searchString));

return query;

}

public IQueryable<Semester> SearchSemesters(string searchString)

{

// Lọc Faculty có tên chứa chuỗi tìm kiếm

var query = Semesters.Where(f => f.SemesterName.Contains(searchString));

return query;

}

public IQueryable<AcademicYear> SearchAcademicYear(string searchString)

{

// Lọc Category có tên chứa chuỗi tìm kiếm

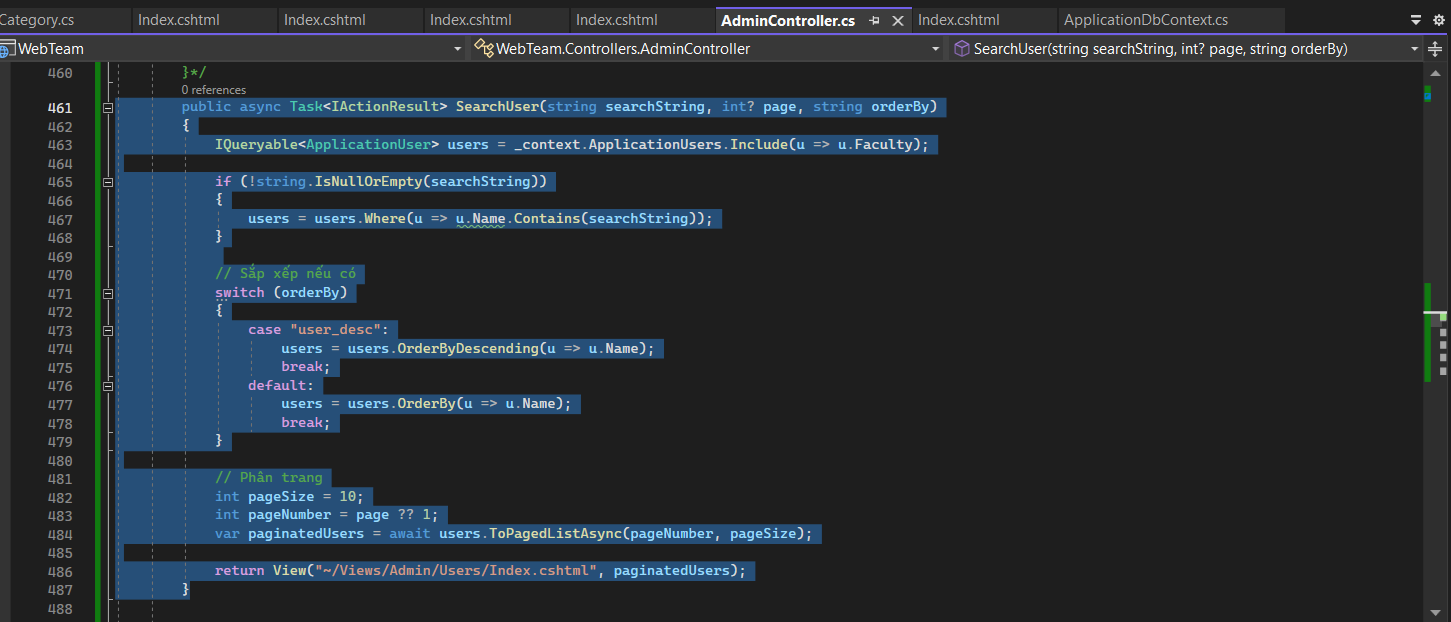
var query = AcademicYears.Where(c => c.AcademicYears.Contains(searchString));

return query;

}

## **Controller:**

**----User----**

****

**Code:**

public async Task<IActionResult> SearchUser(string searchString, int? page, string orderBy)

{

IQueryable<ApplicationUser> users = \_context.ApplicationUsers.Include(u => u.Faculty);

if (!string.IsNullOrEmpty(searchString))

{

users = users.Where(u => u.Name.Contains(searchString));

}

// Sắp xếp nếu có

switch (orderBy)

{

case "user\_desc":

users = users.OrderByDescending(u => u.Name);

break;

default:

users = users.OrderBy(u => u.Name);

break;

}

// Phân trang

int pageSize = 10;

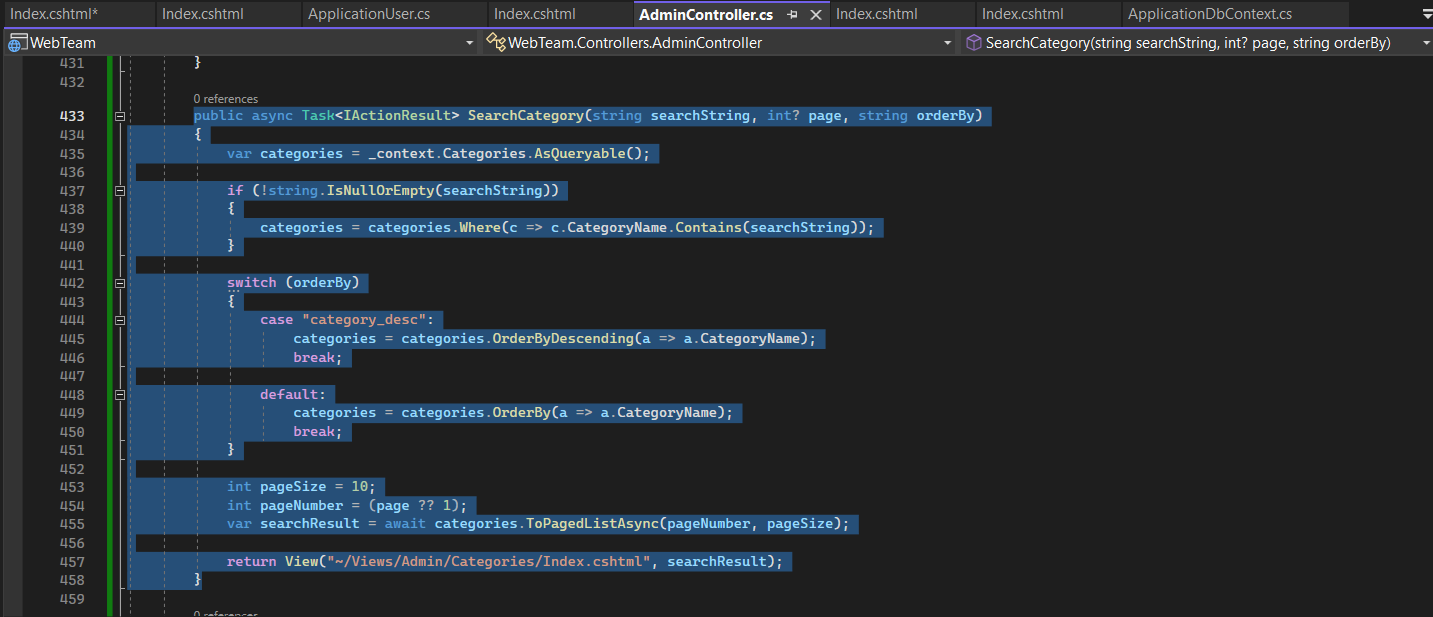
int pageNumber = page ?? 1;

var paginatedUsers = await users.ToPagedListAsync(pageNumber, pageSize);

return View("~/Views/Admin/Users/Index.cshtml", paginatedUsers);

}

**----Category----**



Code:

public async Task<IActionResult> SearchCategory(string searchString, int? page, string orderBy)

{

IQueryable<Category> categories = \_context.Categories.Include(c => c.Faculty);

if (!string.IsNullOrEmpty(searchString))

{

categories = categories.Where(c => c.CategoryName.Contains(searchString));

}

switch (orderBy)

{

case "category\_desc":

categories = categories.OrderByDescending(a => a.CategoryName);

break;

default:

categories = categories.OrderBy(a => a.CategoryName);

break;

}

int pageSize = 10;

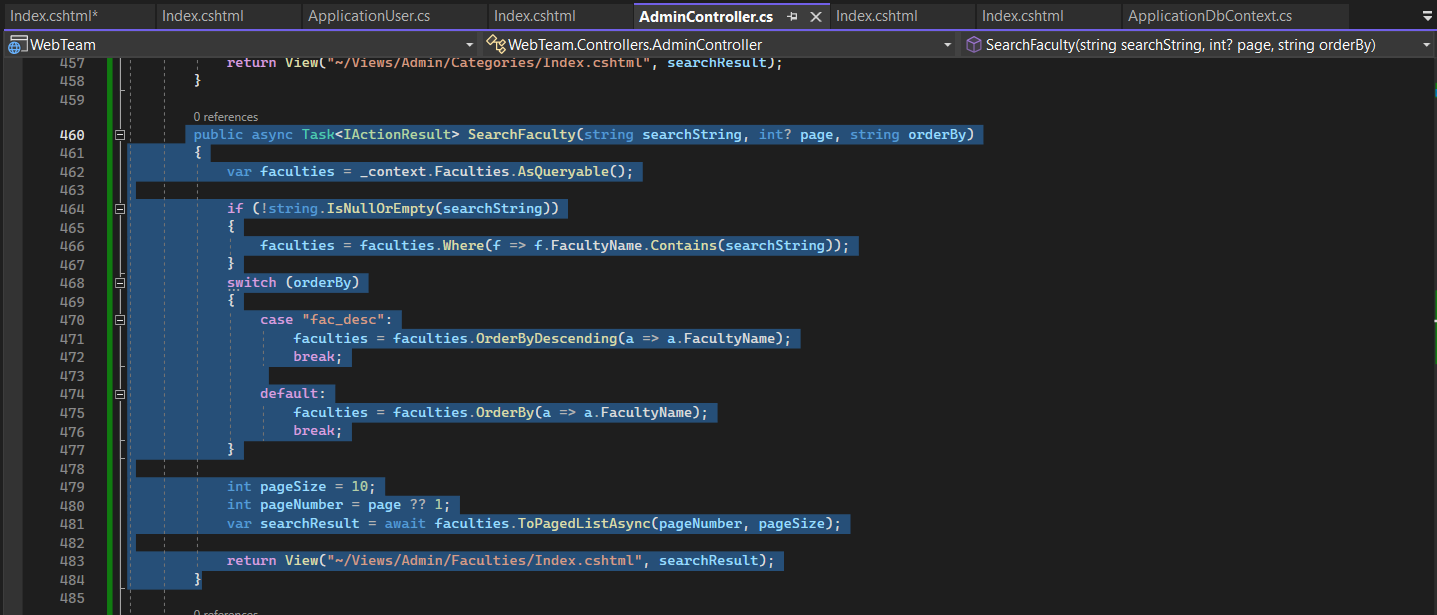
int pageNumber = (page ?? 1);

var searchResult = await categories.ToPagedListAsync(pageNumber, pageSize);

return View("~/Views/Admin/Categories/Index.cshtml", searchResult);

}

**----Faculty----**



Code:

public async Task<IActionResult> SearchFaculty(string searchString, int? page, string orderBy)

{

var faculties = \_context.Faculties.AsQueryable();

if (!string.IsNullOrEmpty(searchString))

{

faculties = faculties.Where(f => f.FacultyName.Contains(searchString));

}

switch (orderBy)

{

case "fac\_desc":

faculties = faculties.OrderByDescending(a => a.FacultyName);

break;

default:

faculties = faculties.OrderBy(a => a.FacultyName);

break;

}

int pageSize = 10;

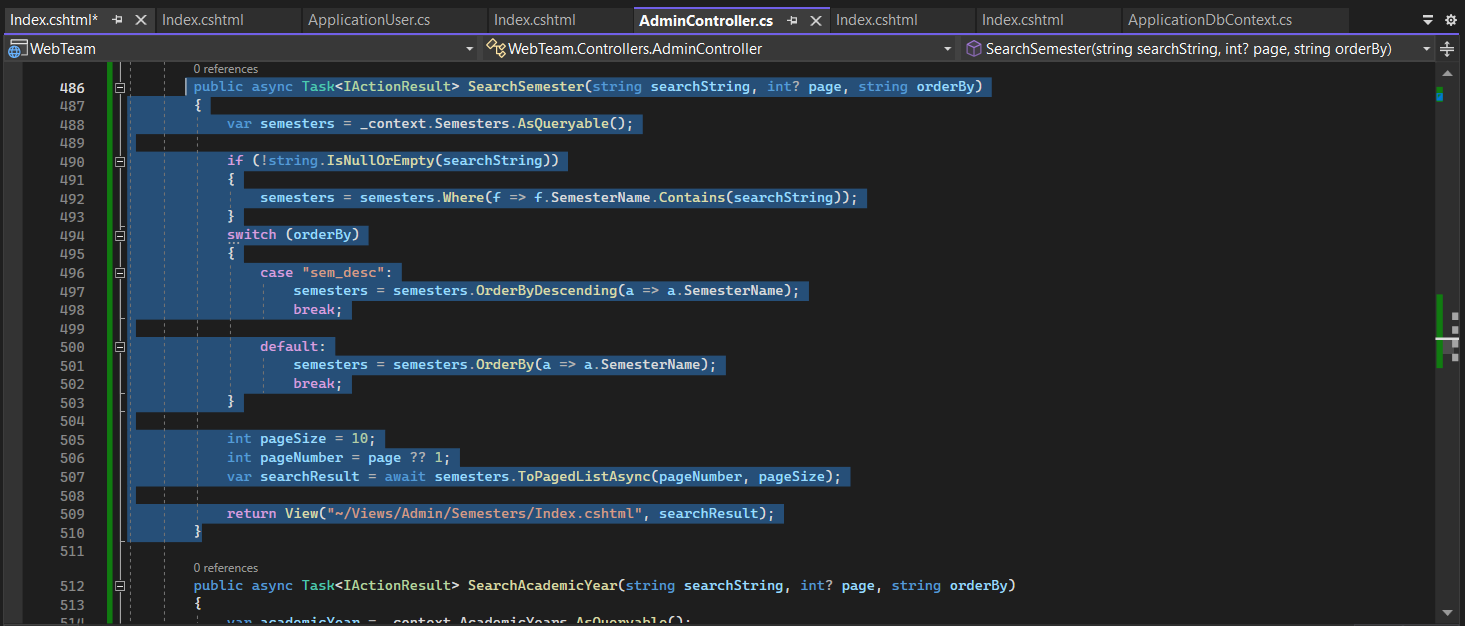
int pageNumber = page ?? 1;

var searchResult = await faculties.ToPagedListAsync(pageNumber, pageSize);

return View("~/Views/Admin/Faculties/Index.cshtml", searchResult);

}

**----Semester----**



Code:

public async Task<IActionResult> SearchSemester(string searchString, int? page, string orderBy)

{

var semesters = \_context.Semesters.AsQueryable();

if (!string.IsNullOrEmpty(searchString))

{

semesters = semesters.Where(f => f.SemesterName.Contains(searchString));

}

switch (orderBy)

{

case "sem\_desc":

semesters = semesters.OrderByDescending(a => a.SemesterName);

break;

default:

semesters = semesters.OrderBy(a => a.SemesterName);

break;

}

int pageSize = 10;

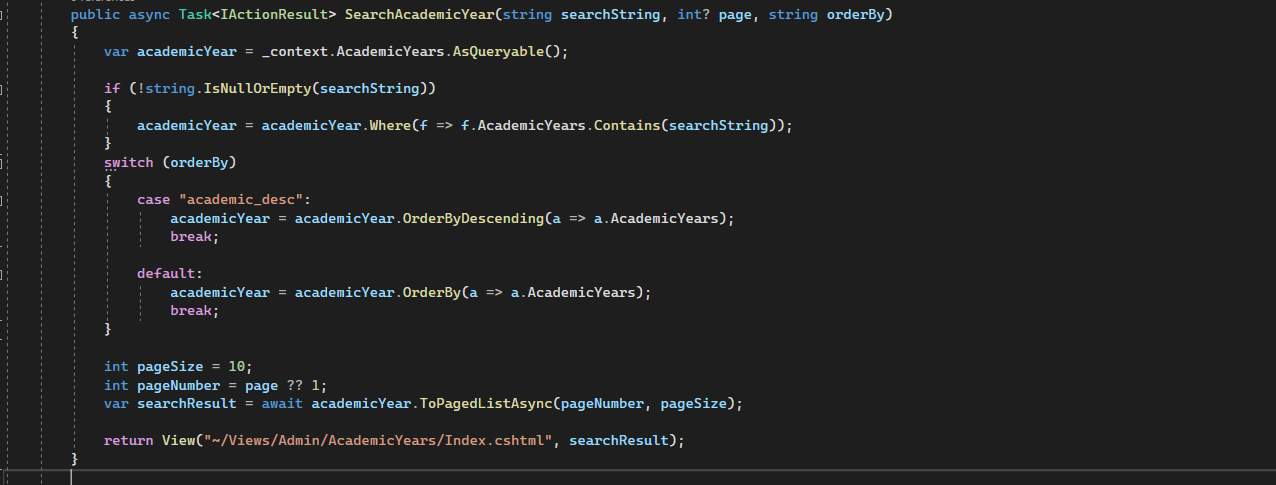
int pageNumber = page ?? 1;

var searchResult = await semesters.ToPagedListAsync(pageNumber, pageSize);

return View("~/Views/Admin/Semesters/Index.cshtml", searchResult);

}

**----AcademicYear----**



Code:

public async Task<IActionResult> SearchAcademicYear(string searchString, int? page, string orderBy)

{

var academicYear = \_context.AcademicYears.AsQueryable();

if (!string.IsNullOrEmpty(searchString))

{

academicYear = academicYear.Where(f => f.AcademicYears.Contains(searchString));

}

switch (orderBy)

{

case "academic\_desc":

academicYear = academicYear.OrderByDescending(a => a.AcademicYears);

break;

default:

academicYear = academicYear.OrderBy(a => a.AcademicYears);

break;

}

int pageSize = 10;

int pageNumber = page ?? 1;

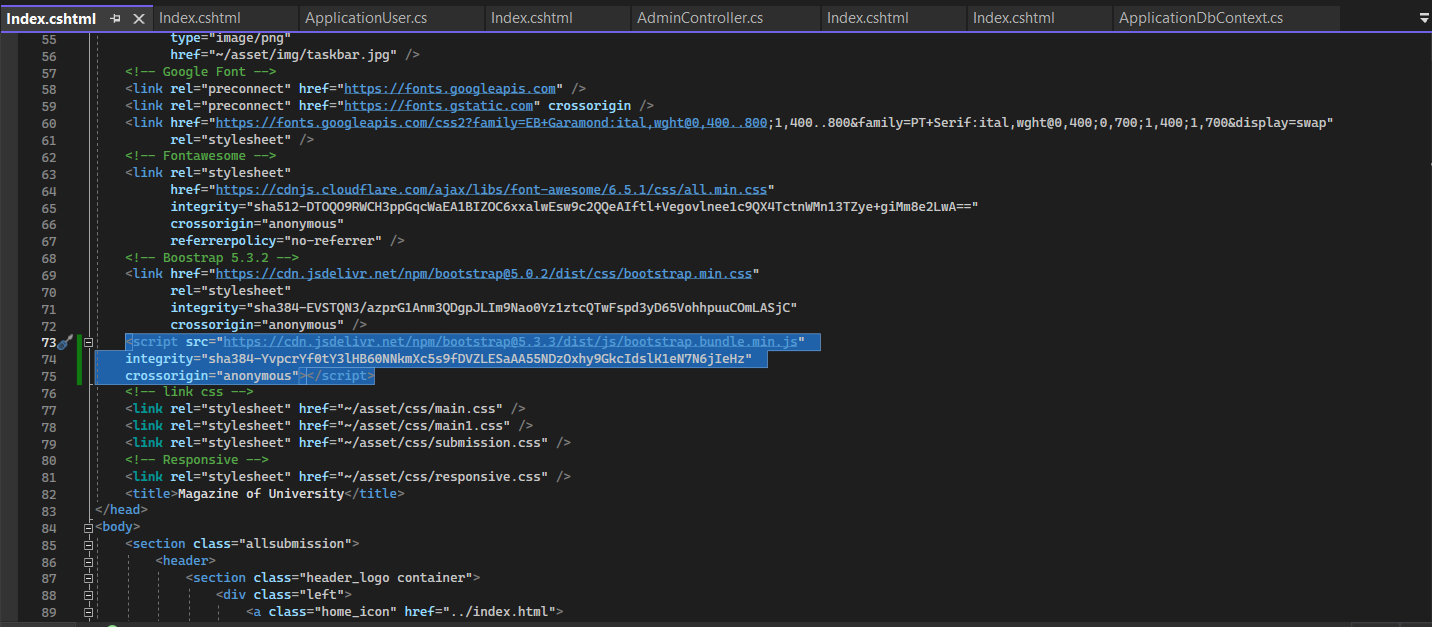
var searchResult = await academicYear.ToPagedListAsync(pageNumber, pageSize);

return View("~/Views/Admin/AcademicYears/Index.cshtml", searchResult);

}

## **Index:**

### **1. bỏ dòng này vào phía trước tất cả các Index**

****

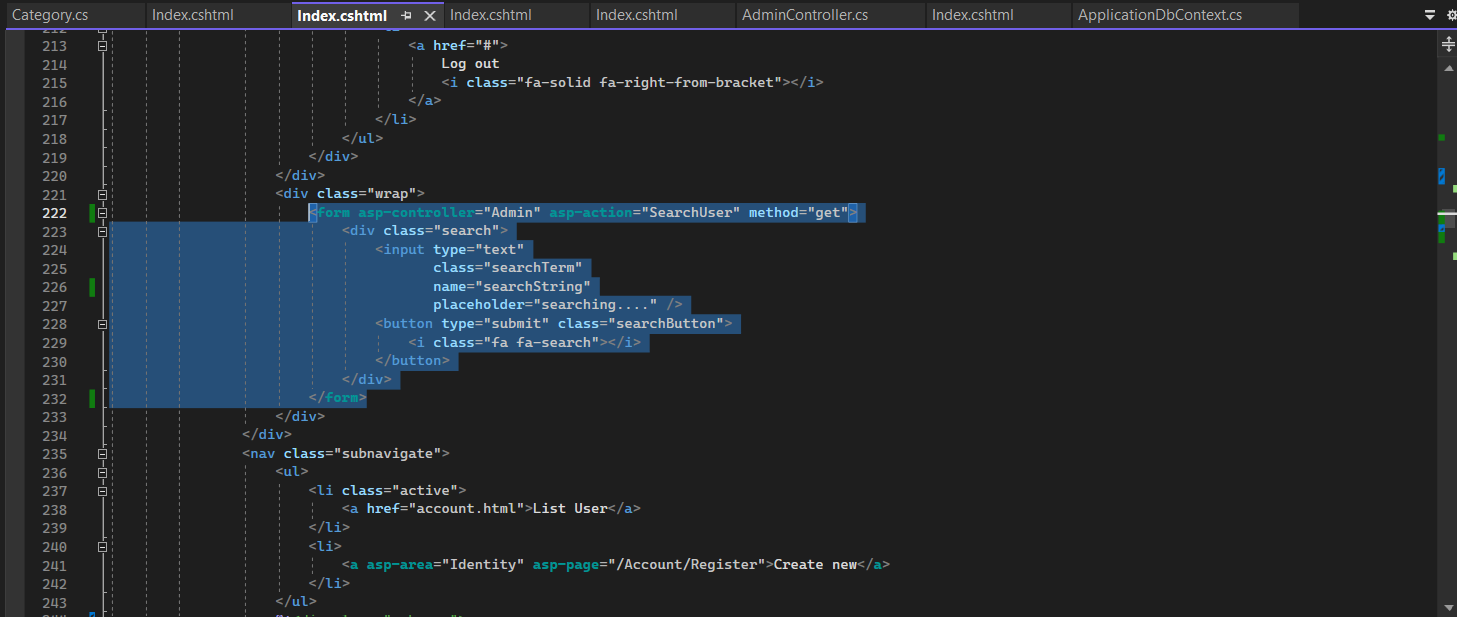
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"

crossorigin="anonymous"></script>

### **2. Bỏ tiếp cái này vào và nhớ chỉ cần chỉnh cái tên asp-action thôi để nó chạy. Tên thì có trong controller như trên đó**

**---User---**

****

**Code:**

<**form** **asp-controller**="Admin" **asp-action**="SearchUser" method="get">

<div class="search">

<input type="text"

class="searchTerm"

name="searchString"

placeholder="searching...." />

<button type="submit" class="searchButton">

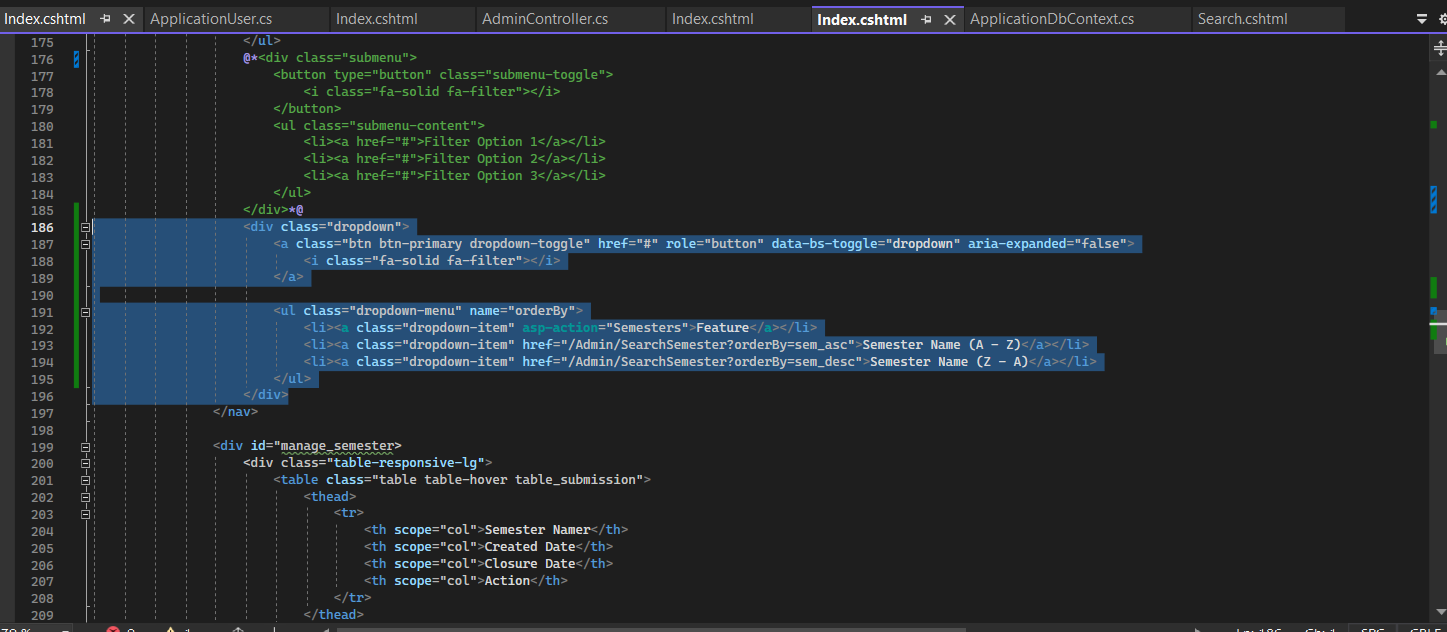
<i class="fa fa-search"></i>

</button>

</div>

</**form**>

**---Category---**

****

**Code:**

<**form** **asp-controller**="Admin" **asp-action**="SearchCategory" method="get">

<div class="search">

<input type="text"

class="searchTerm"

name="searchString"

placeholder="searching...." />

<button type="submit" class="searchButton">

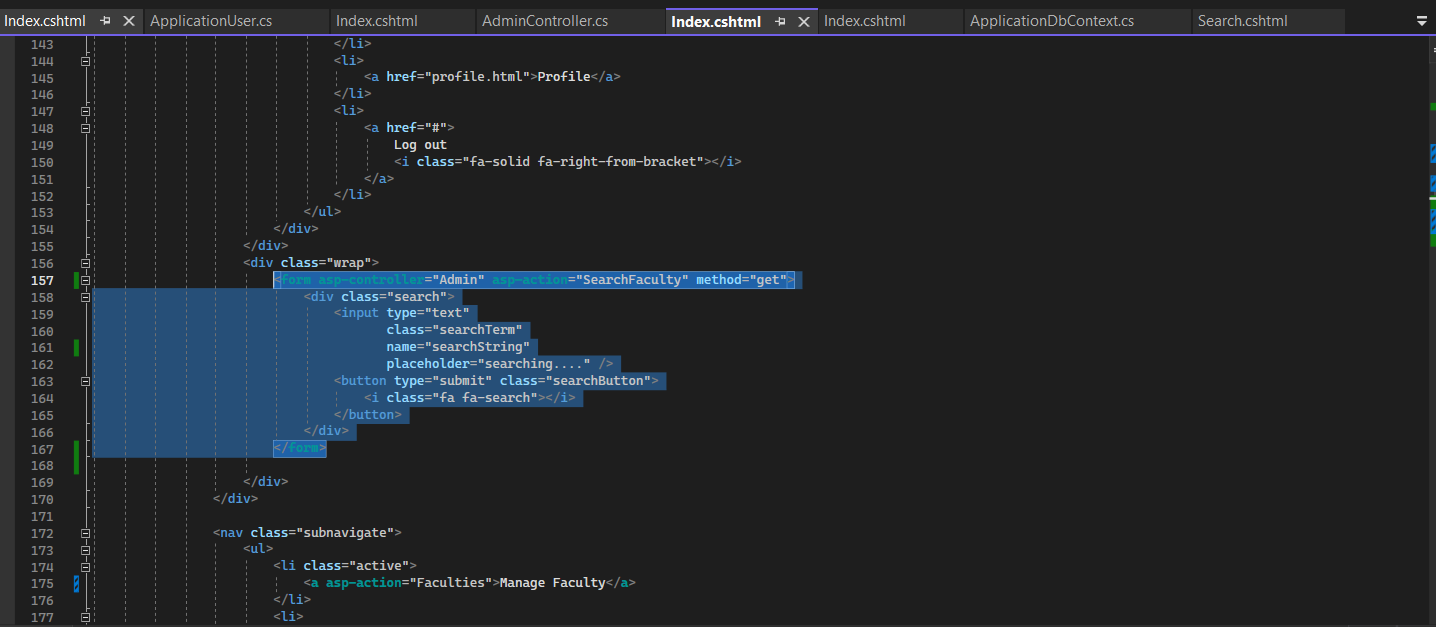
<i class="fa fa-search"></i>

</button>

</div>

</**form**>

**---Faculty---**

****

**Code:**

<**form** **asp-controller**="Admin" **asp-action**="SearchFaculty" method="get">

<div class="search">

<input type="text"

class="searchTerm"

name="searchString"

placeholder="searching...." />

<button type="submit" class="searchButton">

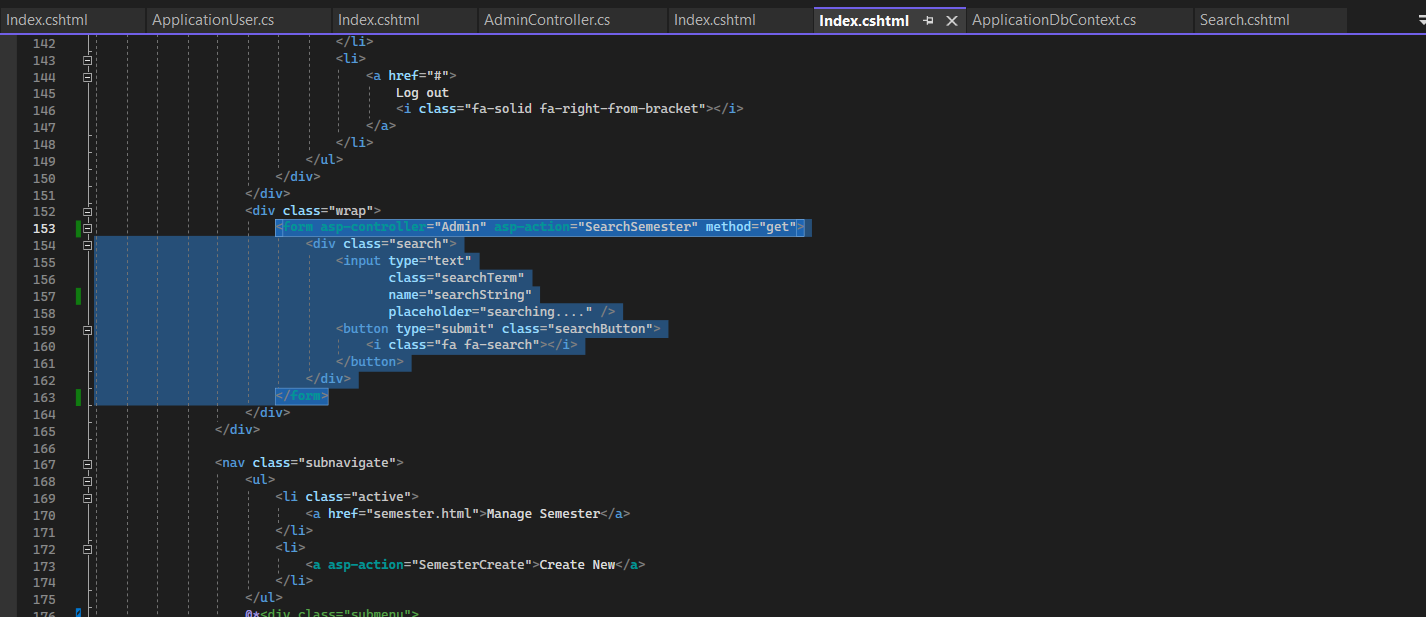
<i class="fa fa-search"></i>

</button>

</div>

</**form**>

**---Semester---**

****

**Code:**

<**form** **asp-controller**="Admin" **asp-action**="SearchSemester" method="get">

<div class="search">

<input type="text"

class="searchTerm"

name="searchString"

placeholder="searching...." />

<button type="submit" class="searchButton">

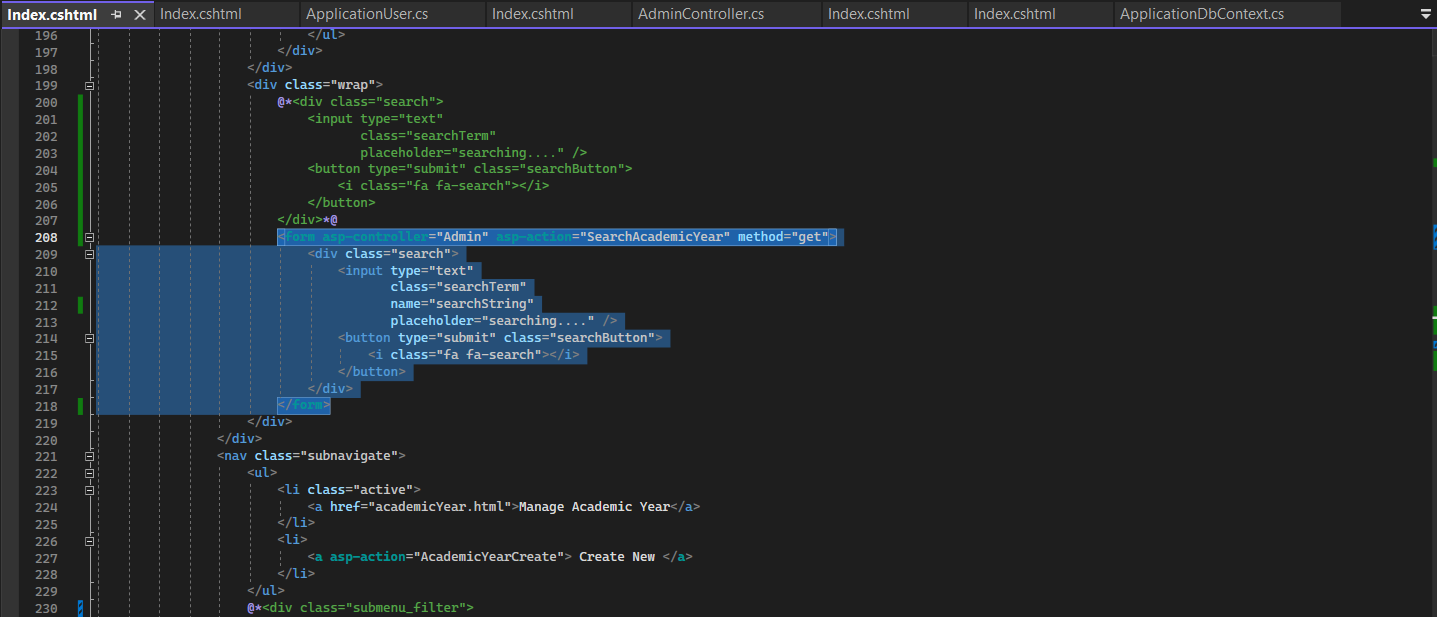
<i class="fa fa-search"></i>

</button>

</div>

</**form**>

**---Academicyear---**



**Code:**

<**form** **asp-controller**="Admin" **asp-action**="SearchAcademicYear" method="get">

<div class="search">

<input type="text"

class="searchTerm"

name="searchString"

placeholder="searching...." />

<button type="submit" class="searchButton">

<i class="fa fa-search"></i>

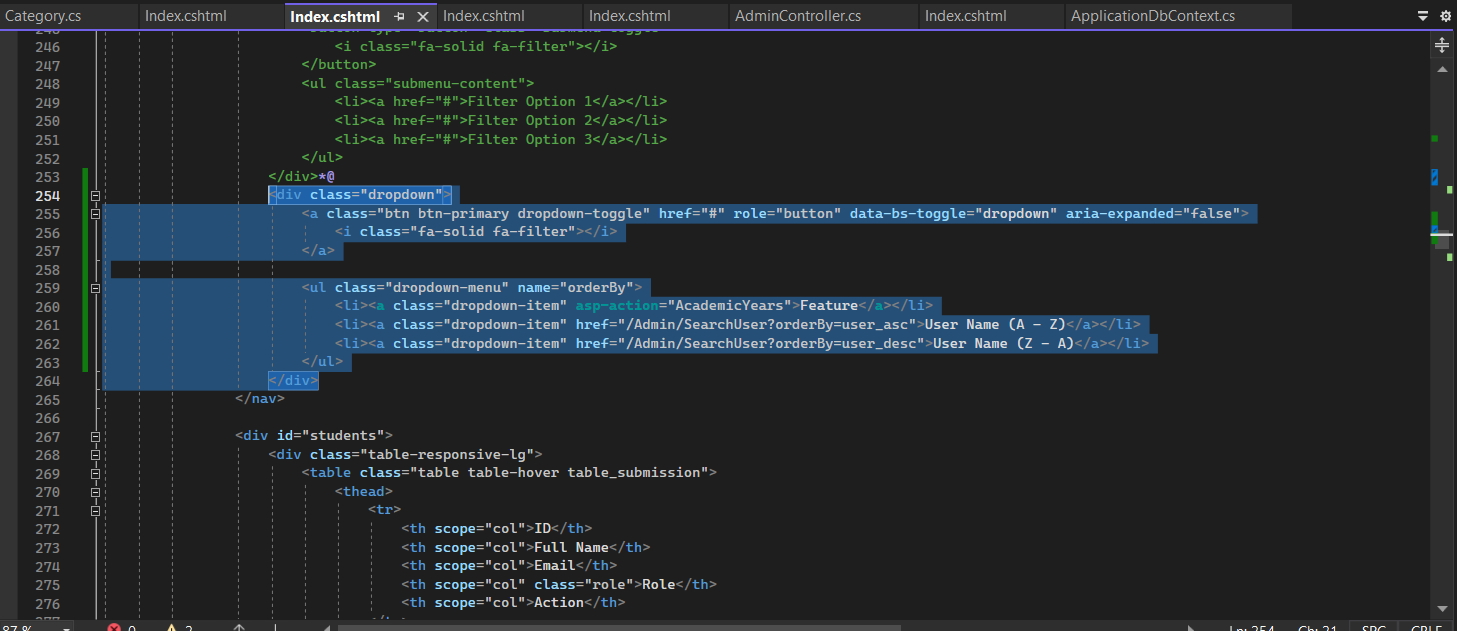
</button>

</div>

</**form**>

### **3. Bỏ cái này để sử dụng sort. Nút sort có sẵn bị hư và không bấm được**

**---User---**

****

**Code:**

<div class="dropdown">

<a class="btn btn-primary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

<i class="fa-solid fa-filter"></i>

</a>

<ul class="dropdown-menu" name="orderBy">

<li><**a** class="dropdown-item" **asp-action**="AcademicYears">Feature</**a**></li>

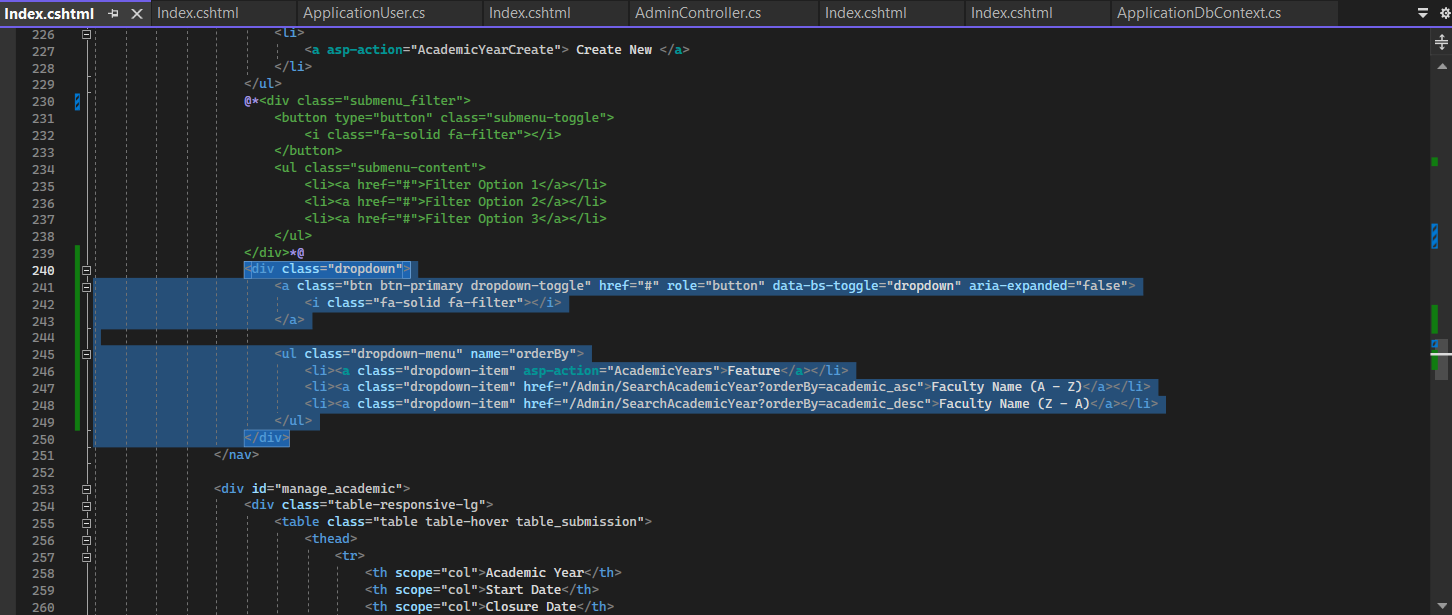
<li><a class="dropdown-item" href="/Admin/SearchUser?orderBy=user\_asc">User Name (A - Z)</a></li>

<li><a class="dropdown-item" href="/Admin/SearchUser?orderBy=user\_desc">User Name (Z - A)</a></li>

</ul>

</div>

**---AcademicYear---**

****

**Code:**

<div class="dropdown">

<a class="btn btn-primary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

<i class="fa-solid fa-filter"></i>

</a>

<ul class="dropdown-menu" name="orderBy">

<li><**a** class="dropdown-item" **asp-action**="AcademicYears">Feature</**a**></li>

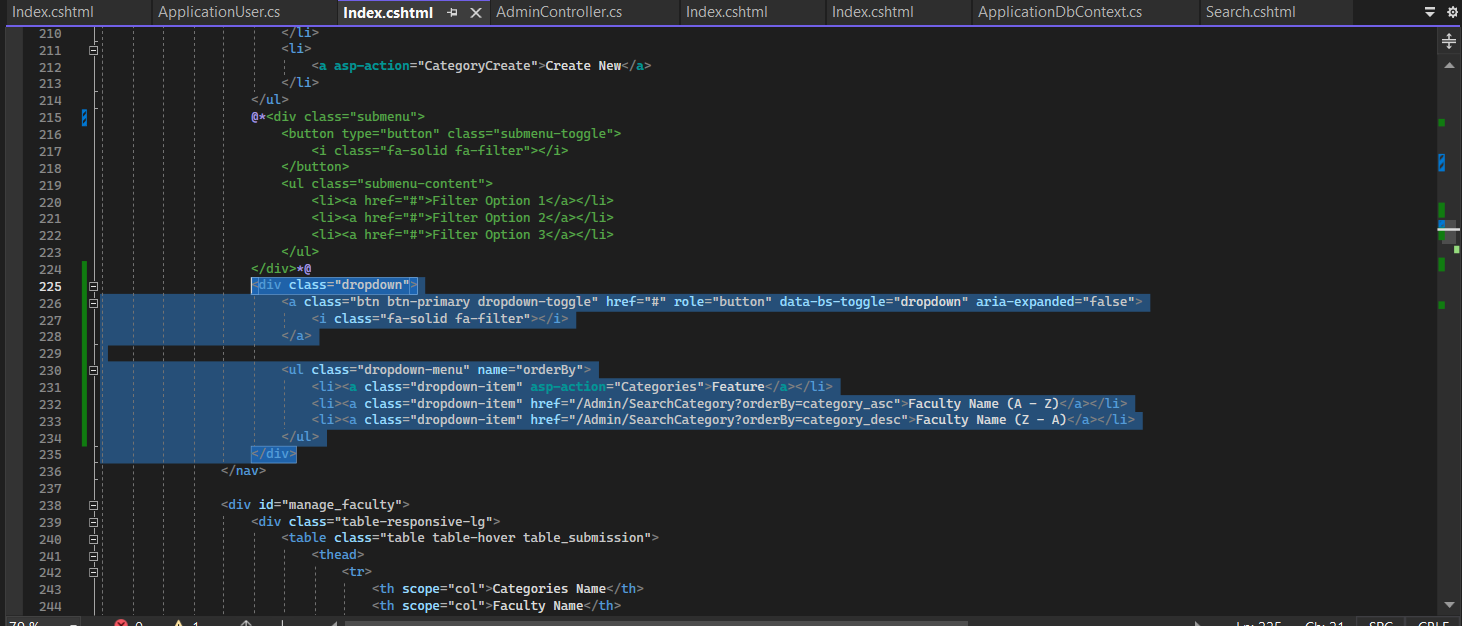
<li><a class="dropdown-item" href="/Admin/SearchAcademicYear?orderBy=academic\_asc">Faculty Name (A - Z)</a></li>

<li><a class="dropdown-item" href="/Admin/SearchAcademicYear?orderBy=academic\_desc">Faculty Name (Z - A)</a></li>

</ul>

</div>

**---Category---**

****

**Code:**

<div class="dropdown">

<a class="btn btn-primary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

<i class="fa-solid fa-filter"></i>

</a>

<ul class="dropdown-menu" name="orderBy">

<li><**a** class="dropdown-item" **asp-action**="Categories">Feature</**a**></li>

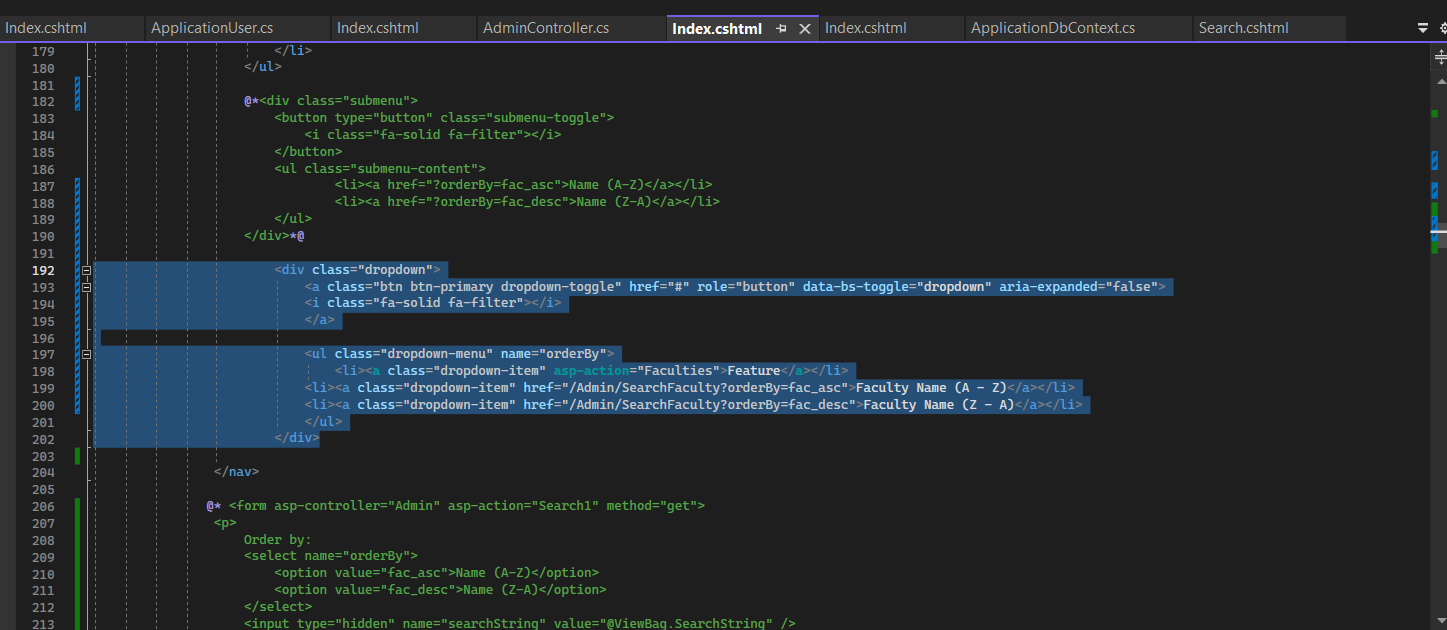
<li><a class="dropdown-item" href="/Admin/SearchCategory?orderBy=category\_asc">Faculty Name (A - Z)</a></li>

<li><a class="dropdown-item" href="/Admin/SearchCategory?orderBy=category\_desc">Faculty Name (Z - A)</a></li>

</ul>

</div>

**---Faculty---**

****

**Code:**

<div class="dropdown">

<a class="btn btn-primary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

<i class="fa-solid fa-filter"></i>

</a>

<ul class="dropdown-menu" name="orderBy">

<li><**a** class="dropdown-item" **asp-action**="Faculties">Feature</**a**></li>

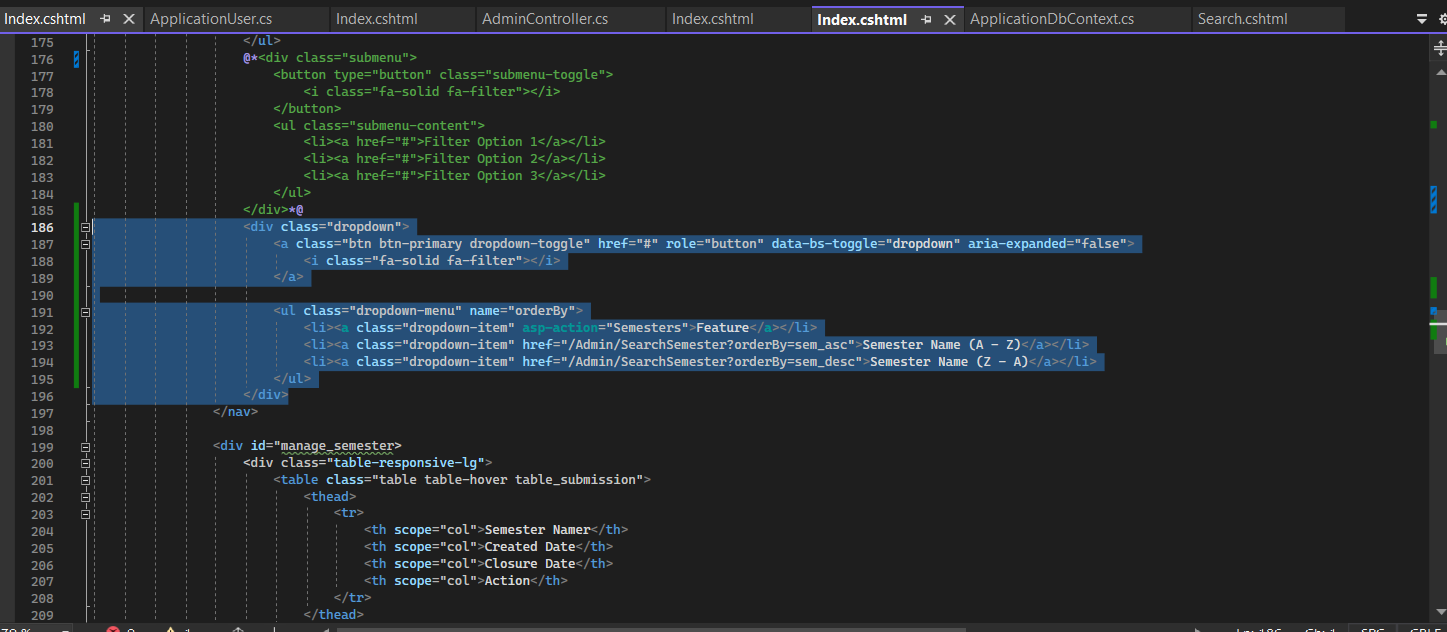
<li><a class="dropdown-item" href="/Admin/SearchFaculty?orderBy=fac\_asc">Faculty Name (A - Z)</a></li>

<li><a class="dropdown-item" href="/Admin/SearchFaculty?orderBy=fac\_desc">Faculty Name (Z - A)</a></li>

</ul>

</div>

**---Semester---**

****

**Code:**

<div class="dropdown">

<a class="btn btn-primary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">

<i class="fa-solid fa-filter"></i>

</a>

<ul class="dropdown-menu" name="orderBy">

<li><**a** class="dropdown-item" **asp-action**="Semesters">Feature</**a**></li>

<li><a class="dropdown-item" href="/Admin/SearchSemester?orderBy=sem\_asc">Semester Name (A - Z)</a></li>

<li><a class="dropdown-item" href="/Admin/SearchSemester?orderBy=sem\_desc">Semester Name (Z - A)</a></li>

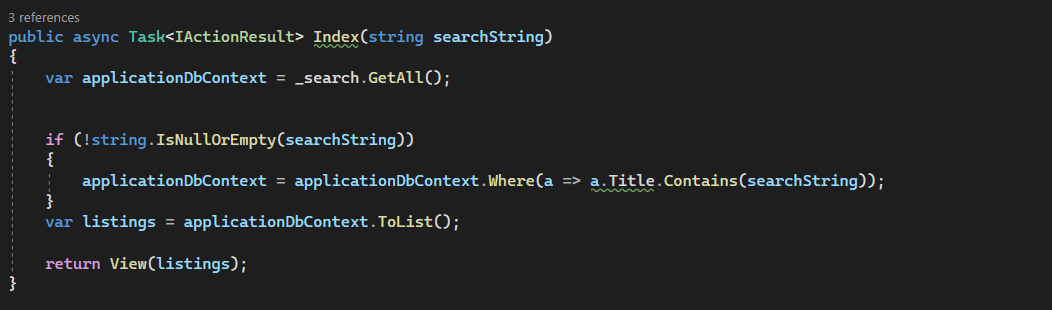
</ul>

</div>

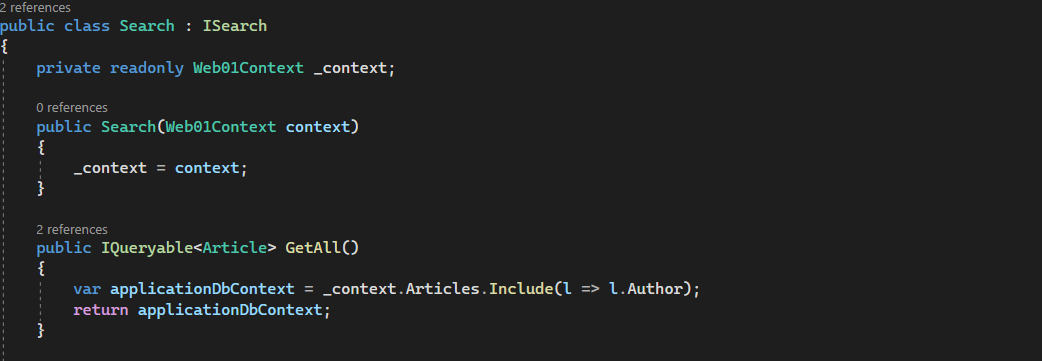
----------------------------------------------------------------------------------------------------------------------

# **SEARCH**

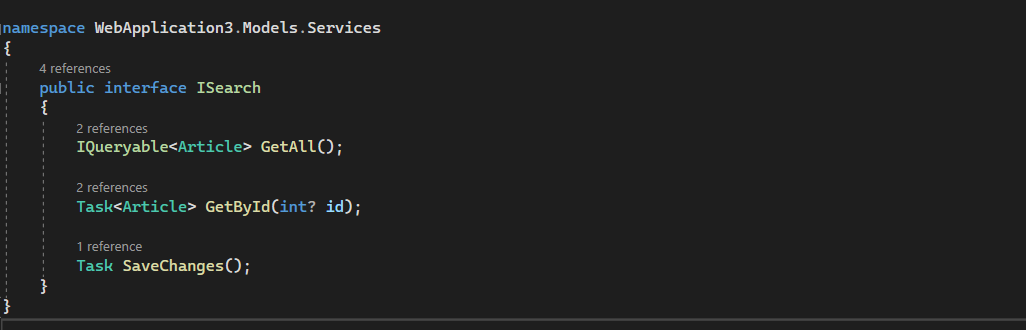
Explain code:



To conduct the search function, we will add a searchString argument to the Index class Controller, which will contain the data type string. The first index invokes \_search.The Search class's GetAll() method returns all posts. If searchString is supplied, the list is limited to articles with search phrases in their titles. Finally, Index delivers the list of articles (filtered or complete) to the "view" for presentation to the user.

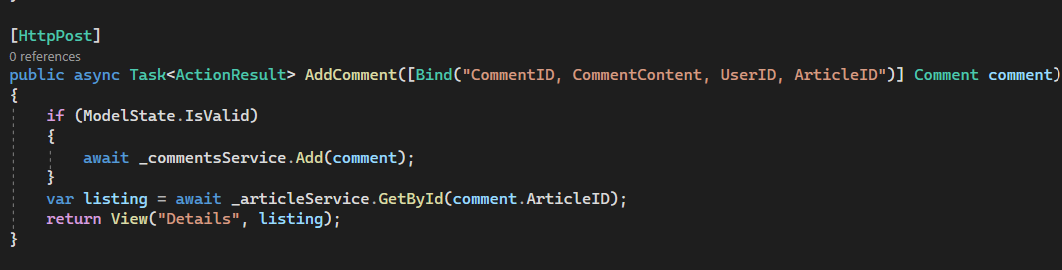


We will construct an Interface class called ISearch to define the operations that article search management classes may execute. In particular, GetAll is used to retrieve all postings.



We will construct an extra Search class to meet search demands that will be compatible with ISearch. The Search class will include a method named GetAll(). GetAll utilizes \_context to access all posts in the database and can extract additional "Author" information.

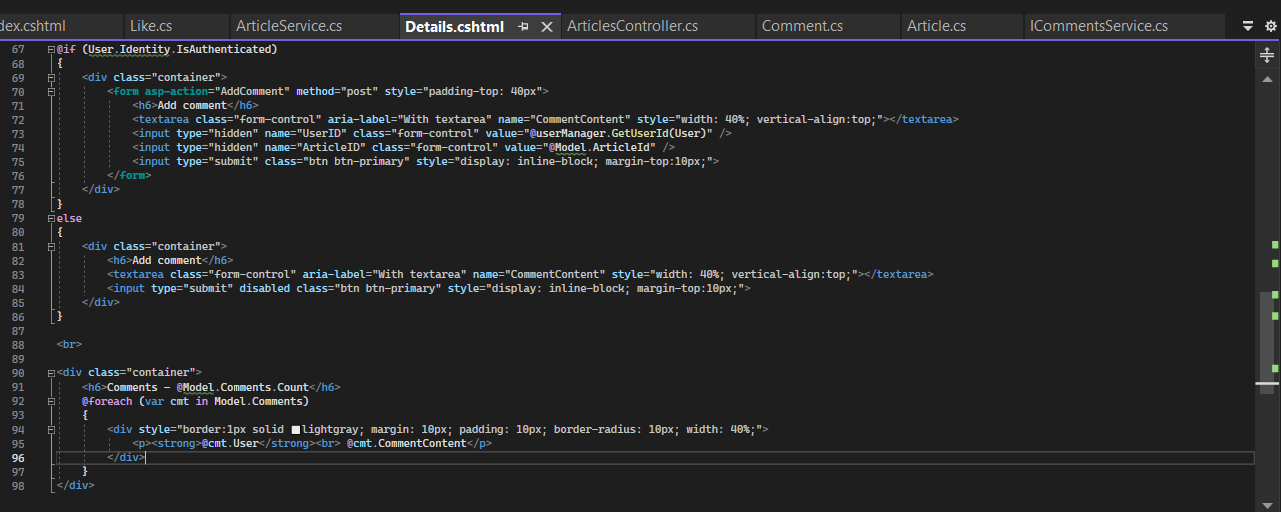
# --------------------------**Comment--**-----------------------------



Explanation:

* This method is used to handle the addition of a comment to an article.
* It's an HTTP POST method, indicating it's invoked when a form is submitted.
* The comment to be added is passed as a parameter.
* It first checks if the ModelState is valid, ensuring that the comment data is properly formatted.
* If the ModelState is valid, it awaits the \_commentsService.Add(comment) call to add the comment to the database asynchronously.
* Then it retrieves the updated article (with the newly added comment) asynchronously using \_articleService.GetById(comment.ArticleID).
* Finally, it returns a view (presumably showing the details of the article) passing the retrieved article as a parameter to the view.





1. Comment Form for Authenticated Users:

* If the user is authenticated, a form is displayed where the user can enter a comment.
* The form is submitted to the AddComment action using the asp-action attribute.
* It includes a textarea for the comment content, hidden fields for the user ID (UserID) and the article ID (ArticleID), and a submit button.

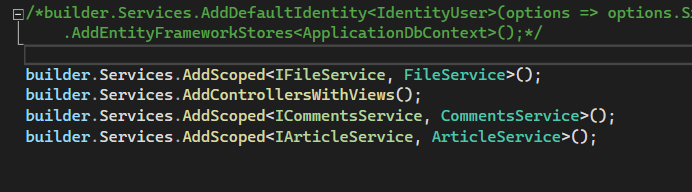
2. Comment Form for Unauthenticated Users:

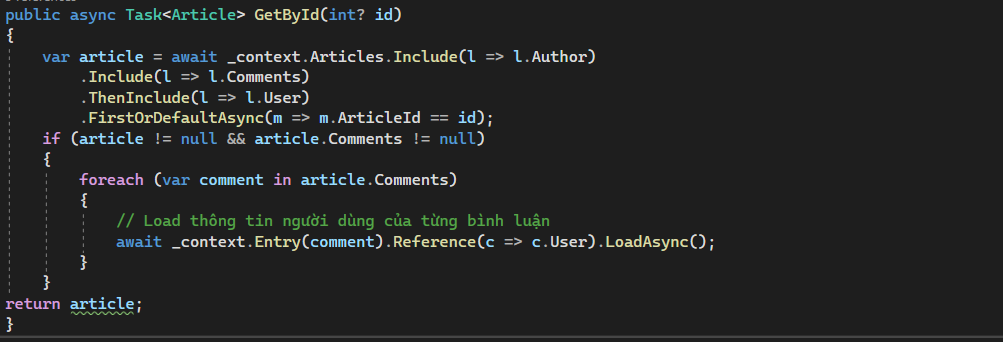
* If the user is not authenticated, a disabled submit button is displayed, and the user is not allowed to submit a comment.

3. Display Existing Comments:

* After the comment form or the message for unauthenticated users, existing comments are displayed.
* It iterates through each comment in the Model.Comments collection.
* Each comment is displayed in a bordered box with the user's name and comment content.

4. Conditional UI Rendering: The first part of the code checks if the user is authenticated (User.Identity.IsAuthenticated). If the user is authenticated, it renders a form to add a comment; otherwise, it displays a message indicating that the user needs to be authenticated to add a comment.

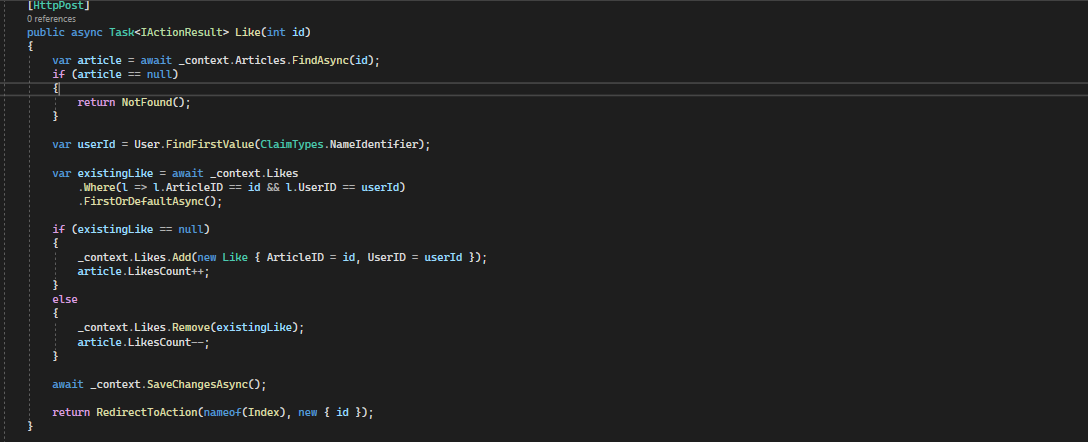




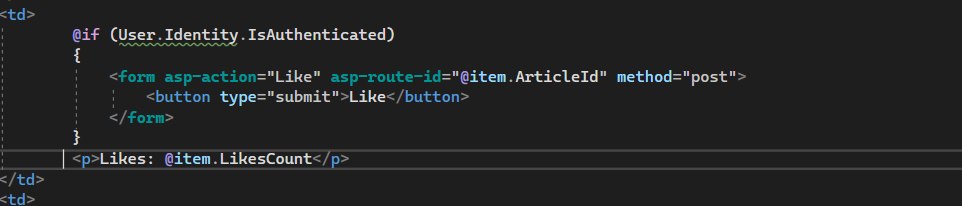
Explanation:

* This method is used to retrieve an article by its ID asynchronously.
* It utilizes Entity Framework Core's asynchronous methods for database interaction.
* It first fetches an article from the database based on the provided ID.
* .Include() is used to eagerly load related entities (Author, Comments) to avoid lazy loading issues.
* .ThenInclude() is used to further include the User related to each Comment.
* The method then checks if the article and its comments exist and are not null.
* If comments exist, it iterates through each comment and loads its associated user asynchronously.
* Finally, it returns the fetched article.

# **--------------------------Like** -------------------------------

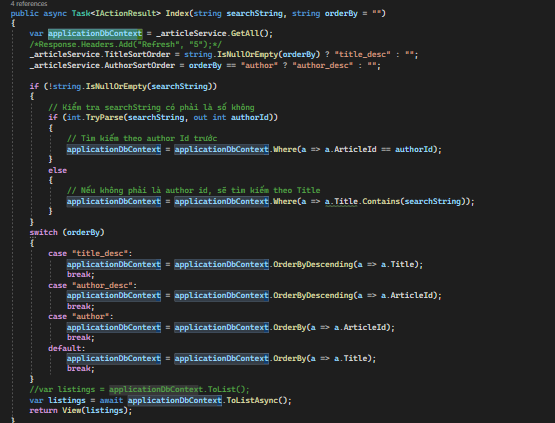


* HTTP POST Method: This method is decorated with the [HttpPost] attribute, indicating that it handles HTTP POST requests.
* Finding the Article: It finds the article with the provided id using Entity Framework Core's FindAsync method. If the article does not exist, it returns a 404 NotFound response.
* User Identification: It retrieves the user ID of the current user from the claims.
* Checking Existing Like: It checks if the user has already liked the article by querying the Likes table based on the article ID and user ID.
* Handling Like Action:
  + If the user has not liked the article yet, it adds a new entry to the Likes table indicating that the user has liked the article and increments the LikesCount property of the article.
  + If the user has already liked the article, it removes their existing like from the Likes table and decrements the LikesCount property of the article.
* Saving Changes: It saves the changes to the database using SaveChangesAsync method.
* Redirecting: It redirects the user to the Index action, passing the article ID as a parameter. This action presumably displays the details of the article, showing the updated like count.



* This code is inside a table cell (<td>).
* It first checks if the user is authenticated using User.Identity.IsAuthenticated.
* If the user is authenticated, it renders a form to submit a "Like" action.
  + The form submits to the "Like" action with the article ID as a route parameter.
  + The button triggers a form submission when clicked.
* After the form (if rendered), it displays the number of likes for the item (@item.LikesCount) outside of the form.
* So, if the user is authenticated, they can click the "Like" button to submit a like for the item, and the number of likes is displayed. If the user is not authenticated, only the number of likes is displayed without the like button.

# **--------------------------Sort** -------------------------------



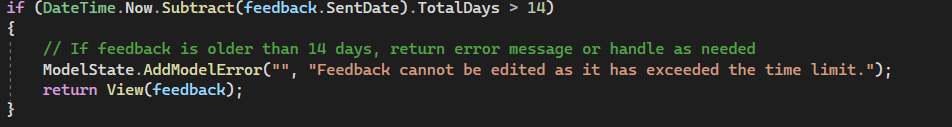
1. **Index(string orderBy = "")**: This is the **Index** action method, which accepts a parameter named **orderBy**, with a default value of an empty string. This parameter is used to determine how the data should be sorted before displaying it on the **Index** page.
2. **Fetching data from the \_articleService**: Data is fetched from a service or service class called **\_articleService**. This service has a method **GetAll()** that returns a list of articles.
3. **Determining data sorting**: Based on the value of **orderBy**, this action determines how the data should be sorted before returning it to the view. There are the following cases:
   * If **orderBy** is **"title\_desc"**, the data will be sorted in descending order by the article title (**Title**).
   * If **orderBy** is **"author\_desc"**, the data will be sorted in descending order by **ArticleId**.
   * If **orderBy** is **"author"**, the data will be sorted in ascending order by **ArticleId**.
   * By default, the data will be sorted in ascending order by the article title.
4. **Calling ToListAsync() method**: The sorted data is converted to a list and assigned to the variable **listings**.
5. **Returning view with data**: After fetching and sorting the data, the action returns the **Index** view with the sorted list of articles.
6. **Returning IActionResult**: This action returns an **IActionResult**, allowing it to return a **View**, a **PartialView**, or even a **JsonResult** or **ContentResult**. In this case, the action returns a **View**.

# --------------------------**Set up limit 14days to Feedback and lock -------------------------------**

This line queries the database to retrieve the feedback record with the specified **id**. **AsNoTracking()** is used to indicate that the retrieved entities are not going to be tracked by the DbContext, which can be beneficial for read-only operations like this one.



This section updates some properties of the **feedback** object (presumably the updated feedback) with values from the **oldFeedback** retrieved from the database. It appears to be retaining the original values for **Send** and **SentDate** from the old feedback.



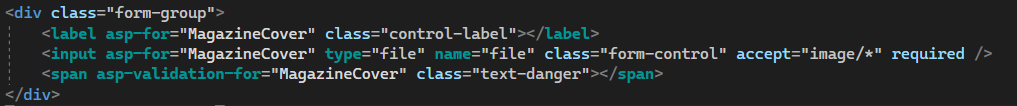
This condition checks if the difference between the current date and the **SentDate** of the updated feedback is greater than 14 days. If it is, it indicates that the feedback is older than 14 days, and an error message is added to the ModelState. This typically means that the feedback cannot be edited due to exceeding a time limit.

# **--------------------------Upload Image ( + Limit 5MB) -------------------------------**



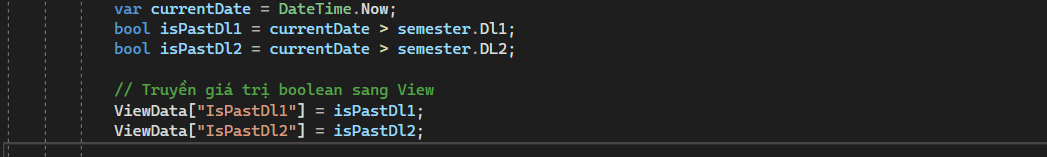
1. **IFormFile**: **IFormFile** is an interface provided by ASP.NET Core for file uploads. It represents a file sent with the HttpRequest. **( I have to add IForm File first)**
2. **if (file != null && file.Length > 0)**: This condition checks whether a file has been uploaded and if its length is greater than zero, ensuring that the file contains data. If the condition is true, it proceeds to handle the uploaded file.
3. **long maxFileSize = 5 \* 1024 \* 1024; // 5MB**: This line sets the maximum allowed file size to 5 MB. This value can be adjusted according to your application's requirements.
4. **if (file.Length > maxFileSize)**: This condition checks if the uploaded file exceeds the maximum allowed file size. If it does, it returns a BadRequest response with an appropriate error message.
5. **var fileName = Path.GetFileName(file.FileName);**: This line extracts the file name from the **FileName** property of the uploaded file.
6. **var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/Uploads", fileName);**: This line constructs the path where the uploaded file will be saved. Typically, it's within the **wwwroot** directory to ensure accessibility from the client side.
7. **using (var fileStream = new FileStream(filePath, FileMode.Create))**: This line creates a **FileStream** to write the uploaded file to the specified file path.
8. **await file.CopyToAsync(fileStream);**: This line asynchronously copies the contents of the uploaded file to the **FileStream**, effectively saving the file to disk.
9. **article.MagazineCover = fileName;**: Assuming **article** is an instance of a class that contains information about the uploaded file (in this case, the file name), this line assigns the file name to a property (**MagazineCover**), presumably to be stored in a database or used elsewhere in the application.

HTML:

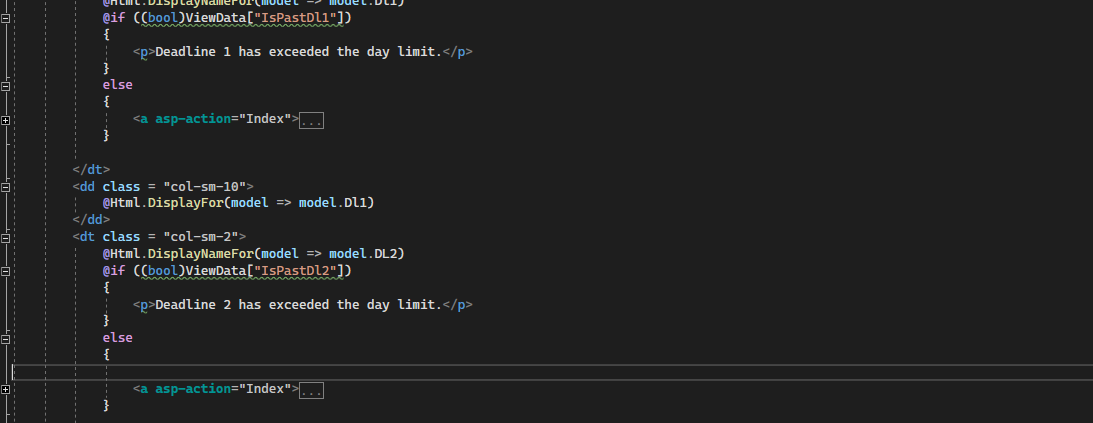


* **<div class="form-group">**: This is a container element with the class "form-group". It's commonly used in Bootstrap for styling form controls.
* **<label asp-for="MagazineCover" class="control-label"></label>**: This is a label element associated with the "MagazineCover" property of the model. The **asp-for** attribute generates the appropriate label text based on the model's metadata.
* **<input asp-for="MagazineCover" type="file" name="file" class="form-control" accept="image/\*" required />**: This is an input element used for file uploads. It's associated with the "MagazineCover" property of the model. The **type="file"** attribute specifies that this input should allow users to select a file. The **name="file"** attribute specifies the name of the input field. The **class="form-control"** attribute applies Bootstrap's "form-control" class for styling. The **accept="image/\*"** attribute specifies that only image files can be selected. The **required** attribute makes this input field required, meaning that the user must select a file before submitting the form.
* **<span asp-validation-for="MagazineCover" class="text-danger"></span>**: This is a span element used for displaying validation error messages related to the "MagazineCover" property. The **asp-validation-for** attribute specifies the property to which the validation message should be associated. The **class="text-danger"** attribute applies Bootstrap's "text-danger" class for styling the error message in red.

# **--------------------------Limit day for dl1 and dl2**-------------------------------



We calculate whether the current time has passed ‘**Dl1’** by comparing it with the value of the ‘**Dl1’** property of that semester. The result of this comparison is stored in the ‘**isPastDl1’** variable. Similarly, we calculate whether the current time has passed ‘**Dl2’** and store the result in the ‘**isPastDl2’** variable. Next, we pass the boolean values ‘**isPastDl1’** and ‘**isPastDl2’** through ViewData to use in the View.

In the **Details.cshtml** View, we use the ‘**IsPastDl1** and **IsPastDl2’** boolean values to determine whether to display the "Go back to Index page" button. If either of the boolean variables is **true**, we display a replacement message; otherwise, we display the button.

-------------------------------------------------------

**Không Liên quan**

**Dòng mã này yêu cầu ứng dụng ASP.NET Core đăng ký lớp Search làm dịch vụ triển khai giao diện ISearch. Thời lượng tồn tại của dịch vụ này được đặt thành "scoped", nghĩa là một phiên bản mới sẽ được tạo cho mỗi yêu cầu, đảm bảo rằng mỗi người dùng nhận được một phiên bản mới cho nhu cầu cụ thể của họ. Điều này thúc đẩy quản lý tài nguyên tốt hơn và tránh các vấn đề tiềm ẩn với trạng thái được chia sẻ giữa các yêu cầu khác nhau.**

1. **IQueryable nằm trong namespace System.Linq**
2. **IQueryable cũng chỉ có thể di chuyển 1 chiều tiến lên trong collection, nó không thể move back lại.**
3. **IQueryable tốt nhất cho truy vấn dữ liệu out-memory như là database.**
4. **Khi truy vấn, IQueryable thực thi câu lệnh truy vấn và lọc dữ liệu trên Server luôn**
5. **IQueryable phù hợp cho Linq to SQL**
6. **IQueryable hỗ trợ custom query sử dụng phương thức CreateQuery và Execute.**
7. **IQueryable hỗ trợ lazy loading. Vì thế nó phù hợp cho trường hợp phân trang.Ví dụ về IQueryable**