Dokumentacja techniczna Projektu na programowanie zaawansowane

Autorzy projektu: Wiktor Piechowiak, Radek Domagała

Tytuł projektu: ITstudy

Krótki opis działania projektu:

- Forum internetowe – Strona internetowa (ukierunkowana na studentów IT) pozwalająca na dyskusje online. Pozwoli na tworzenie nowych tematów lub dodawanie wpisów do już aktywnych tematów. Pozwoli też na prostą moderacje dla użytkowników należących do administracji.

Specyfikacja wykorzystanych technologii:

- C# .NET 8, Postgres, ASP.Net Core MVC
- Wykorzystane pakiety NuGet: Microsoft.AspNetCore.Identity.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Tools, Npgsql.EntityFrameworkCore.PostgreSQL

instrukcje pierwszego uruchomienia projektu:

- Zainstalować postgresa https://www.postgresql.org/download/windows/
- upewnić się że postgres działa, istnieje (domyślny) użytkownik postgres, ma domyślne hasło (w przypadku innego użytkownika/hasła, można zmienić connection stringa projektu w appsettings.json)
- polecenie “update-database” w konsoli nuget
- Można uruchomić projekt (nie potrzeba gotowej bazy danych, projekt sam seeduje dane)

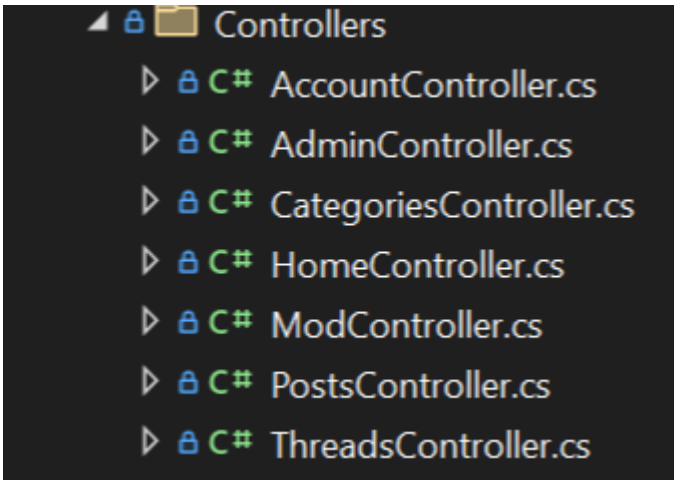
opis struktury projektu:

- Projekt napisany został w ASP.net Core MVC - z tym typem projektów mieliśmy do czynienia na zajęciach, dlatego go wybieraliśmy
- Wykorzystaliśmy Postgresa ponieważ mieliśmy z nim zajęciach w poprzednich semestrach i czuliśmy się z nim swobodnie, a implementacja go jest prosta i przyjemna w ASP.net (dodanie jednej paczki Nuget)
- Projekt wykorzystuje Kontrolery, Modele, View i ViewModels

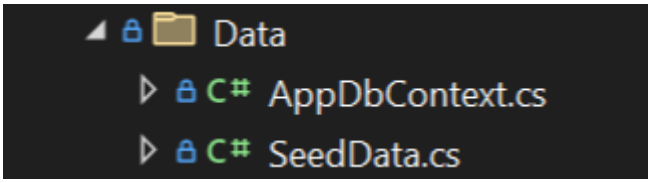
- ViewModel zostały wykorzystane w celu uproszczenia/zwiększenia przejrzystości kodu - pozwala w łatwy sposób połączyć 2 modele i podać to do View

Struktura plików projektu:

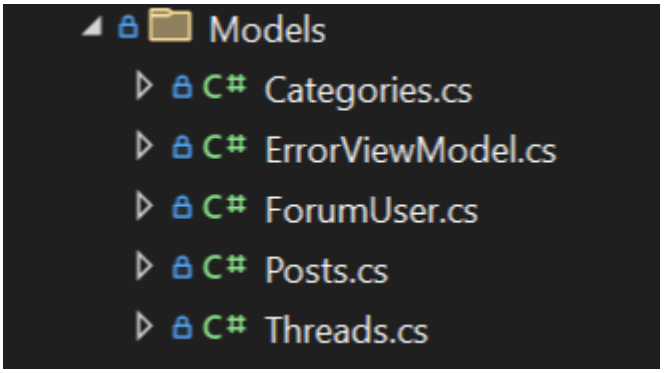
Controllers - zawiera kontrolery



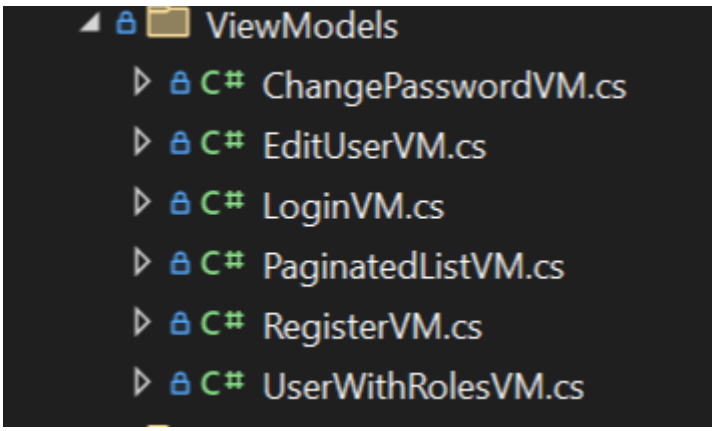
Data - Zawiera plik konfiguracyjny bazy danych (AppDbContext.cs) oraz plik potrzebny do automatycznego generowania danych (SeedData.cs).



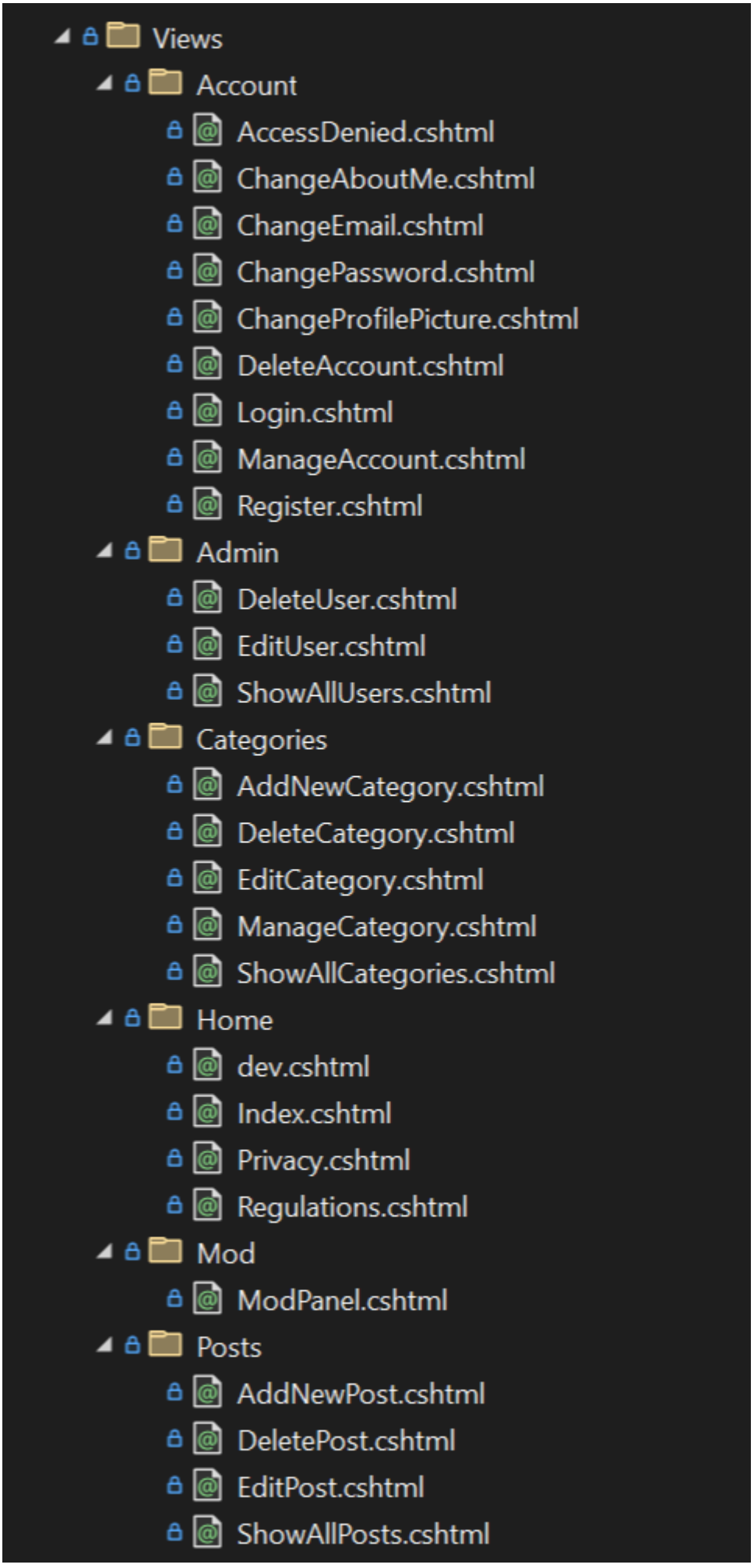
Models - zawiera modele (opis obiektów/tabel, potrzeby do działania EntityFramework)

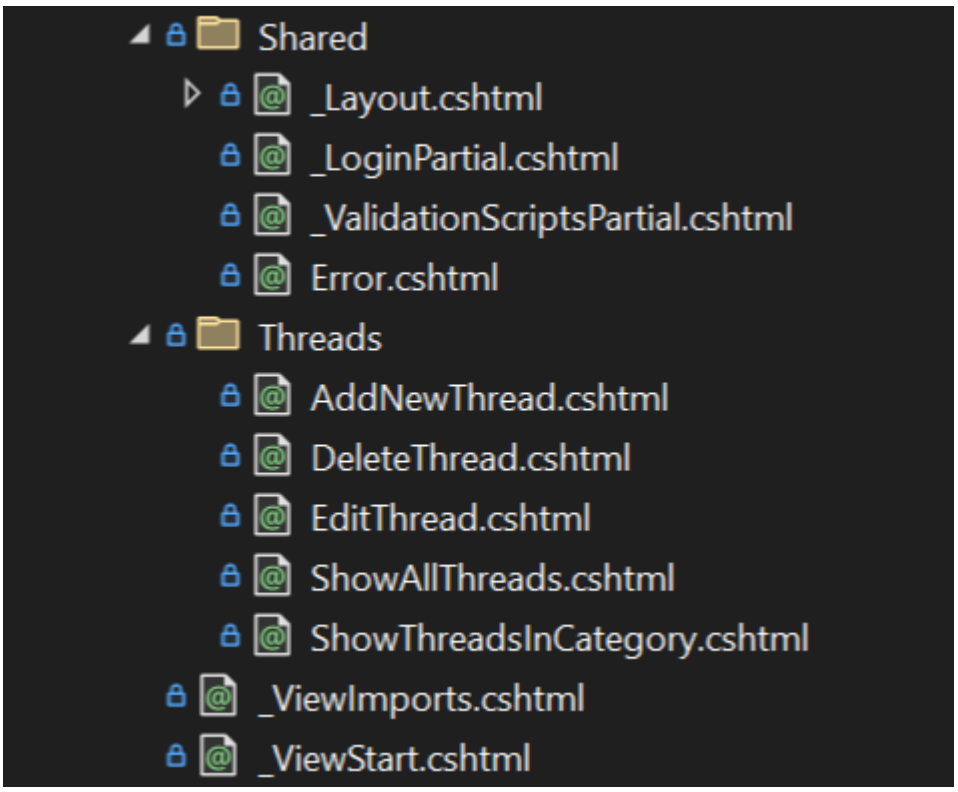


ViewModels - zawiera ViewModels, wykorzystane w projekcie do uproszczenia logiki pobierania/wysyłania danych przez Views



Views - zawiera Views, czyli pliki wyświetlające naszą aplikację


Views
Account
AccessDenied.cshtml
ChangeAboutMe.cshtml
ChangeEmail.cshtml
ChangePassword.cshtml
ChangeProfilePicture.cshtml
DeleteAccount.cshtml
Login.cshtml
ManageAccount.cshtml
Register.cshtml
Admin
DeleteUser.cshtml
EditUser.cshtml
ShowAllUsers.cshtml
Categories
AddNewCategory.cshtml
DeleteCategory.cshtml
EditCategory.cshtml
ManageCategory.cshtml
ShowAllCategories.cshtml
Home
dev.cshtml
Index.cshtml
Privacy.cshtml
Regulations.cshtml
Mod
ModPanel.cshtml
Posts
AddNewPost.cshtml
DeletePost.cshtml
EditPost.cshtml
ShowAllPosts.cshtml



Wykorzystane Modele:

Categories:

Wykorzystywany do segregacji tematów w grupy, zarządzać nimi mogą tylko użytkownicy administracji/moderacji, wyświetlać mogą je wszyscy, nawet nie zarejestrowani użytkownicy.

Pola modelu:

public int Id - Identyfikator, wymagany, ustawiany automatycznie, inkrementalnie

public string? Name - Nazwa, wymagana, nie więcej niż 100 znaków.

public string? Description - Opis, wymagany, nie więcej niż 255 znaków.

Threads:

Tematy tworzone przez użytkowników, główny element przechowujący posty, przydzielane są do kategorii. Wyświetlanie dostępne jest nawet dla niezarejestrowanych użytkowników, dodawanie dostępne jest dla każdego zalogowanego, edycja i usuwanie dostępne są tylko dla własnych postów, chyba, że jest się adminem/moderatorem.

Pola modelu:

public int Id - Identyfikator, klucz główny, wymagany, ustawiany automatycznie, inkrementalnie

public string? Title - Tytuł, wymagany

public DateTime CreatedAt - Data powstania, domyślnie ustawiona jako aktualny czas przy tworzeniu.

public int Views - Wyświetlenia, bazowa wartość 0, zwiększają się przy odwiedzaniu tematu przez użytkowników.

public string? UserId - ID Usera, wykorzystywane do przechowywania identyfikatora użytkownika, który stworzył temat.

public int CategoryId - ID Kategorii, wykorzystywana do przechowywania identyfikatora kategorii w której znajduje się temat.

public virtual ForumUser? User - Referencja do tabeli ForumUser, służy do pobierania danych odnośnie użytkownika, który utworzył temat.

public virtual Categories? Category - Referencja do tabeli Category, służy do pobierania danych odnośnie kategorii w której znajduje się temat.

Posts:

Wykorzystywany do przechowywania postów użytkowników na forum. Nie zalogowani użytkownicy mogą wyświetlać wszystkie posty, Zalogowani użytkownicy mogą dodawać swoje własne oraz je edytować jak i usuwać. Administratorzy oraz moderatorzy również posiadają opcję edycji i usuwania (oni mogą modyfikować każdy post)

Pola modelu:

public int Id - zawiera id posta

public string? Content - zawiera zawartość posta (tekst)

public DateTime CreatedDate - data utworzenia posta (automatycznie dodawana aktualna data)

public bool Edited - Zmienna wskazująca czy post został edytowany

public string? UserId - Zmienna wskazująca który użytkownik napisał post

public int ThreadId - Zmienna wskazująca do którego wątku/tematu należy po*st*

public virtual ForumUser? User - Referencja do tabeli ForumUser, służy do pobierania danych odnośnie użytkownika który napisał post.

public virtual Threads? Thread - Referencja do tabeli Threads, służy do pobierania danych odnośnie wątku/tematu do którego odwołuje się dany post

ForumUser:

Wykorzystywany do przechowywania informacji na temat użytkowników, obsługi ich rejestracji/logowania oraz aktywności na forum. Jest to model rozszerzający Identity.EntityFrameworkCore. Zawiera w sobie dodatkowe pola których nie ma Identity

Warto nadmienić że w naszym ForumUser nie ma pól odpowiadających za hasło oraz E-mail - wykorzystujemy je z Identity. Email jest poddawany standardowej weryfikacji (czy jest to email)

hasło ma wylistowane wymagania w opcjach Identity (plik Program.cs, builder.services.AddIdentity()). Jedynym prawdziwym wymaganiem jest min. 8 znaków

Role userów przechowujemy w tabelach tworzonych przez Identity - AspNetRoles, oraz AspNetUserRoles.

Pola Modelu:

public string? FirstName - Imię użytkownika forum

public string? LastName - nazwisko użytkownika forum

public DateTime JoinDate - data dołączenia użytkownika do forum (automatycznie tworzona w trakcie rejestracji konta)

public string? ProfilePictureURL - String domyślnie pusty, użytkownik może go modyfikować z poziomu panelu użytkownika - po podaniu url do dostępnego

publicznie obrazka z rozszerzeniem wspieranym przez przeglądarki, będzie wyświetlany jako avatar na forum

public string? Bio - String domyślnie pusty, użytkownik może go zmodyfikować z poziomu panelu użytkownika - po dodaniu dowolnego tekstu, będzie wyświetlany na profilu użytkownika

ErrorViewModel:

standardowy model ASP.NET

Wykorzystane kontrolery:

CategoriesController:

Metody:

ShowAllCategories(int pageNumber = 1, int pageSize = 10) - Wyświetla listę wszystkich kategorii, razem z paginacją. Wykorzystuje GET

```
@model ITstudyv4.ViewModels.PaginatedListVM<Categories>
@{
    ViewData["Title"] = "Wszystkie kategorie";
}

<h1>Wszystkie Kategorie</h1>
<table class="table table-striped table-hover">
    <thead class="table-dark">
        <tr>
            <th>Nazwa</th>
            <th>Opis</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var category in Model.Items)
        {
            <tr>
                <td>
                    <a asp-controller="Threads" asp-action="ShowThreadsInCategory" asp-route-categoryId="@category.Id">
                        @category.Name
                    </a>
                </td>
                <td>@category.Description</td>
            </tr>
        }
    </tbody>
</table>

<div class="pagination">
    @if (Model.HasPreviousPage)
    {
        <a href="@Url.Action("ShowAllCategories", new { pageNumber = Model.CurrentPage - 1, pageSize = Model.PageSize })" class="btn btn-primary m-1">Poprzednia</a>
    }
    <span>Strona @Model.CurrentPage z @Model.TotalPages</span>
    @if (Model.HasNextPage)
    {
        <a href="@Url.Action("ShowAllCategories", new { pageNumber = Model.CurrentPage + 1, pageSize = Model.PageSize })" class="btn btn-primary m-1">Następna</a>
    }
</div>
```

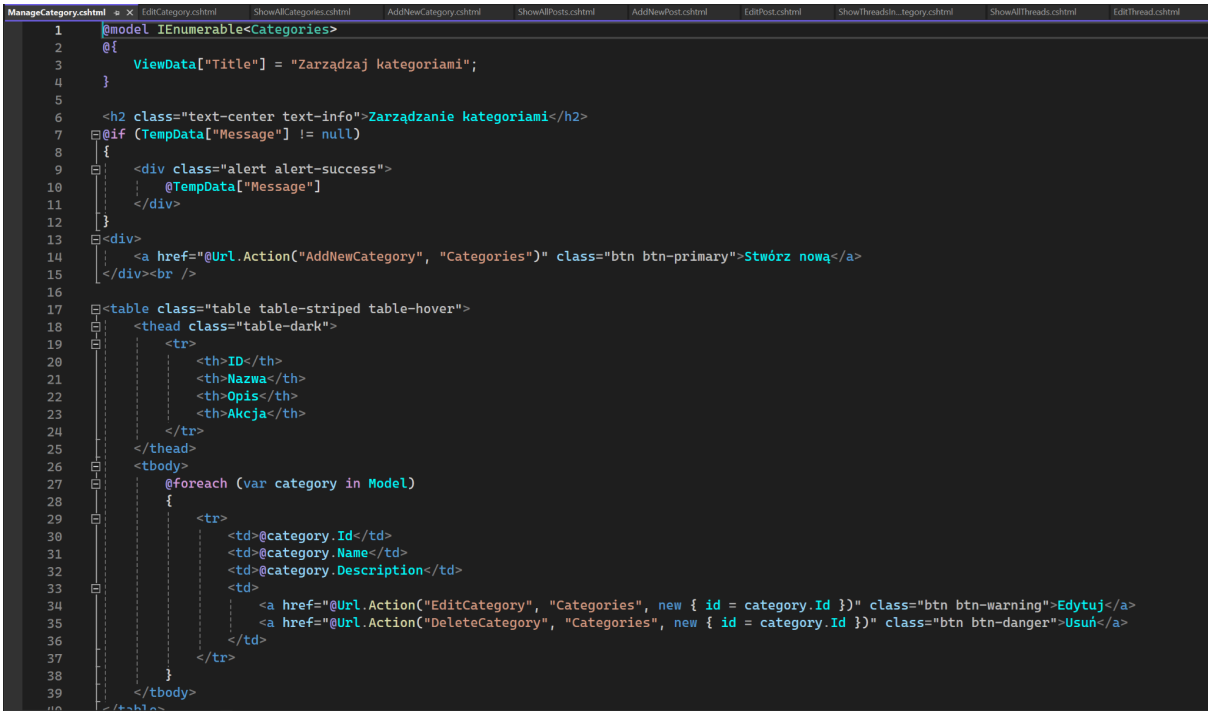
AddNewCategory([Bind("Name,Description")] - Dodaje nową kategorię, przypisuje nazwę i opis, wpisane przez użytkownika w widoku. Wykorzystuje POST. Autoryzacja pozwala na zrobienie tego tylko Adminowi lub Moderatorowi

```cshtml
@model Categories
@{
    ViewData["Title"] = "Dodaj nową kategorię";
}

<h2 class="text-center text-info">Dodawanie nowej kategori</h2>

<form method="post">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Description" class="control-label"></label>
        <input asp-for="Description" class="form-control" />
        <span asp-validation-for="Description" class="text-danger"></span>
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Dodaj</button>
</form>

@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

EditCategory(int id, [Bind("Id,Name,Description")] - Edytuje wybraną kategorię, przypisuje Id, ukryte i przypisane automatycznie, oraz nazwę i opis wpisane przez użytkownika w widoku. Wykorzystuje POST. Autoryzacja pozwala na zrobienie tego tylko Adminowi lub Moderatorowi

```cshtml
@model Categories
@{
    ViewData["Title"] = "Edycja kategori";
}

<h2 class="text-center text-info">Edycja Kategorii</h2>

<form method="post">
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.Id)

    <div class="form-group">
        <label for="Name">Nazwa</label>
        @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
    </div>

    <div class="form-group">
        <label for="Description">Opis</label>
        @Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Zapisz zmianę</button>
</form>
```

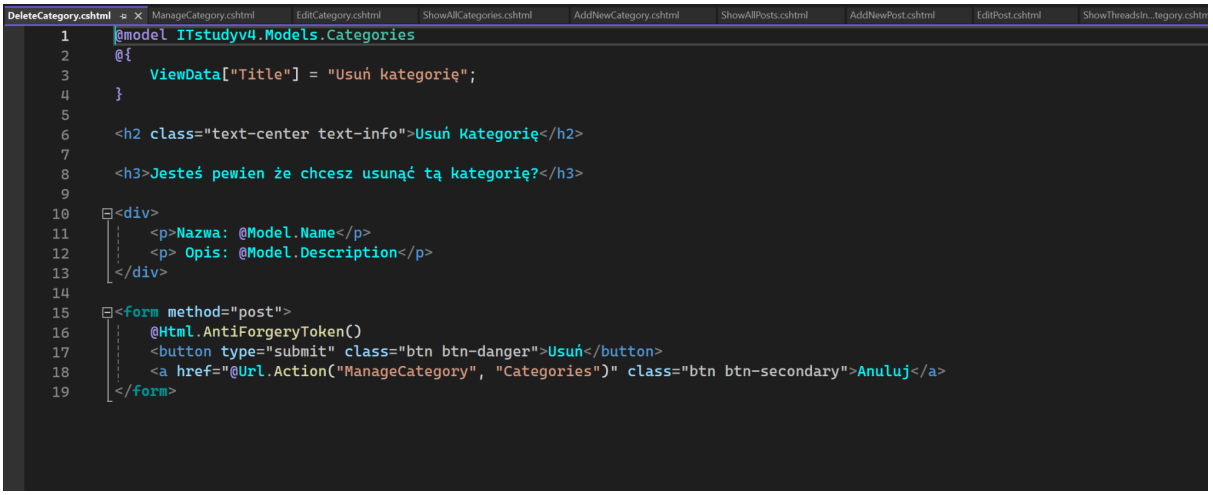
ManageCategory(int id, [Bind("id,Name,Description")] - Wyświetla listę wszystkich kategorii stworzonych na stronie wraz z opcją edycji lub usunięcia danej kategorii. Autoryzacja pozwala na zrobienie tego tylko Adminowi lub Moderatorowi. Wykorzystuje POST aby przejść do DeleteCategory() oraz EditCategory().



```cshtml
@model IEnumerable<Categories>
@{
    ViewData["Title"] = "Zarządzaj kategoriami";
}

<h2 class="text-center text-info">Zarządzanie kategoriami</h2>
@if (TempData["Message"] != null)
{
    <div class="alert alert-success">
        @TempData["Message"]
    </div>
}
<div>
    <a href="@Url.Action("AddNewCategory", "Categories")" class="btn btn-primary">Stwórz nową</a>
</div><br />

<table class="table table-striped table-hover">
    <thead class="table-dark">
        <tr>
            <th>ID</th>
            <th>Nazwa</th>
            <th>Opis</th>
            <th>Akcja</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var category in Model)
        {
            <tr>
                <td>@category.Id</td>
                <td>@category.Name</td>
                <td>@category.Description</td>
                <td>
                    <a href="@Url.Action("EditCategory", "Categories", new { id = category.Id })" class="btn btn-warning">Edytuj</a>
                    <a href="@Url.Action("DeleteCategory", "Categories", new { id = category.Id })" class="btn btn-danger">Usuń</a>
                </td>
            </tr>
        }
    </tbody>
```

DeleteCategory - Wyświetla szczegóły wybranej kategorii przed jej usunięciem. Użytkownik musi być autoryzowany jako Admin lub Moderator. Wykorzystuje metodę GET.

DeleteConfirmed - Usuwa kategorię z bazy danych na podstawie jej identyfikatora. Użytkownik musi być autoryzowany jako Admin lub Moderator. Wykorzystuje metodę POST

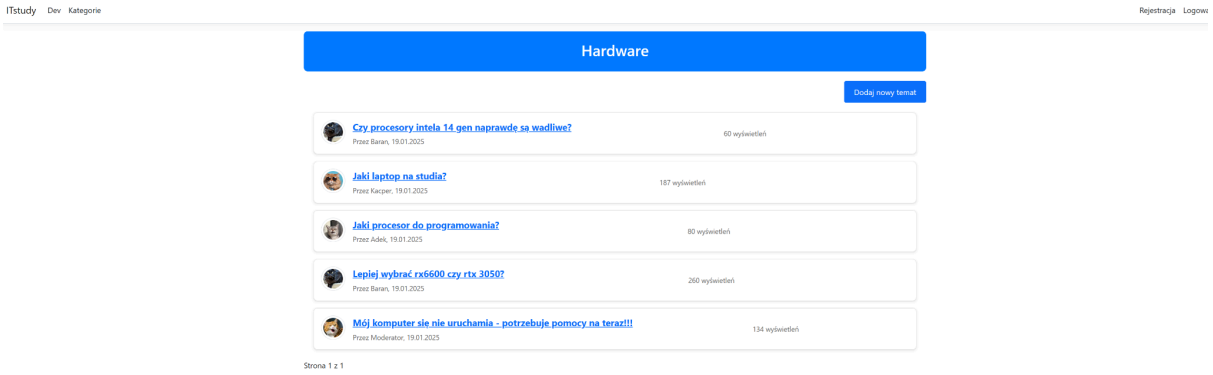


```cshtml
@model ITstudyv4.Models.Categories
@{
    ViewData["Title"] = "Usuń kategorię";
}

<h2 class="text-center text-info">Usuń Kategorię</h2>

<h3>Jesteś pewien że chcesz usunąć tą kategorię?</h3>

<div>
    <p>Nazwa: @Model.Name</p>
    <p> Opis: @Model.Description</p>
</div>

<form method="post">
    @Html.AntiForgeryToken()
    <button type="submit" class="btn btn-danger">Usuń</button>
    <a href="@Url.Action("ManageCategory", "Categories")" class="btn btn-secondary">Anuluj</a>
</form>
```

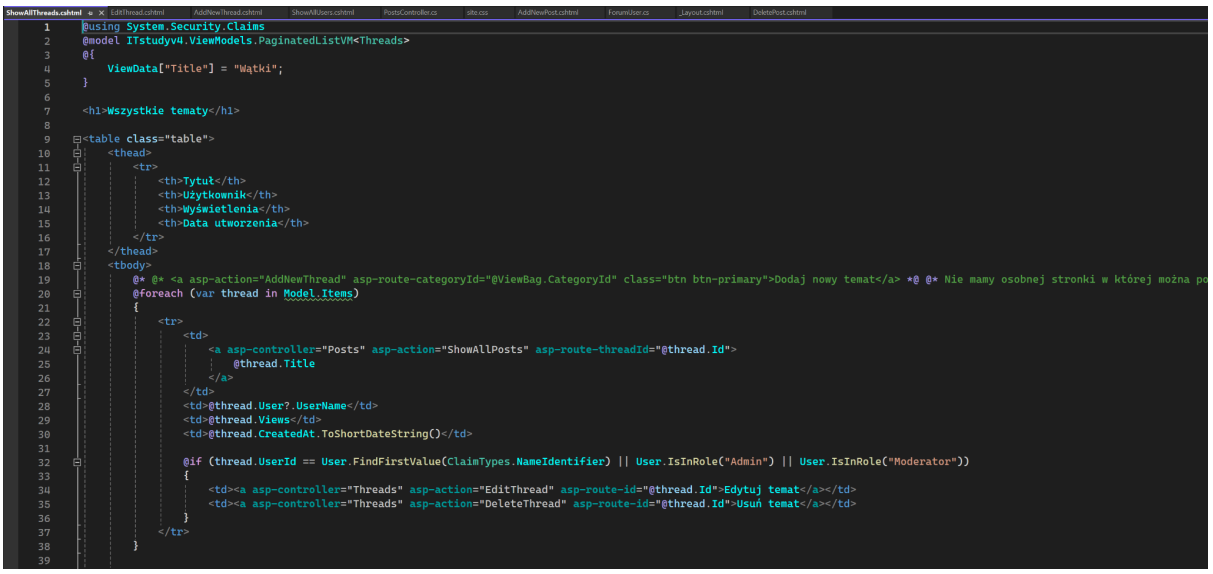
ThreadsController:

ShowThreadsInCategory(int categoryId, int pageNumber = 1, int pageSize = 10) - Wyświetla listę tematów w danej kategorii, razem z paginacją. Wykorzystuje GET

```razor
@using System.Security.Claims
@model ITstudyv4.ViewModels.PaginatedListVM<Threads>
@{
    ViewData["Title"] = "Wątki";
}

<div id="thread-title-card">
    <h1>@ViewBag.CategoryName</h1>
    <p>@ViewBag.Description</p>
</div>

<span class="add-btn">
    <a asp-action="AddNewThread" asp-route-categoryId="@ViewBag.CategoryId" class="btn btn-primary">Dodaj nowy temat</a>
</span>

<div class="threads-container">
    @foreach (var thread in Model.Items)
    {
        <div class="thread-container">
            <div class="thread-con-left">
                <div class="thread-avatar">
                    <img src="@thread.User?.ProfilePictureURL" alt="Zdjęcie profilowe" class="img-thumbnail align-items-center rounded-circle" style="width: 50px; height: 50px; object-fit:cover;" />
                </div>
                <div class="thread-content">
                    <p>
                        <a asp-controller="Posts" asp-action="ShowAllPosts" asp-route-threadId="@thread.Id">
                            @thread.Title
                        </a>
                    </p>
                    <p>Przez @thread.User?.UserName, @thread.CreatedAt.ToShortDateString()</p>
                </div>
            </div>

            <div class="thread-info">
                <td>@thread.Views wyświetleń</td>
            </div>

            <div class="thread-options">
                @if (thread.UserId == User.FindFirstValue(ClaimTypes.NameIdentifier) || User.IsInRole("Admin") || User.IsInRole("Moderator"))
                {
                    <td><a asp-controller="Threads" asp-action="EditThread" asp-route-id="@thread.Id">Edytuj temat</a></td>
                    <td><a asp-controller="Threads" asp-action="DeleteThread" asp-route-id="@thread.Id">Usuń temat</a></td>
                }
            </div>
        </div>
    }
</div>
```

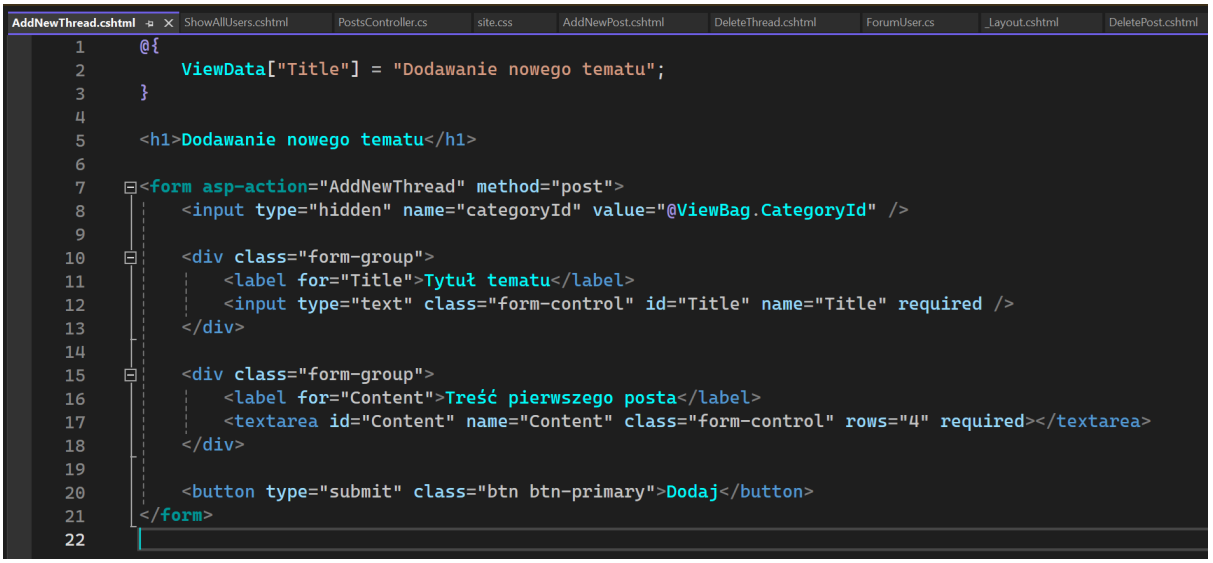


ShowAllThreads(int pageNumber = 1, int pageSize = 10) - Wyświetla listę wszystkich tematów, razem z paginacją. Wykorzystuje GET

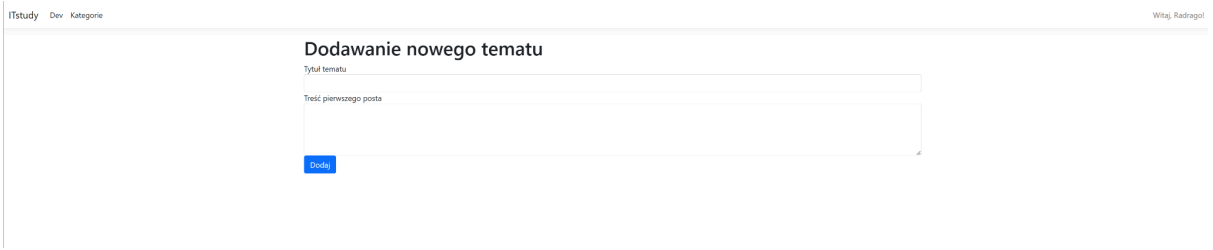


```
@using System.Security.Claims
@model ITstudyv4.ViewModels.PaginatedListVM<Threads>
@{
    ViewData["Title"] = "Wątki";
}

<h1>Wszystkie tematy</h1>

<table class="table">
    <thead>
        <tr>
            <th>Tytuł</th>
            <th>Użytkownik</th>
            <th>Wyświetlenia</th>
            <th>Data utworzenia</th>
        </tr>
    </thead>
    <tbody>
        @* @* <a asp-action="AddNewThread" asp-route-categoryId="@ViewBag.CategoryId" class="btn btn-primary">Dodaj nowy temat</a> *@ @* Nie mamy osobnej stronki w której można po
        @foreach (var thread in Model.Items)
        {
            <tr>
                <td>
                    <a asp-controller="Posts" asp-action="ShowAllPosts" asp-route-threadId="@thread.Id">
                        @thread.Title
                    </a>
                </td>
                <td>@thread.User?.UserName</td>
                <td>@thread.Views</td>
                <td>@thread.CreatedAt.ToShortDateString()</td>

                @if (thread.UserId == User.FindFirstValue(ClaimTypes.NameIdentifier) || User.IsInRole("Admin") || User.IsInRole("Moderator"))
                {
                    <td><a asp-controller="Threads" asp-action="EditThread" asp-route-id="@thread.Id">Edytuj temat</a></td>
                    <td><a asp-controller="Threads" asp-action="DeleteThread" asp-route-id="@thread.Id">Usuń temat</a></td>
                }
            </tr>
        }
```

AddNewThread(int categoryId, [Bind("Title")] - Dodaje nowy temat, przypisuje nazwę wpisaną przez użytkownika w widoku. Wykorzystuje POST. Autoryzacja pozwala na wykonanie tego tylko przez zalogowanego użytkownika

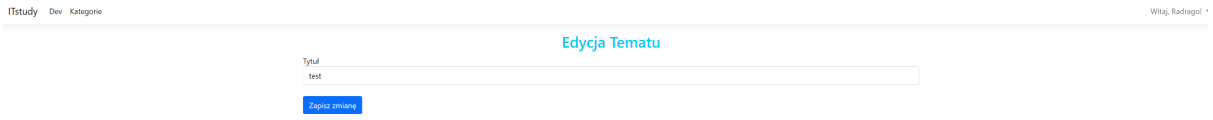


```
@{
    ViewData["Title"] = "Dodawanie nowego tematu";
}

<h1>Dodawanie nowego tematu</h1>

<form asp-action="AddNewThread" method="post">
    <input type="hidden" name="categoryId" value="@ViewBag.CategoryId" />

    <div class="form-group">
        <label for="Title">Tytuł tematu</label>
        <input type="text" class="form-control" id="Title" name="Title" required />
    </div>

    <div class="form-group">
        <label for="Content">Treść pierwszego posta</label>
        <textarea id="Content" name="Content" class="form-control" rows="4" required></textarea>
    </div>

    <button type="submit" class="btn btn-primary">Dodaj</button>
</form>
```



EditThread(int id, [Bind("Id,Title")] - Edytuje wybrany temat, przypisuje zmieniony tytuł, wpisany przez użytkownika w widoku. Wykorzystuje POST. Autoryzacja pozwala na wykonanie tego tylko przez autora lub admina/moderatora

```cshtml
@model Threads
@{
    ViewData["Title"] = "Edycja tematu";
}

<h2 class="text-center text-info">Edycja Tematu</h2>

<form method="post">
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.Id)

    <div class="form-group">
        <label for="Title">Tytuł</label>
        @Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Zapisz zmianę</button>
</form>
```



DeleteThread(int? id) - Wyświetla szczegóły wybranego tematu przed usunięciem. Użytkownik musi być autorem tematu lub adminem/moderatorem. Wykorzystuje GET.

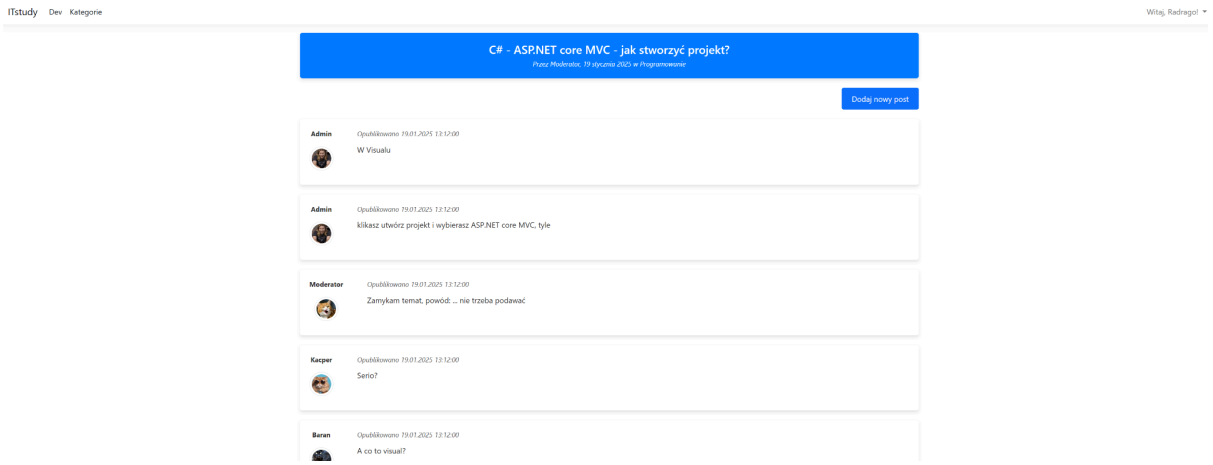
DeleteConfirmed(int id) - Usuwa temat z bazy danych na podstawie jej identyfikatora. Użytkownik musi być autorem tematu lub adminem/moderatorem. Wykorzystuje metodę POST.

```cshtml
@model Threads
@{
    ViewData["Title"] = "Usuwanie tematu";
}

<h2>Potwierdź usunięcie tematu</h2>

<p>Czy na pewno chcesz usunąć temat: <strong>@Model.Title</strong>?</p>

<form asp-action="DeleteThread" method="post">
    <input type="hidden" asp-for="Id" />
    <button type="submit" class="btn btn-danger">Usuń</button>
    <a asp-action="ShowAllThreads" asp-route-categoryId="@Model.CategoryId" class="btn btn-secondary">Anuluj</a>
</form>
```

PostsController:

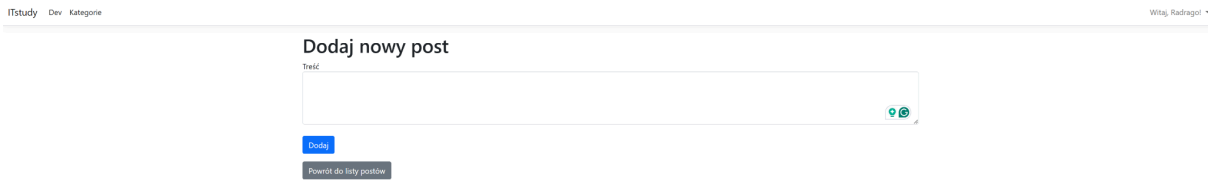
ShowAllPosts(int threadId) - Wyświetla listę tematów w danym temacie. Wykorzystuje GET.

```razor
@using System.Security.Claims
@model IEnumerable<Posts>
@{
    ViewData["Title"] = "Posty";
}

<div id="title-card">
    <h1>@ViewBag.ThreadTitle</h1>
    <p>Przez @ViewBag.UserName, @ViewBag.CreatedDate w @ViewBag.CategoryName</p>
</div>

<span class="add-btn">
    <a asp-action="AddNewPost" asp-route-threadId="@ViewBag.ThreadId" class="btn btn-primary">Dodaj nowy post</a>
</span>

<div class="posts-container">
    @foreach (var post in Model)
    {
        <div class="post-container">
            <div class="user-info">
                <p>@post.User?.UserName</p>
                <img src="@post.User?.ProfilePictureURL" alt="Zdjęcie profilowe" class="img-thumbnail align-items-center rounded-circle" style="width: 50px; height: 50px; object-fi
            </div>
            <div class="post-content">
                <div class="post-info">
                    <span>Opublikowano @post.CreatedDate @if (post.Edited) {<span>(Edytowany)</span>}</span>
                    @if (post.UserId == User.FindFirstValue(ClaimTypes.NameIdentifier) || User.IsInRole("Admin") || User.IsInRole("Moderator"))
                    {
                        <a class="post-btn" asp-controller="Posts" asp-action="EditPost" asp-route-id="@post.Id">Edytuj post</a>
                        <a class="post-btn" asp-controller="Posts" asp-action="DeletePost" asp-route-id="@post.Id">Usuń post</a>
                    }
                </div>
                <div class="post-text">
                    <p>@post.Content</p>
                </div>
            </div>
        </div>
    }
</div>
```



AddNewPost(Posts post) - Dodaje nowy post, wykorzystuje POST, autoryzacja pozwala na wykonanie tego tylko przez zalogowanego użytkownika

```razor
@model Posts
@{
    ViewData["Title"] = "Dodaj nowy post";
}

<h1>Dodaj nowy post</h1>

<form asp-action="AddNewPost" method="post">
    <div class="form-group">
        <label for="Content">Treść</label>
        <textarea id="Content" name="Content" class="form-control" rows="4" required></textarea>
    </div>
    <input type="hidden" asp-for="ThreadId" />
    <br />
    <button type="submit" class="btn btn-primary">Dodaj</button>
</form>

<a asp-action="ShowAllPosts" asp-route-threadId="@Model.ThreadId" class="btn btn-secondary mt-3">Powrót do listy postów</a>
```



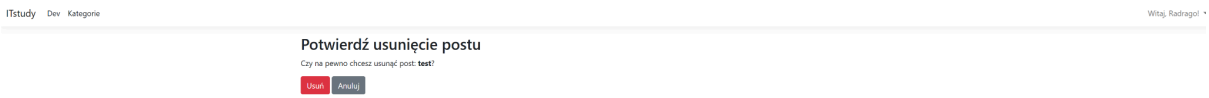
EditPost(int id, [Bind("Id,Content")]) - Edytuje wybrany post, autoryzacja pozwala na wykonanie tego tylko przez autora lub admina/moderatora. Wykorzystuje POST

```
@model Posts
@{
    ViewData["Title"] = "Edycja postu";
}

<h2 class="text-center text-info">Edycja Postu</h2>

<form method="post">
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.Id)

    <div class="form-group">
        <label for="Title">Treść posta</label>
        @Html.EditorFor(model => model.Content, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Content, "", new { @class = "text-danger" })
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Zapisz zmianę</button>
</form>
```

DeletePost(int? id) - Wyświetla szczegóły wybranego posta przed usunięciem. Użytkownik musi być autorem postu lub adminem/moderatorem. Wykorzystuje GET

DeleteConfirmed(int id) - Usuwa post z bazy danych na podstawie identyfikatora. Użytkownik musi być autorem postu lub adminem/moderatorem. Wykorzystuje POST

```
@model Posts
@{
    ViewData["Title"] = "Usuwanie postu";
}

<h2>Potwierdź usunięcie postu</h2>

<p>Czy na pewno chcesz usunąć post: <strong>@Model.Content</strong>?</p>

<form asp-action="DeletePost" method="post">
    <input type="hidden" asp-for="Id" />
    <button type="submit" class="btn btn-danger">Usuń</button>
    <a asp-action="ShowAllPosts" asp-route-threadId="@Model.ThreadId" class="btn btn-secondary">Anuluj</a>
</form>
```



AccountController:

Metody:

AccessDenied() - zwraca stronę informującą o odmowie dostępu i przyciskiem przekierowującym do storny głównej. Dostęp do niej mają wszyscy (nie zalogowani również)

Login() - Zwraca stronę z formularzem logowania. Dostęp do niej mają wszyscy (nie zalogowani również). Posiada Odpowiednik POST który przyjmuje obiekt typu LoginVM (czyli ForumUser, ale dostosowanym specjalnie do obsługi błędów logowania). Obiekt ten zawiera userName oraz hasło. Po zalogowaniu użytkownik jest przekierowywany na stronę główną

Register() - Zwraca stronę z formularzem rejestracji. Dostęp do niej mają wszyscy (nie zalogowani również). Nowy użytkownik podaje tutaj swoje imię, nazwisko, email oraz nazwę użytkownika. Posiada Odpowiednik POST który przyjmuje obiekt typu RegisterVM (czyli ForumUser, ale dostosowanym specjalnie do obsługi błędów rejestracji). Obiekt ten zawiera wylistowane wyżej rzeczy. Po zarejestrowaniu użytkownik jest przekierowywany na stronę główną

Logout() - Dostęp do niej mają wszyscy zalogowani userzy. Wylogowuje aktualnego użytkownika

ManageAccount() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę z panelem zarządzania kontem.Strona zawiera wylistowany E-mail, opis konta, zdjęcie profilowe, datę dołączenia opis konta oraz rangę/rolę. Wyświetlane są również opcje zmiany Email, hasła, zdjęcia profilowego, oraz usunięcie konta aktywnego użytkownika. Przekierowywuje do poniższych metod. Poniższe metody powracają do tej metody i przez Temp message potwierdzają zmianę danych.

ChangeAboutMe() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę wyświetlającą aktualny opis konta, oraz możliwość jej edycji. Posiada Odpowiednik POST przyjmujący obiekt typu UserWithRolesVM (czyli ForumUser połączony z AspNetRoles). Obiekt ten zawierać będzie tylko userId oraz zawartość Bio (opisu użytkownika). Po wykonaniu akcji POST przekierowywuje do ManageAccount()

ChangePassword() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę wyświetlającą Formularz z 3 polami - aktualne hasło, nowe, powtórz nowe.

Posiada Odpowiednik POST przyjmujący obiekt typu UserWithRolesVM (czyli ForumUser połączony z AspNetRoles). Po wykonaniu akcji POST wyświetla błędy w przypadku złych danych, w przypadku dobrych, przekierowywuje do ChangeAboutMe()

ChangeEmail() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę wyświetlającą aktualny email konta, oraz możliwość jego edycji. Posiada Odpowiednik POST przyjmujący obiekt typu UserWithRolesVM (czyli ForumUser połączony z AspNetRoles). Po wykonaniu akcji POST przekierowywuje do ChangeAboutMe()

ChangeProfilePicture() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę wyświetlającą aktualne zdjęcie profilowe (jeżeli istnieje), opis jak działa funkcjonalność, oraz pole tekstowe w którym można dokonać edycji linku do obrazka. Posiada Odpowiednik POST przyjmujący obiekt typu UserWithRolesVM (czyli ForumUser połączony z AspNetRoles). Po wykonaniu akcji POST przekierowywuje do ChangeAboutMe()

DeleteAccount() - Dostęp do niej mają wszyscy zalogowani userzy. Zwraca stronę wyświetlającą informacje o aktualnym użytkowniku (username, emil, zdjęcie, range) oraz przyciski do wykonania akcji post.

DeleteAccountConfirmed() - Akcja POST odpowiadająca poprzedniej metodzie - za pomocą userManagera (klasa wbudowana w identity) usuwa użytkownika, dokonuje jego wylogowania

AdminController:

Do wszystkich poniższych metod dostęp mają tylko użytkownicy z rolą “Admin” lub “Moderator”

ShowAllUsers() - Zwraca stronę z wylistowanymi wszystkimi użytkownikami forum. Wykorzystuje UserWithRolesVM oraz pomocniczą klasę PaginatedListVM (do wyświetlania paginacji - możliwości przewijania strony, wyświetlania wyselekcjonowanej ilości danych naraz). Wyświetla przy każdym użytkowniku przycisk do Moderacji konta, oraz przycisk do usunięcia konta.

EditUser() - Zwraca stronę z wszystkimi informacjami użytkownika (nie licząc hasła), czyli danymi zawartymi w UserWithRolesVM. Posiada odpowiednie przyciski do ich modyfikacji. Posiada listę z dostępnymi rolami i pozwala na zmianę roli dla danego użytkownika. Posiada Odpowiednik POST przyjmujący obiekt typu EditUserVM (czyli ForumUser połączony z AspNetRoles. Różni się ona od UserWithRolesVM brakiem niepotrzebnych pól). Po wykonaniu akcji POST przekierowywuje do ShowAllUsers()

DeleteUser() - Zwraca stronę z krótką listą informacji na temat usera, oraz przyciskiem powrót (do ShowAllUsers()) i usuń. Przycisk powrót przekierowywuje do poniższej metody

DeleteUserConfirmed() - POST. Po otrzymaniu id usera z metody powyżej, usuwa wskazanego usera i wraca do ShowAllUsers()

HomeController:

Index() - wyświetla index strony. Index potrafi odczytać wiadomości z TEMP messages (czyli reaguje na akcje logowania/rejestracji). Zawiera w sobie tabele z wszystkimi kategoriami, najnowszymi tematami oraz najpopularniejszymi tematami

Privacy() - wyświetla statyczną stronę

Regulations() - wyświetla statyczną stronę

dev() - Zwraca stronę która przydatna była w trakcie budowy projektu. Posiada większość dostępnych linków/interakcji z forum. Cała funkcjonalność znajdująca się tutaj znajduje się również w swoich prawidłowych miejscach

Error() - wbudowana metoda w ASP.NET

ModController:

ModPanel() - Dostęp tylko dla użytkowników z rolą “Admin” lub “Moderator”. Zwraca stronę z panelem moderatora/administratora

System użytkowników:

Obecnie wykorzystywane są trzy rodzaje kont: Użytkownik, Administrator i Moderator.

Rola użytkownika jest nadawana przy rejestracji. Rola Administratora i Moderatora może być nadana tylko przez innych administratorów i moderatorów w specjalnym panelu.

Użytkownicy niezalogowani mogą jedynie wyświetlać kategorie, tematy i posty.

Użytkownicy zalogowani mogą wyświetlać kategorie, tematy i posty, dodawać własne tematy i posty i edytować/usuwać własne tematy i posty. Mogą też zmieniać swoje dane użytkownika.

Administratorzy i Moderatorzy, poza uprawnieniami zwykłych użytkowników, mogą do woli dodawać/edytować/usuwać kategorie, mają dostęp do wszystkich funkcjonalności CRUD dla wszystkich użytkowników, tematów i postów.

Najciekawsze funkcjonalności:

- Paginacja - Podział danych na podstrony i Możliwość przechodzenia pomiędzy nimi, gdy w tabeli jest za dużo danych)
- Seedowanie danych - Automatyczne tworzenie bazowej wersji bazy i zapełnienie jej danymi (konta, kategorie, tematy i posty)
- Dostosowane do własnych potrzeb Identity - logowanie i rejestracja zostały przygotowane do potrzeb forum, w tym avatary użytkowników.