Axxes Haxx Conference System

Project Assignment

Design and implement a Web based Conference system for Haxx.

- Sessions have a title, date, description, category and speaker field. Sessions can be public (visible by everyone) or private (visible to their owner author only). Sessions may have comments. Comments belong to a certain session and have content (text), date and optional author (owner).

- Anonymous users (without login) can view all public sessions. The home page displays all public sessions, in two groups: sessions that are already past and upcoming sessions. Sessions are shown in short form (title, date, category and speaker) and have a [View Details] button, which loads dynamically (by AJAX) their description, comments and [Edit] / [Delete] buttons.

- Anonymous users can register in the system and login / logout. Users should have mandatory email, password and full name. User's email should be unique. User's password should be non-empty but can be just one character.

- Logged-in users should be able to view their own sessions, to create new sessions, edit their own sessions and delete their own sessions. Deleting sessions requires a confirmation. Implement client-side and sever-side data validation.

- A special user "admin@admin.com" should have the role "Administrator" and should have full permissions to edit / delete sessions and comments. This user can also see ALL sessions.

Step 1: List Session Details with AJAX

The next step in the development of the Conference management application is to **display sessions details dynamically with AJAX**. This can be done in a few steps.

1. First, define the **SessionDetailsViewModel** class that will hold the session details.

2. Second, define the CommentViewModel class that will hold the comment data.

3. Next, write the **controller action**, that will return the session details when users click the **[View Details]** button. The logic is not quite straightforward, because session details should be shown only when the user has **permissions to access** them:
   - The session speaker can view and edit its own sessions (even private).
   - System administrators can view and edit all sessions (even private).
   - Public sessions are visible by everyone, but not editable by everyone.

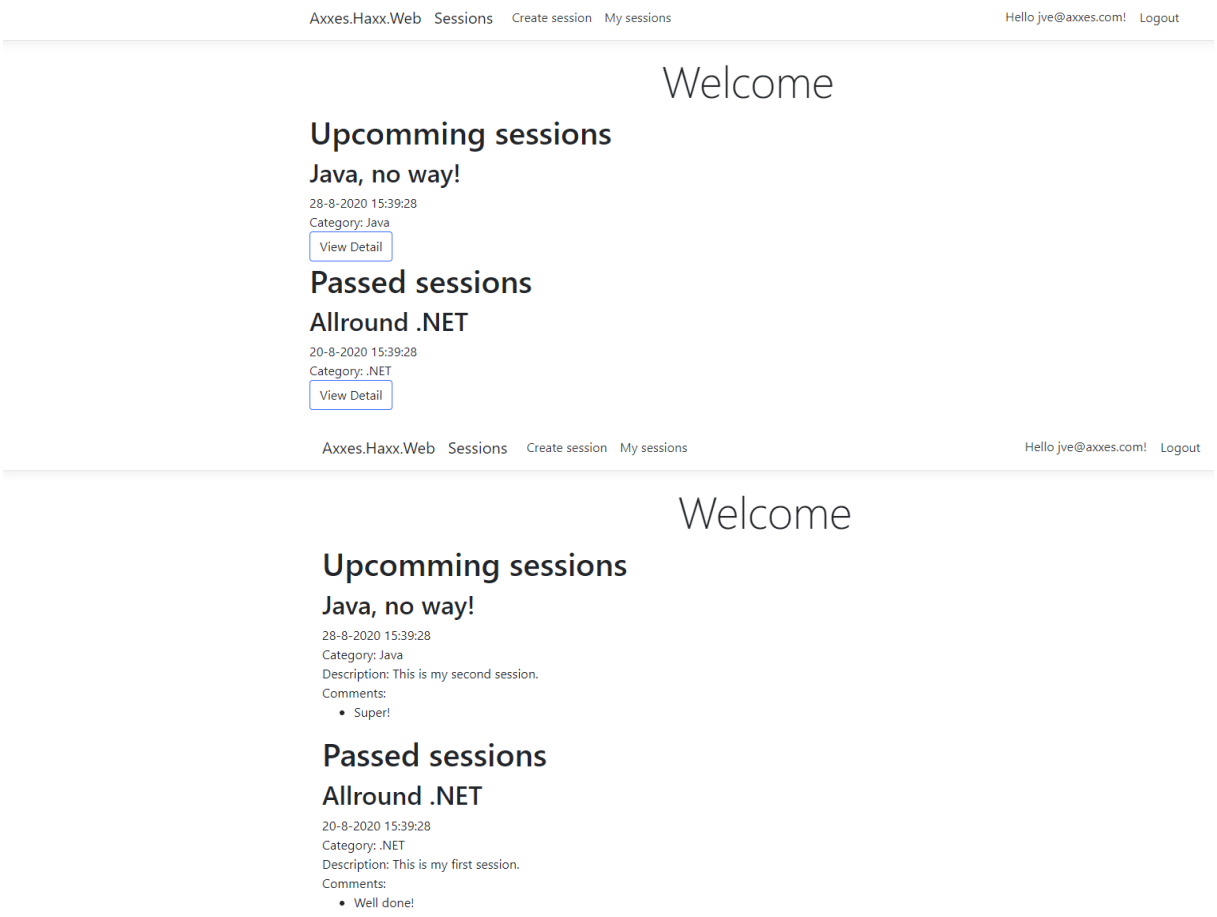
4. Next, let's create the **partial view** that will be returned when the AJAX request for session detail is made. The partial view for the above AJAX action should stay in **\Views\Home_SessionDetails.cshtml**.

5. Now, let's edit the sessions display template **\Views\Shared\DisplayTemplates\SessionViewModel.cshtml** and add the AJAX

call in it. Research “Jquery-Ajax-Unobtrusive” on how to add a link with an Ajax request. You can use the following code snippet to add the needed js file to the project:

```
@section scripts {
        <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-ajax-
unobtrusive/3.2.6/jquery.unobtrusive-ajax.min.js" integrity="sha512-
DedNBWPF0hLGUPNbCYfj8qjlEnNE92Fqn7xd3Sscfu7ipy7Zu33unHdugqRD3c4Vj7/yLv+slqZhMls/4Oc7Zg
==" crossorigin="anonymous"></script>}
```

GOAL:



ONCE YOU ARE FINISHED WITH STEP 1, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.

Step 2: Create New Session

The next feature to be implemented in the Conference management system is “**Create New Session**“. This will require **creating a “New Session”** form (Razor view) + input **model** for the form + **controller action** to handle the submitted form data.

1. First, define the **input model** for the create Session form. It will hold all session properties that the user will fill when creating a session. Create the class **Models\SessionInputModel.cs**:

2. It is a good idea to attach **validation annotations** like **[Required]** and **[MaxLength]** for each property to simplify the validation of the form. Check the Session Entity for which ones you’ll need.

3. Next, **create the "New Session" form**. The easiest way to start is by using the **Razor view generator** in Visual Studio. Create a folder "**\Views\Session**". Right click on the "**Sessions**" folder and choose [Add] -> [View…] and generate a Create template. Now **customize the generated form** to your needs.
    a. Remove the "Back to List" link.
    b. Add **[Cancel]** link to "My Sessions", just after the **[Create]** button. (Tip: try to use Html.ActionLink)
    c. Make the IsPublic field checked by default.

6. Create **SessionController** to handle the actions related to Sessions: **create** / **edit** / **delete** / **list** Sessions.

7. Add the **"Create" action** (HTTP GET) to display the "New Session" form

8. Build the project, run it and **test** the new functionality (**login** and click **[Create Session]**). The "New Session" form should be rendered in the Web browser

Create New Session Logic

1. After the "Create New Session" form is ready, it is time to write the **logic** behind it. It should create a new Session and save it in the database. The correct way to handle form submissions in ASP.NET MVC is by **HTTP POST** action in the controller behind the form. Let's write the "Create New Session" logic.

2. First, extend the **BaseController** to inherit the database context

3. Next, write the action to handle **POST \Sessions\CreateSession.** After creating the session. Redirect to the **MySessions** action.

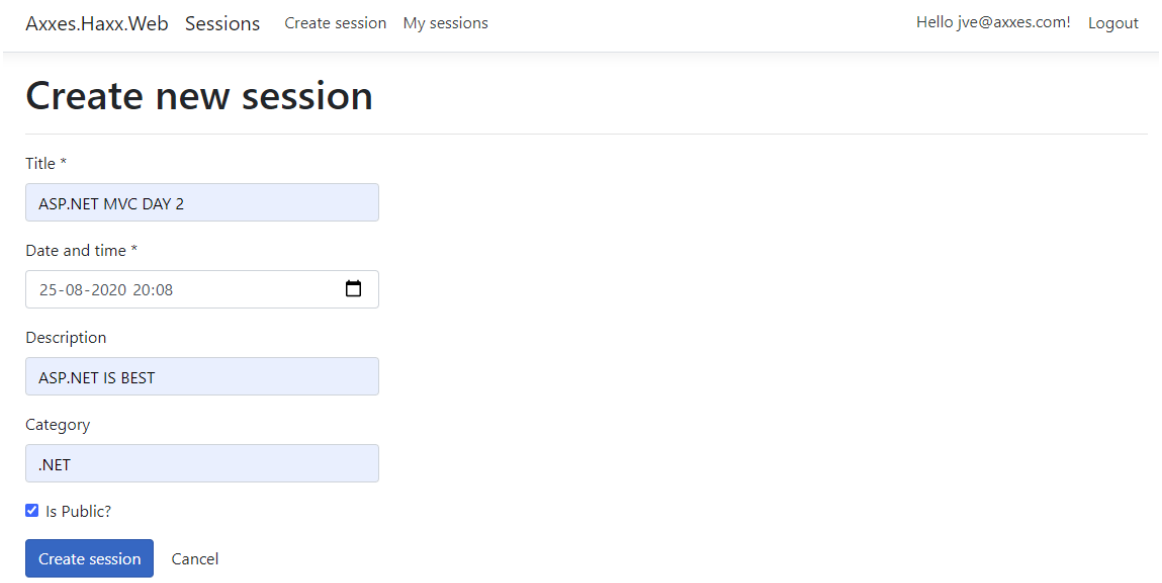
4. Run and **test the code**. Now creating Sessions almost works:

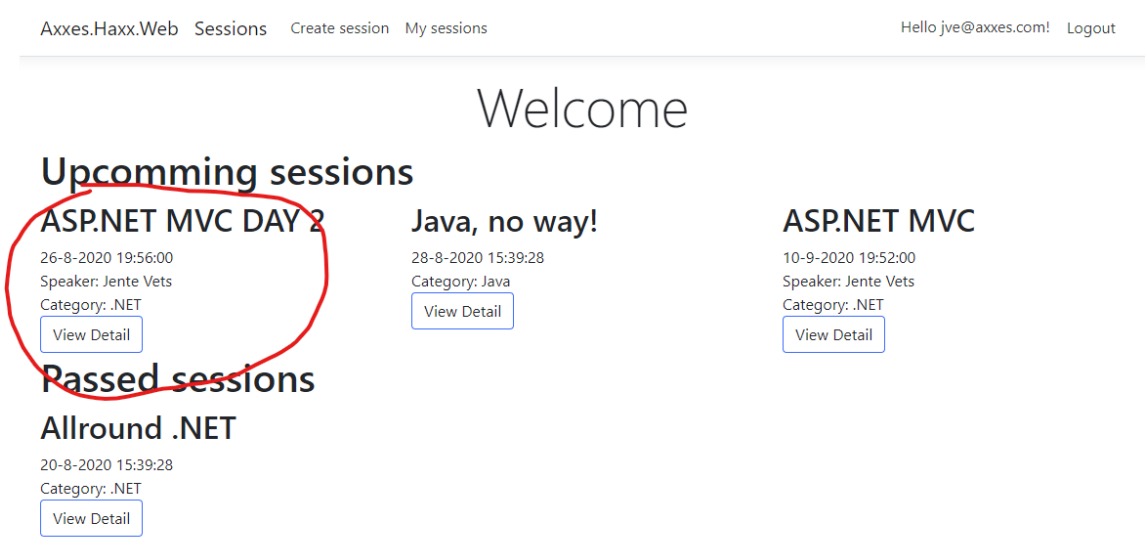
5. When the **form is submitted**, the **result** can be a 404. This is quite normal. The "**MySessions**" action in the **SessionController** is missing. Let's define the action "**MySession**" and the view behind it in the **SessionContoller**. Create an empty view **\Views\Sessions\MySessions.cshtml** (it will be implemented later).

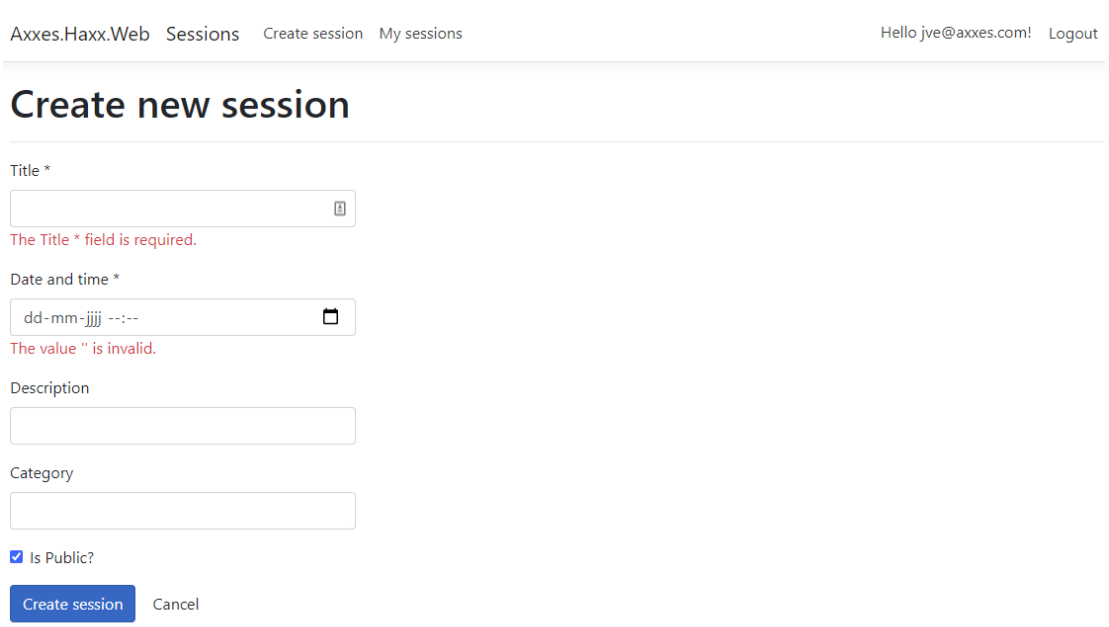
6. Now run and test the application again. The create should work.

7. Last, make sure that if an anonymous user surfs to **/Session/CreateSession or /Session/MySessions**, he/she is redirected to the login page. Anonymous users shouldn't be able to access the create or my sessions page. (Tip: this can be easily done with one attribute)

GOAL:


Axxes.Haxx.Web Sessions Create session My sessions Hello jve@axxes.com! Logout
Create new session
Title *
ASP.NET MVC DAY 2
Date and time *
25-08-2020 20:08
Description
ASP.NET IS BEST
Category
.NET
Is Public?
Create session
Cancel


Axxes.Haxx.Web Sessions Create session My sessions Hello jve@axxes.com! Logout
Welcome
Upcomming sessions
ASP.NET MVC DAY 2
26-8-2020 19:56:00
Speaker: Jente Vets
Category: .NET
View Detail
Java, no way!
28-8-2020 15:39:28
Category: Java
View Detail
ASP.NET MVC
10-9-2020 19:52:00
Speaker: Jente Vets
Category: .NET
View Detail
Passed sessions
Allround .NET
20-8-2020 15:39:28
Category: .NET
View Detail


Axxes.Haxx.Web Sessions Create session My sessions Hello jve@axxes.com! Logout
Create new session
Title *
The Title * field is required.
Date and time *
dd-mm-jjjj --:--
The value '' is invalid.
Description
Category
Is Public?
Create session
Cancel

ONCE YOU ARE FINISHED WITH STEP 2, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.

Step 3: List My Sessions

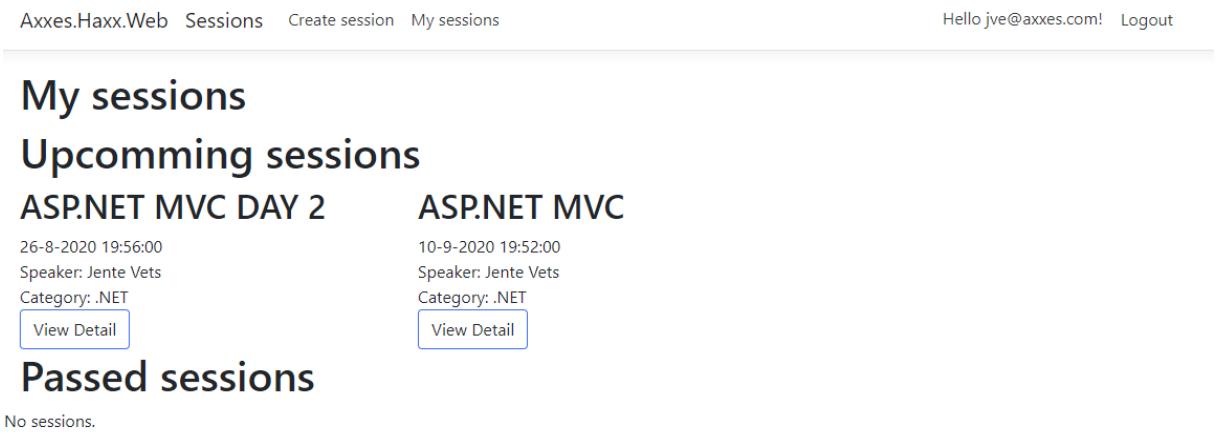
Now let's list current user's sessions at its personal sessions page: **`\Session\MySessions`**.

1. First, add the HTTP GET controller action "**MySessions**" in **SessionController** that will display only the current user's sessions. We want to have a similar view as in the index page, with upcoming and passed sessions.

2. Next, create the view **MySessions.cshtml** behind the above action. It is essentially the same like the **Index.cshtml** view of the **HomeController**. Duplicating code is very bad practice, right? So let's **reuse the code**. First, extract a partial view **\Views\Shared_Sessions.cshtml**, then reference it from **\Views\Session\MySessions.cshtml** and again from **\Views\Home\Index.cshtml**.

3. Test the application both on the homepage as on my sessions page. The result should be similar.

GOAL:



ONCE YOU ARE FINISHED WITH STEP 3, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.

Step 4: Edit Existing Session Form

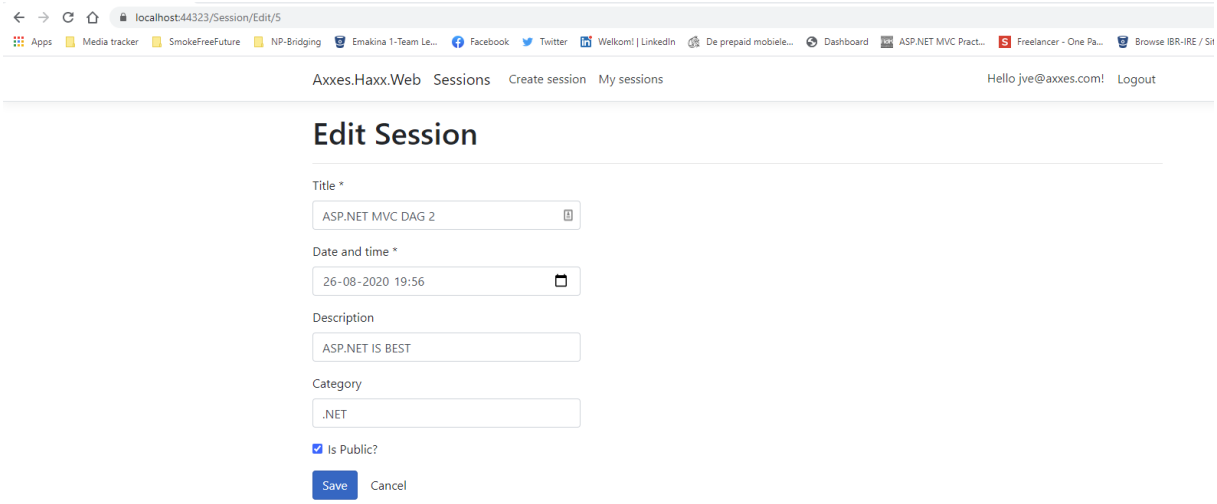
1. The "**Edit Session**" functionality is very similar to "**Create Session**". It uses the same input model **Sessions.Web.Models.SessionInputModel**. Create a view **\Views\Session\Edit.cshtml**.

2. Next, write the controller action for editing sessions in the **SessionController**.Write a HTTP **GET action "Edit"** to prepare the form for editing a session. In case of invalid Session ID, **redirect to** "**My Sessions**". The session should only be loaded in case the user has permissions to edit it.

3. Write a HTTP **POST action "Edit"** to save the changes after submitting the session editing form. It **checks for validation errors**. In case of validation errors, the same form is rendered again (it will show the validation errors). Finally, the **Edit** method modifies the database and redirects to "**My Sessions**".

4. Run and test the new functionality "Edit Session"

GOAL:



ONCE YOU ARE FINISHED WITH STEP 4, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.

Step 5. Delete Existing Event Form

1. Create a form and controller action, **similar to "Edit Session"**. The form should load the session data in read-only mode so that the user can't change the data in the form. There also should be **[Confirm]** and **[Cancel]** buttons. After deleting a session, I should be redirected to the "My Sessions" page.

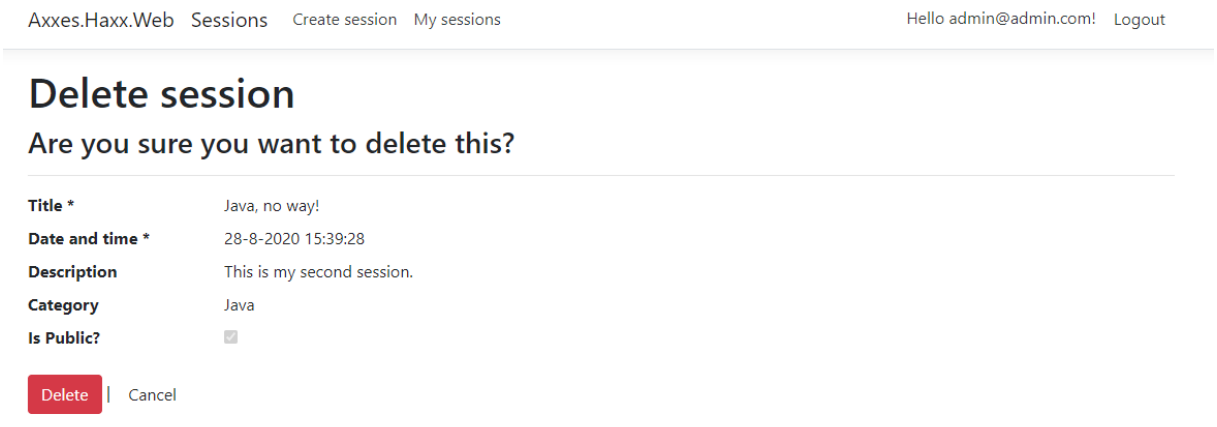
Delete Existing Event Logic

1. The logic behind the "Delete Session" form is **similar to "Edit Session"**. Try to implement is yourself.

2. Note that the .NET CLI requires overloaded methods to have a unique parameter signature (same method name but different list of parameters). However, here you need two Delete methods -- one for GET and one for POST -- that both have the same parameter signature. See if you can solve this problem without adding another parameter in one of the Delete actions.

3. Test the delete functionality. It should work.

GOAL:

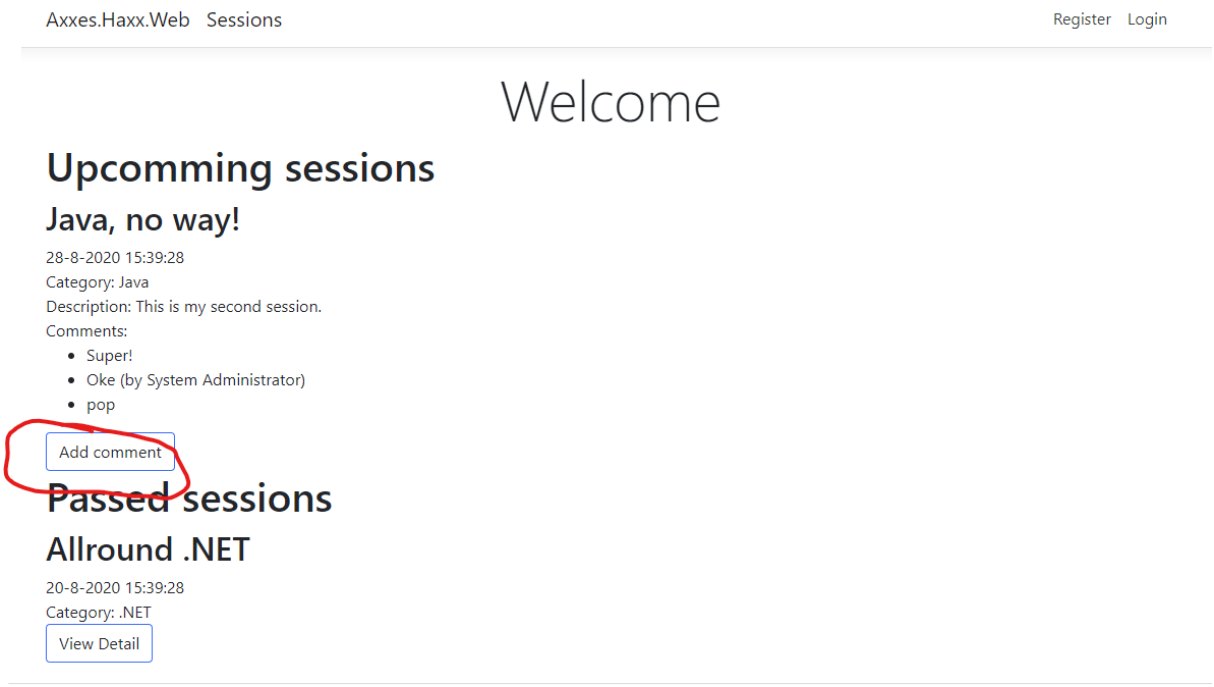


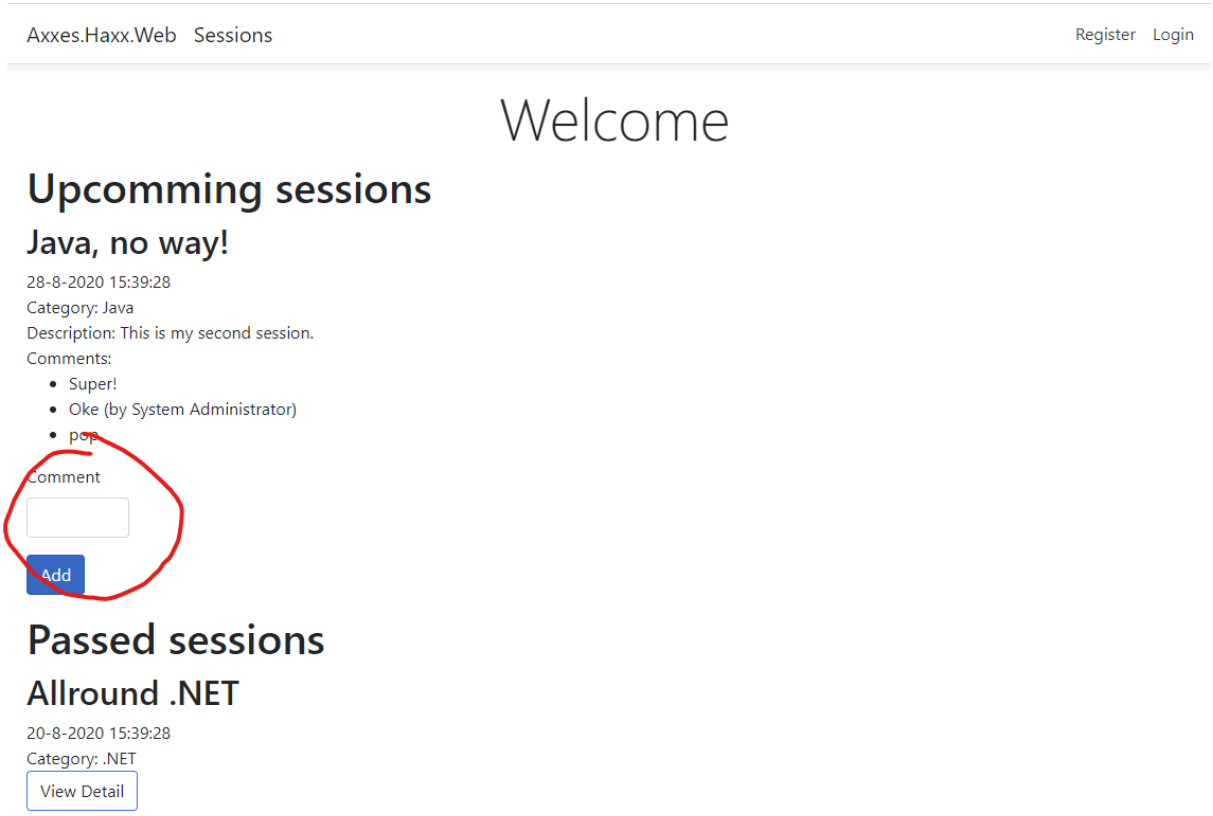
ONCE YOU ARE FINISHED WITH STEP 5, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.

Step 6: Add Comments with AJAX

For each session, implement an **[Add Comment]** button that shows (with AJAX) a new comment form. After the form is submitted, it creates a comment and appends it after the other comments for the current event. Comments can be added by the currently logged-in user and anonymously (without login).

GOAL:





ONCE YOU ARE FINISHED WITH STEP 6, PLEASE SEND ME A PRIVATE MESSAGE VIA TEAMS OR ZOOM.