Programowanie Aplikacji Webowych(PAW)

Krzysztof Makówka 173664

System do obsługi filmów

1. Wstęp

- **Opis projektu:** Aplikacja webowa dla bazy danych z filmami, która pozwala kontu administratora uzupełnić bazę filmów dla wszystkich. Dla użytkownika jest przygotowana funkcja szukania filmu po tytule. Aplikacja służy do wyszukiwania filmów i najważniejszych informacji o nim.
- **Cele aplikacji:**
  - Umożliwia użytkownikom szybkie przeglądanie filmów
  - Wyszukiwanie filmów po tytule
  - Zarządzanie zbiorem filmów dla konta administratora
- Technologie:
  - Backend: C# .NET, ASP.NET
  - Frontend: Cshtml, Razor pages
  - Baza danych: MySQL

2. Opis

Modele:

1. **LoginViewModel**

```
namespace WebApplication1.Models
{

    public class LoginViewModel
    {
        [Required(ErrorMessage = "Username or Email is required ")]
        [MaxLength(20, ErrorMessage = "Max 20 characters")]
        [DisplayName("Username or Email")]

        public string UserNameOrEmail { get; set; }
        [Required(ErrorMessage = "Password is required. ")]
        [MaxLength(20, ErrorMessage = "Max 20 characters")]
        [DataType(DataType.Password)]
        0 references
        public string Password { get; set; }
    }
}
```

Model używany do obsługi logowania użytkowników.

- UserNameOrEmail: Wymagane pole dla nazwy użytkownika lub e-maila. Maksymalna długość to 20 znaków. Oznaczone atrybutem DisplayName jako "Username or Email".

- Password: Wymagane pole dla hasła. Maksymalna długość to 20 znaków. Użyto atrybutu DataType.Password, aby wskazać na poufny charakter danych.

2. RegistrationViewModel

```
public class RegistrationViewModel
{
    [Key]

    public int Id { get; set; }
    [Required(ErrorMessage = "First name has to be provided. ")]
    [MaxLength(50, ErrorMessage = "Max 50 characters")]

    public string FirstName { get; set; }
    [Required(ErrorMessage = "Last name has to be provided. ")]
    [MaxLength(50, ErrorMessage = "Max 50 characters")]

    public string LastName { get; set; }
    [Required(ErrorMessage = "Email has to be provided. ")]
    [RegularExpression(@"^([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([\w-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$", ErrorMessage = "Please Enter Valid Email.")]
    public string Email { get; set; }
    [Required(ErrorMessage = "Username is required ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    public string Username { get; set; }
    [Required(ErrorMessage = "Password is required. ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [MaxLength(5, ErrorMessage = "Max 5 characters allowed.")]
    public string Role { get; set; } = "USER";

    [Compare("Password", ErrorMessage = "Please confirm your password.")]
    [StringLength(20, MinimumLength = 5, ErrorMessage = "Max 20 characters, min 5")]
    [DataType(DataType.Password)]
    public string ConfirmPassword { get; set; }
}
```

Model używany do rejestracji nowych użytkowników.

- Id: Klucz główny.

- FirstName: Wymagane pole dla imienia. Maksymalna długość to 50 znaków.

- LastName: Wymagane pole dla nazwiska. Maksymalna długość to 50 znaków.

- Email: Wymagane pole dla adresu e-mail. Zawiera wyrażenie regularne do walidacji poprawnego formatu.

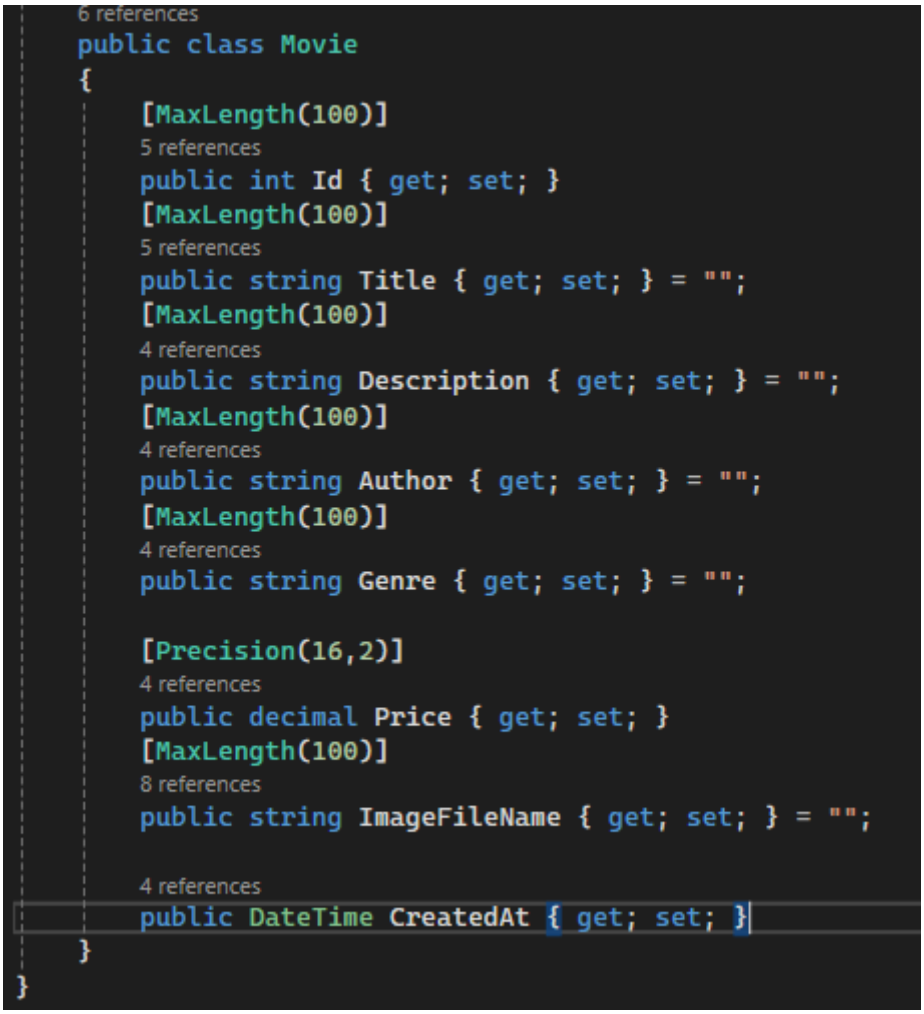
- Username: Wymagane pole dla nazwy użytkownika. Maksymalna długość to 20 znaków.

- Password: Wymagane pole dla hasła. Maksymalna długość to 20 znaków.

- Role: Opcjonalne pole dla roli użytkownika. Domyślna wartość to "USER", maksymalnie 5 znaków.

- ConfirmPassword: Pole do potwierdzenia hasła. Musi być zgodne z Password. Minimalna długość to 5 znaków, maksymalna 20.

3. Movie



```
6 references
public class Movie
{
    [MaxLength(100)]
    5 references
    public int Id { get; set; }
    [MaxLength(100)]
    5 references
    public string Title { get; set; } = "";
    [MaxLength(100)]
    4 references
    public string Description { get; set; } = "";
    [MaxLength(100)]
    4 references
    public string Author { get; set; } = "";
    [MaxLength(100)]
    4 references
    public string Genre { get; set; } = "";

    [Precision(16,2)]
    4 references
    public decimal Price { get; set; }
    [MaxLength(100)]
    8 references
    public string ImageFileName { get; set; } = "";

    4 references
    public DateTime CreatedAt { get; set; }
}
}
```

Model reprezentujący film.

- Id: Unikalny identyfikator filmu.

- Title: Tytuł filmu, maksymalna długość to 100 znaków.

- Description: Opis filmu, maksymalna długość to 100 znaków.

- Author: Autor filmu, maksymalna długość to 100 znaków.

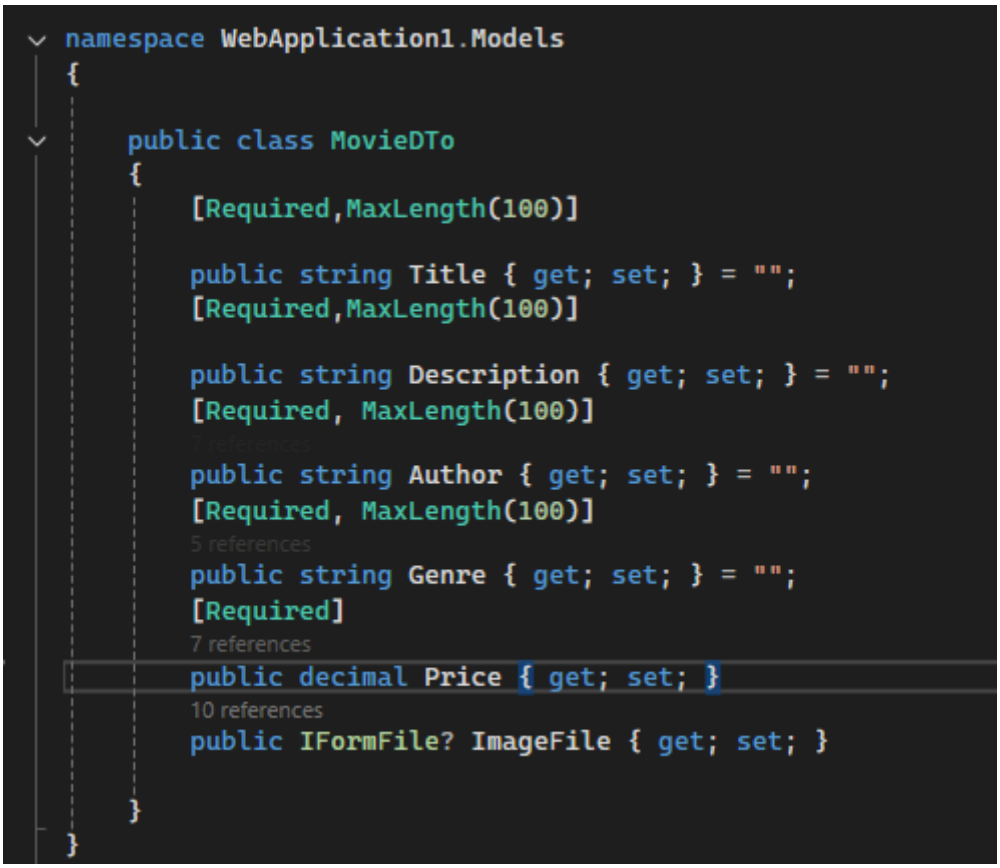
- Genre: Gatunek filmu, maksymalna długość to 100 znaków.

- Price: Cena filmu, przechowywana jako wartość dziesiętna z precyzją 16,2.

- ImageFileName: Nazwa pliku graficznego związanego z filmem, maksymalna długość to 100 znaków.

- CreatedAt: Data utworzenia wpisu o filmie.

4. MovieDTo



```
namespace WebApplication1.Models
{

    public class MovieDTo
    {
        [Required,MaxLength(100)]

        public string Title { get; set; } = "";
        [Required,MaxLength(100)]

        public string Description { get; set; } = "";
        [Required, MaxLength(100)]
        7 references
        public string Author { get; set; } = "";
        [Required, MaxLength(100)]
        5 references
        public string Genre { get; set; } = "";
        [Required]
        7 references
        public decimal Price { get; set; }
        10 references
        public IFormFile? ImageFile { get; set; }

    }
}
```

Data Transfer Object (DTO) dla filmów.

- **Title: Tytuł filmu, wymagany, maksymalnie 100 znaków.**
- **Description: Opis filmu, wymagany, maksymalnie 100 znaków.**
- **Author: Autor filmu, wymagany, maksymalnie 100 znaków.**
- **Genre: Gatunek filmu, wymagany, maksymalnie 100 znaków.**
- **Price: Cena filmu, wymagane pole.**
- **ImageFile: Opcjonalny plik graficzny (przechowywany jako IFormFile).**

5. Model: UserAccount

```
public int Id { get; set; }

[Required(ErrorMessage = "Firstname is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
5 references
public string FirstName { get; set; }

[Required(ErrorMessage = "Lastname is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
4 references
public string LastName { get; set; }

[Required(ErrorMessage = "Email is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
[DataType(DataType.EmailAddress)]
6 references
public string Email { get; set; }

[Required(ErrorMessage = "Username is required.")]
[MaxLength(20, ErrorMessage = "Max 20 characters allowed.")]
5 references
public string UserName { get; set; }

[Required(ErrorMessage = "Password is required.")]
[MaxLength(20, ErrorMessage = "Max 20 characters allowed.")]
[DataType(DataType.Password)]
2 references
public string Password { get; set; }

[Required(ErrorMessage = "USER is required.")]
[MaxLength(5, ErrorMessage = "Max 5 characters allowed.")]
1 reference
public string Role { get; set; } = "USER";
```

UserAccount to encja przeznaczona do reprezentowania danych użytkownika w bazie danych. Zawiera szczegóły takie jak imię, nazwisko, adres e-mail, nazwa użytkownika, hasło i rola użytkownika. Model używa atrybutów walidacyjnych i indeksów dla unikalności.

Wszystkie z poniższych pól posiadają walidację: takie jak, że są wymagane, czy ograniczenie dotyczące długości, oraz dodana jest unikalność.

- Id

- FirstName

- LastName

- Email

- UserName

- Password

- Role

Kontrolery:

AccountController.cs

1. **Index()**

```
0 references
public IActionResult Index()
{
    return View(_context.UserAccountsCollection.ToList());
}
```

- **Typ**: GET
- **Opis**: Zwraca listę wszystkich użytkowników zapisanych w kolekcji UserAccountsCollection i przekazuje ją do widoku.
- **Zastosowanie**: Wyświetlanie listy zarejestrowanych użytkowników.

2. **Registration()**

```
0 references
public IActionResult Registration()
{
    return View();
}
```

- **Typ**: GET
- **Opis**: Renderuje formularz rejestracji użytkownika.
- **Zastosowanie**: Wyświetlanie formularza rejestracyjnego.

3. **Registration(RegistrationViewModel model)**

```
[HttpPost]
0 references
public IActionResult Registration(RegistrationViewModel model)
{
    model.Role = "USER";
    if (ModelState.IsValid)
    {
        UserAccount account = new UserAccount();
        account.Email = model.Email;
        account.FirstName = model.FirstName;
        account.LastName = model.LastName;
        account.Password = model.Password;
        account.UserName = model.Username;

        try
        {
            _context.UserAccountsCollection.Add(account);
            _context.SaveChanges();

            ModelState.Clear();
            ViewBag.Message = $"{account.FirstName} {account.LastName} registered successfully. Please Log in.";

        }
        catch (DbUpdateException ex)
        {

            ModelState.AddModelError("", "Please, enter unique Email or Password. ");
            return View(model);
        }
        return View();


    }
    return View(model);
```

- **Typ**: POST
- **Opis**:
    - Przyjmuje dane z formularza rejestracyjnego.
    - Tworzy nowy obiekt UserAccount na podstawie danych wejściowych.
    - Waliduje dane i zapisuje nowego użytkownika w bazie danych.
    - Jeśli wystąpi błąd (np. duplikat emaila lub username), dodaje komunikat o błędzie do ModelState.
- **Zastosowanie**: Obsługa procesu rejestracji użytkownika.

4. **Login()**

```
public IActionResult Login()
{
    return View();
}
```

- **Typ**: GET
- **Opis**: Renderuje formularz logowania.
- **Zastosowanie**: Wyświetlanie formularza logowania.

5. **Login(LoginViewModel model)**

```
[HttpPost]
0 references
public IActionResult Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = _context.UserAccountsCollection.Where(x => (x.UserName == model.UserNameOrEmail || x.Email == model.UserNameOrEmail)
        && x.Password == model.Password).FirstOrDefault();
        if (user != null)
        {
            // mamy to, działa, create cookie
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Email),
                new Claim("Name", user.FirstName),
                new Claim(ClaimTypes.Role, user.Role),
            };

            var claimsIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
            HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(claimsIdentity));

            return RedirectToAction("SecurePage");
        }
        else
        {
            ModelState.AddModelError("", "Username/Email or Password isn`t correct");
        }
    }
    return View(model);
}
```

- **Typ**: POST
- **Opis**:
    - Przyjmuje dane logowania z formularza.
    - Wyszukuje użytkownika w bazie danych na podstawie nazwy użytkownika/emaila oraz hasła.
    - Jeśli dane są poprawne, tworzy **ciasteczko uwierzytelniające** zawierające informacje o użytkowniku (claims).

- Jeśli dane są niepoprawne, dodaje komunikat o błędzie do ModelState.

- **Zastosowanie**: Obsługa procesu logowania i tworzenie sesji użytkownika.

6. **LogOut()**

```
0 references
public IActionResult LogOut()
{
    HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction("Index");
}
```

- **Typ**: GET
- **Opis**:
  - Wylogowuje użytkownika poprzez usunięcie ciasteczka uwierzytelniającego.
  - Przekierowuje użytkownika na stronę główną (metoda Index).
- **Zastosowanie**: Obsługa procesu wylogowywania.

7. **SecurePage()**

```
[Authorize]
0 references
public IActionResult SecurePage()
{
    ViewBag.Name = HttpContext.User.Identity.Name;
    return View();
}
```

- **Typ**: GET
- **Opis**:
  - Strona chroniona, dostępna tylko dla zalogowanych użytkowników.
  - Wyświetla informacje o zalogowanym użytkowniku, takie jak nazwa (email).
- **Zastosowanie**: Wyświetlanie treści tylko dla użytkowników posiadających autoryzację.

MoviesController.cs

1. **Index(string searchString)**

```
0 references
public IActionResult Index(string searchString)
{
    var movies = context.MoviesCollection.ToList();

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(n => n.Title.Contains(searchString)).ToList();
    }
    return View(movies);
}
```

- **Typ**: GET

- **Opis**: Pobiera listę filmów z bazy danych. Filtruje wyniki na podstawie przekazanej frazy searchString (szuka w tytule filmu).
- **Zastosowanie**: Wyświetlanie listy filmów z opcjonalnym filtrowaniem.

2. **Create()**

```
public IActionResult Create()
{
    return View();
}
```

- **Typ**: GET
- **Opis**: Renderuje formularz do dodawania nowego filmu.
- **Zastosowanie**: Wyświetlanie formularza tworzenia filmu.

3. **Create(MovieDTo movieDTo)**

```
0 references
public IActionResult Create(MovieDTo movieDTo)
{
    if (movieDTo.ImageFile == null)
    {
        ModelState.AddModelError("ImageFile", "The image is required");
    }
    if (!ModelState.IsValid) {
        return View(movieDTo);
    }

    string newFileName = DateTime.Now.ToString("yyyyMMddHHmmssfff");
    newFileName += Path.GetExtension(movieDTo.ImageFile!.FileName);

    string imageFullPath = environment.WebRootPath + "/photos/"  + newFileName;
    using (var stream = System.IO.File.Create(imageFullPath))
    {
        movieDTo.ImageFile.CopyTo(stream);
    }

    Movie movie = new Movie()
    {
        Title = movieDTo.Title,
        Description = movieDTo.Description,
        Author = movieDTo.Author,
        Genre = movieDTo.Genre,
        Price = movieDTo.Price,
        ImageFileName = newFileName,
        CreatedAt = DateTime.Now,
    };
    context.MoviesCollection.Add(movie);
    context.SaveChanges();


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

- **Typ**: POST
- **Opis**:
    - Waliduje dane wejściowe.

- Zapisuje obraz filmu na serwerze.
- Tworzy nowy obiekt Movie, zapisuje go w bazie danych.

- **Zastosowanie**: Obsługa dodawania nowego filmu.

4. **Edit(int id)**

```
0 references
public IActionResult Edit(int id)
{
    var movie = context.MoviesCollection.Find(id);
    if (movie == null)
    {
        return RedirectToAction("Index", "movies");
    }

    var movieDTo = new MovieDTo()
    {
        Title = movie.Title,
        Description = movie.Description,
        Author = movie.Author,
        Genre = movie.Genre,
        Price = movie.Price,
    };


    ViewData["id"] = movie.Id;
    ViewData["ImageFileName"] = movie.ImageFileName;
    ViewData["CreatedAt"] = movie.CreatedAt.ToString("MM/dd/yyyy");
    return View(movieDTo);
}
```

- **Typ**: GET
- **Opis**:
  - Pobiera film o podanym id.
  - Jeśli film istnieje, przygotowuje dane do edycji i renderuje formularz.
- **Zastosowanie**: Wyświetlanie formularza edycji filmu.

5. **Edit(int id, MovieDTo movieDTo)**

```
[HttpPost]
0 references
public IActionResult Edit(int id, MovieDTo movieDTo)
{
    var movie = context.MoviesCollection.Find(id);
    if(movie == null)
    {
        return RedirectToAction("Index", "movies");
    }

    if(!ModelState.IsValid)
    {
        ViewData["id"] = movie.Id;
        ViewData["ImageFileName"] = movie.ImageFileName;
        ViewData["CreatedAt"] = movie.CreatedAt.ToString("MM/dd/yyyy");

        return View(movieDTo);
    }

    string newFileName = movie.ImageFileName;
    if (movieDTo.ImageFile != null)
    {
        newFileName = DateTime.Now.ToString("yyyyMMddHHmmssfff");
        newFileName += Path.GetExtension(movieDTo.ImageFile.FileName);

        string imageFullPath = environment.WebRootPath + "/photos/" + newFileName;
        using (var stream = System.IO.File.Create(imageFullPath))
        {
            movieDTo.ImageFile.CopyTo(stream);
        }

        // delete the old image
        string oldImageFullPath = environment.WebRootPath + "/photos/" + movie.ImageFileName;
        System.IO.File.Delete(oldImageFullPath);
    }
    }

    movie.Title = movieDTo.Title;
    movie.Description = movieDTo.Description;
    movie.Author = movieDTo.Author;
    movie.Genre = movieDTo.Genre;
    movie.Price = movieDTo.Price;
    movie.ImageFileName = newFileName;



    context.SaveChanges();
    return RedirectToAction("Index", "movies");
}
```

- **Typ**: POST
- **Opis**:
    - Aktualizuje dane filmu o podanym id na podstawie przesłanych danych.
    - Jeśli dodano nowy obraz, zastępuje stary (usuwa poprzedni plik z serwera).
- **Zastosowanie**: Obsługa edycji filmu.

6. **Delete(int id)**

```
public IActionResult Delete(int id)
{
    var movie = context.MoviesCollection.Find(id);
    if(movie == null)
    {
        return RedirectToAction("Index", "movies");
    }

    string imageFullPath = environment.WebRootPath + "/photos/" + movie.ImageFileName;
    System.IO.File.Delete(imageFullPath);

    context.MoviesCollection.Remove(movie);
    context.SaveChanges(true);

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

- **Typ**: GET
- **Opis**:
    - Usuwa film o podanym id z bazy danych.
    - Usuwa również powiązany plik obrazu z serwera.
- **Zastosowanie**: Obsługa usuwania filmu.

HomeController

1. **Index()**

```
public IActionResult Index()
{
    return View();
}
```

- **Typ**: GET
- **Opis**: Renderuje stronę główną aplikacji.
- **Zastosowanie**: Wyświetlanie głównej strony.

Widoki

1. **Index (dla kolekcji UserAccount)**

```
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Id)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.LastName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Email)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.UserName)
            </th>
        </tr>
    </thead>
    <tbody>
```

```
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Id)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FirstName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.UserName)
            </td>
        </tr>
}
    </tbody>
</table>
```

Cel widoku:

- Wyświetlenie listy użytkowników (UserAccount) w formie tabeli.

Elementy widoku:

- **Nagłówek**: Tytuł "Index".
- **Przycisk "Create New"**: Link do akcji Create, pozwalający na dodanie nowego użytkownika.
- **Tabela**:
    - Wyświetla dane użytkowników (Id, FirstName, LastName, Email, UserName).
    - Dane są generowane dynamicznie w pętli foreach, iterującej po modelu (IEnumerable<UserAccount>).

2. **Login**

```
<h1>Login</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Login">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="UserNameOrEmail" class="control-label"></label>
                <input asp-for="UserNameOrEmail" class="form-control" />
                <span asp-validation-for="UserNameOrEmail" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Login" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    Don`t have an account? <a asp-action="Registration">Go to Registration</a>
</div>

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

Cel widoku:

- Formularz logowania użytkownika.

Elementy widoku:

- **Nagłówek**: Tytuł "Login".
- **Formularz logowania**:
    - **Pola**:

        - UserNameOrEmail (nazwa użytkownika lub e-mail).
        - Password (hasło).
    - **Przycisk "Login"**: Przesyła dane formularza do akcji Login.
    - **Walidacja**: Wyświetla komunikaty o błędach, jeśli walidacja modelu nie powiedzie się.
- **Link do rejestracji**: Przekierowanie do akcji Registration, jeśli użytkownik nie ma konta.

3. **Registration**

```
<h1>Registration</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Registration">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            @if (ViewBag.Message != null)
            {
                <div class="form-group" style="color: green;">@ViewBag.Message</div>
            }
            <div class="form-group">
                <label asp-for="FirstName" class="control-label"></label>
                <input asp-for="FirstName" class="form-control" />
                <span asp-validation-for="FirstName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="LastName" class="control-label"></label>
                <input asp-for="LastName" class="form-control" />
                <span asp-validation-for="LastName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
```

```
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ConfirmPassword" class="control-label"></label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Register" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    Already have an account? <a asp-action="Login">Go to Login</a>
</div>

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

Cel widoku:

- Formularz rejestracji nowego użytkownika.

Elementy widoku:

- **Nagłówek**: Tytuł "Registration".
- **Formularz rejestracyjny**:
    - **Pola**:
        - FirstName, LastName, Email, Username, Password, ConfirmPassword.
    - **Przycisk "Register"**: Przesyła dane formularza do akcji Registration.
    - **Komunikaty**:
        - Wyświetla wiadomość z ViewBag.Message, jeśli rejestracja zakończy się sukcesem.
        - Pokazuje błędy walidacji, jeśli dane są nieprawidłowe.
- **Link do logowania**: Przekierowanie do akcji Login, jeśli użytkownik ma już konto.

4. SecurePage Widok powitalny (Hello @ViewBag.Name)

Cel widoku:

- Wyświetlenie powitania z nazwą użytkownika (@ViewBag.Name) na stronie, dostępnej po zalogowaniu.

Elementy widoku:

- Nagłówek z dynamicznie generowanym powitaniem (np. "Hello Krzysztof").

5. New Movie (Dodawanie filmu)

Cel widoku:

- Umożliwienie użytkownikowi dodania nowego filmu do systemu.

Elementy widoku:

- **Nagłówek**: Tytuł "New Movie".
- **Formularz**:
    - Pola do wprowadzania danych:
        - Title (tytuł filmu).
        - Description (opis filmu).
        - Genre (gatunek filmu - wybierany z rozwijanej listy).
        - Price (cena filmu).
        - Author (autor filmu).
        - ImageFile (zdjęcie wgrywane jako plik).
    - Walidacja: Wyświetlane są komunikaty o błędach przy nieprawidłowych danych.
    - Przycisk "Submit": Zapisuje dane filmu.
    - Przycisk "Cancel": Przekierowuje użytkownika z powrotem do listy filmów.

6. Edit Movie (Edycja filmu)

Cel widoku:

- Umożliwienie użytkownikowi edytowania istniejących danych filmu.

Elementy widoku:

- **Nagłówek**: Tytuł "Edit Movie".
- **Formularz**:
    - Pola do edycji danych:
        - Title, Description, Genre, Price, Author.
    - Wyświetlane tylko do odczytu:

        - Id (identyfikator filmu).
        - Created At (data utworzenia).
    - Sekcja podglądu:
        - Obraz filmu (ImageFileName).
    - Przycisk "Submit": Zapisuje zmiany.
    - Przycisk "Cancel": Powrót do listy filmów.

7. List of Movies (Lista filmów)

Cel widoku:

- Wyświetlenie wszystkich dostępnych filmów w systemie oraz udostępnienie opcji zarządzania (dla administratora).

Elementy widoku:

- **Nagłówek**: Tytuł "List of Movies".
- **Sekcja wyszukiwania**:
    - Pole wyszukiwania po nazwie filmu.
    - Link do powrotu do pełnej listy.
- **Przycisk "New Movie"** (dla administratorów): Przekierowuje do formularza dodawania nowego filmu.
- **Tabela z danymi**:
    - Kolumny:
        - Id, Title, Description, Author, Genre, Price, Created At, Action.
    - Wiersze:
        - Dynamiczne wyświetlanie danych o filmach z modelu.
        - Kolumna "Action" (dla administratorów):
            - Przycisk "Edit": Przekierowanie do edycji filmu.
            - Przycisk "Delete": Usunięcie filmu.
- **Podgląd zdjęć**:
    - Obrazy są wyświetlane w tabeli w kolumnie ImageFileName.

3. Link do repozytorium:

https://github.com/makovvka/MoviesRental

4. Zadanie dodatkowe - Opis wzorca projektowego MVC

Struktura wzorca projektowego MVC – jest to organizacja struktury aplikacji, która zachowuje przejrzystość kodu i łatwość zarządzania nim. Składa się z trzech komponentów: „Model” – to miejsce gdzie znajdują się dane i logika aplikacji. „View” – jest to miejsce, w którym pokazane jest co użytkownik widzi i z czym bezpośrednio ma styczność. „Controller” – jest to miejsce w którym łączy się model i widok, miejsce te można nazwać kluczowym. Kontroler służy do odbioru żądania od użytkownika, które przetwarzane jest za pomocą modelu, a następnie zwraca odpowiedni widok.

- Model jest sercem architektury MVC i odpowiada za logikę aplikacji oraz zarządzanie danymi. Jest odpowiedzialny za przechowywanie, modyfikowanie i udostępnianie danych, które są później prezentowane użytkownikowi.
- Widok (View) jest odpowiedzialny za prezentację danych i interfejs użytkownika. To warstwa, która odpowiada za generowanie wyświetlania informacji w sposób zrozumiały i przyjazny dla użytkownika końcowego.
- Kontroler (Controller) działa jako komunikator pomiędzy Modelem a Widokiem. Kontroler jest odpowiedzialny za odbieranie wejścia od użytkownika (np. poprzez formularze, kliknięcia czy zapytania HTTP), przetwarzanie tych danych oraz zarządzanie interakcjami między Modelem a Widokiem.

Pokazanie MVC w praktyce na podstawie wyżej opisanego projektu:

```
[Key]
2 references
public int Id { get; set; }

[Required(ErrorMessage = "Firstname is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
5 references
public string FirstName { get; set; }

[Required(ErrorMessage = "Lastname is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
4 references
public string LastName { get; set; }

[Required(ErrorMessage = "Email is required.")]
[MaxLength(50, ErrorMessage = "Max 50 characters allowed.")]
[DataType(DataType.EmailAddress)]
6 references
public string Email { get; set; }

[Required(ErrorMessage = "Username is required.")]
[MaxLength(20, ErrorMessage = "Max 20 characters allowed.")]
5 references
public string UserName { get; set; }
```

```
    [Required(ErrorMessage = "Password is required.")]
    [MaxLength(20, ErrorMessage = "Max 20 characters allowed.")]
    [DataType(DataType.Password)]
    2 references
    public string Password { get; set; }

    [Required(ErrorMessage = "USER is required.")]
    [MaxLength(5, ErrorMessage = "Max 5 characters allowed.")]
    1 reference
    public string Role { get; set; } = "USER";
```

```
public class RegistrationViewModel
{
    [Key]
    0 references
    public int Id { get; set; }
    [Required(ErrorMessage = "First name has to be provided. ")]
    [MaxLength(50, ErrorMessage = "Max 50 characters")]
    4 references
    public string FirstName { get; set; }
    [Required(ErrorMessage = "Last name has to be provided. ")]
    [MaxLength(50, ErrorMessage = "Max 50 characters")]
    4 references
    public string LastName { get; set; }
    [Required(ErrorMessage = "Email has to be provided. ")]
    [RegularExpression(@"^([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([\w-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$", ErrorMessage = "Please Enter Valid Email.")]
    4 references
    public string Email { get; set; }
    [Required(ErrorMessage = "Username is required ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    4 references
    public string Username { get; set; }
    [Required(ErrorMessage = "Password is required. ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    [DataType(DataType.Password)]
    4 references
    public string Password { get; set; }

    [MaxLength(5, ErrorMessage = "Max 5 characters allowed.")]
    1 reference
    public string Role { get; set; } = "USER";

    [Compare("Password", ErrorMessage = "Please confirm your password.")]
    [StringLength(20, MinimumLength = 5, ErrorMessage = "Max 20 characters, min 5")]
    [DataType(DataType.Password)]
    3 references
    public string ConfirmPassword { get; set; }
}
```

```
4 references
public class LoginViewModel
{
    [Required(ErrorMessage = "Username or Email is required ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    [DisplayName("Username or Email")]
    5 references
    public string UserNameOrEmail { get; set; }
    [Required(ErrorMessage = "Password is required. ")]
    [MaxLength(20, ErrorMessage = "Max 20 characters")]
    [DataType(DataType.Password)]
    4 references
    public string Password { get; set; }
}
```

Wyżej pokazane zdjęcia są Modelem klasy UserAccount, LoginViewModel i RegistrationViewModel. Zadeklarowane są tutaj dane, które są prezentowane użytkownikowi i też te, które deklarują jego prawa. Nie wszystkie z nich są pokazywane użytkownikowi, lecz są potrzebne w poprawnym funkcjonowaniu całej aplikacji. Jak można zauważyć, w modelu również są użyte pola walidacji, przykładowo użytkownik nie może założyć konta bez hasła, czy też to hasło musi być krótsze niż 20 znaków. Modele do rejestracji i logowania następnie będą wyświetlone w poniższych widokach, a ich logika zaimplementowana w kontrolerach.

```
@{
    ViewData["Title"] = "Registration";
}

<h1>Registration</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Registration">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            @if (ViewBag.Message != null)
            {
                <div class="form-group" style="color: green;">@ViewBag.Message</div>
            }
            <div class="form-group">
                <label asp-for="FirstName" class="control-label"></label>
                <input asp-for="FirstName" class="form-control" />
                <span asp-validation-for="FirstName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="LastName" class="control-label"></label>
                <input asp-for="LastName" class="form-control" />
                <span asp-validation-for="LastName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
```

```
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ConfirmPassword" class="control-label"></label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Register" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    Already have an account? <a asp-action="Login">Go to Login</a>
</div>

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

Przedstawiony jest tutaj widok Rejestracji, wyświetlane są tutaj pola wymagane do rejestracji, wykorzystywane one są właśnie z modelu. Kiedy użytkownik podaje dane, również działa wtedy walidacja wprowadzona wcześniej w modelu. W Widoku można również dodać różne pola, przykładowo użyte na dole przekierowanie do zalogowania, gdy użytkownik posiada już swoje konto.

```
<h1>Login</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Login">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="UserNameOrEmail" class="control-label"></label>
                <input asp-for="UserNameOrEmail" class="form-control" />
                <span asp-validation-for="UserNameOrEmail" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Login" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    Don`t have an account? <a asp-action="Registration">Go to Registration</a>
</div>

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

W widoku logowania generujemy także pola do wypełnienia, podajemy login lub email podany w procesie rejestracji i hasło. Walidacje i pola również pobieramy z modelu, stąd użyteczność wzorca MVC. Na dole widoku również widnieje możliwość przeniesienia do formularza rejestracji.

```
[HttpPost]
0 references
public IActionResult Registration(RegistrationViewModel model)
{
    model.Role = "USER";
    if (ModelState.IsValid)
    {
        UserAccount account = new UserAccount();
        account.Email = model.Email;
        account.FirstName = model.FirstName;
        account.LastName = model.LastName;
        account.Password = model.Password;
        account.UserName = model.Username;

        try
        {
            _context.UserAccountsCollection.Add(account);
            _context.SaveChanges();

            ModelState.Clear();
            ViewBag.Message = $"{account.FirstName} {account.LastName} registered successfully. Please Log in.";

        }
        catch (DbUpdateException ex)
        {

            ModelState.AddModelError("", "Please, enter unique Email or Password. ");
            return View(model);
        }
        return View();


    }
    return View(model);
```

W kontrolerze obsługującym rejestracje używamy pól z modelu, pomaga nam to w utrzymaniu czytelnego kodu i prostoty. Sam kontroler pozwala nam na podstawie tych pól założyć konto przy podaniu wymaganych danych. W momencie gdy rejestracja przebiegnie prawidłowo, konto i jego dane są dodawane do bazy danych, wyświetlany jest również odpowiedni komunikat z danymi konta.

```
[HttpPost]
0 references
public IActionResult Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = _context.UserAccountsCollection.Where(x => (x.UserName == model.UserNameOrEmail || x.Email == model.UserNameOrEmail)
        && x.Password == model.Password).FirstOrDefault();
        if (user != null)
        {
            // mamy to, działa, create cookie
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Email),
                new Claim("Name", user.FirstName),
                new Claim(ClaimTypes.Role, user.Role),
            };

            var claimsIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
            HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(claimsIdentity));

            return RedirectToAction("SecurePage");
        }
        else
        {
            ModelState.AddModelError("", "Username/Email or Password isn`t correct");
        }
    }
    return View(model);
}
```

Kolejną akcja w kontrolerze jest logowanie, przyjęcie danych z formularza, wyszukanie danych – loginu/e-mailu i hasła w bazie i jeśli podane dane są prawidłowe tworzone jest ciasteczko uwierzytelniające, potocznie zwany Claim, który zawiera dane o użytkowniku.

Powyższy punkt pokazał sposobność działania wzorca MVC, który jest bardzo przyjemny w użytkowaniu, oferuję łatwość przy przeglądzie kodu. Ponadto, przy zadeklarowaniu modeli związanych z logowaniem i rejestracją, trzymanie się konwencji pozwoliło na bezproblemowe połączenie Modelu-Widoku-Controllera, co skutkuje bezproblemowymi działaniem aplikacji.