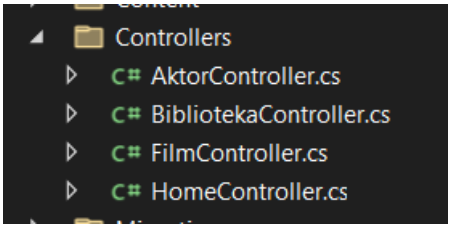
Dokumentacja

Justyna Kożuch, Anna Osmala

Celem projektu było stworzenie aplikacji do zbierania danych o filmach, aktorach i tworzeniu biblioteki na ich podstawie.

Opis kontrolerów:



Do poprawnego działania aplikacji potrzebne było stworzenie 4 kontrolerów. HomeController.cs został wygenerowany automatycznie, a 3 pozostałe zostały dodane, by aplikacja mogła funkcjonować.

AktorController.cs

```
0 references
public class AktorController : Controller
{
    public DatabaseContext db = new DatabaseContext();

    [HttpGet]
    0 references
    public ActionResult Create()
    {

        return View(new Aktor());
    }

    [HttpPost]
    0 references
    public ActionResult Create(Aktor aktor)
    {
        if (ModelState.IsValid)
        {
            using(DatabaseContext db = new DatabaseContext())
            {
                db.Aktorzy.Add(aktor);
                db.SaveChanges();
            }
            return RedirectToAction("ViewAll");
        }
        return View(new Aktor());
    }
```

Metoda **public ActionResult Create()** to metoda kontrolera, która obsługuje żądanie HTTP GET. Zwraca widok o nazwie "Create" z modelem **Aktor.**

Metoda **public ActionResult Create(Aktor aktor)** to metoda kontrolera, która obsługuje żądanie HTTP POST. Przyjmuje parametr **Aktor aktor**, który jest modelem aktora przekazanym z formularza w widoku "Create". Ta metoda dodaje przekazanego aktora do kontekstu bazy danych (**db.Aktorzy.Add(aktor)**) i zapisuje zmiany w bazie danych (**db.SaveChanges()**). Następnie przekierowuje do akcji "ViewAll". Po zapisaniu aktora w bazie danych, przekierowuje do akcji o nazwie "ViewAll", która jest odpowiedzialna za wyświetlanie wszystkich aktorów.

```
[HttpGet]
0 references
public ActionResult ViewAll()
{
    List<Aktor> aktorzy;
    using (DatabaseContext db = new DatabaseContext())
        aktorzy = db.Aktorzy.ToList();
    return View(aktorzy);
}

0 references
public ActionResult View(int id)
{
    Aktor aktor;
    using (DatabaseContext db = new DatabaseContext())
        aktor = db.Aktorzy.FirstOrDefault(x => x.AktorId == id);

    return View(aktor);
}
```

Metoda **public ActionResult ViewAll()** to metoda kontrolera, która obsługuje żądanie HTTP GET. Pobiera listę wszystkich aktorów z bazy danych poprzez utworzenie nowej instancji klasy **DatabaseContext** i wywołanie metody **ToList()** na właściwości **Aktorzy** kontekstu. Następnie zwraca widok o nazwie "ViewAll" z modelem zawierającym listę aktorów.

Metoda **public ActionResult View(int id)** to metoda kontrolera, która obsługuje żądanie HTTP GET z parametrem **id**. Pobiera aktora o określonym identyfikatorze z bazy danych poprzez utworzenie nowej instancji klasy **DatabaseContext** i wywołanie metody **FirstOrDefault()** na właściwości **Aktorzy** kontekstu. Metoda **FirstOrDefault()** zwraca pierwszego aktora spełniającego podany warunek, gdzie sprawdza, czy **AktorId** aktora jest równy **id**. Następnie zwraca widok o nazwie "View" z modelem zawierającym wybranego aktora.

```
[HttpGet]
0 references
public ActionResult Edit(int id)
{
    Aktor aktor;
    using (DatabaseContext db = new DatabaseContext())
        aktor = db.Aktorzy.FirstOrDefault(x => x.AktorId == id);

    return View(aktor);
}

[HttpPost]
0 references
public ActionResult Edit(Aktor aktor)
{
    if (!ModelState.IsValid)
        return View(aktor);

    using (DatabaseContext db = new DatabaseContext())
    {
        db.Entry(aktor).State = EntityState.Modified;
        db.SaveChanges();
    }
    return RedirectToAction("ViewAll");
}
```

Metoda **public ActionResult Edit(int id)** Jest to metoda kontrolera, która obsługuje żądanie HTTP GET z parametrem **id**. Pobiera aktora o określonym identyfikatorze z bazy danych poprzez utworzenie nowej instancji klasy **DatabaseContext** i wywołanie metody **FirstOrDefault()** na właściwości **Aktorzy** kontekstu. Metoda **FirstOrDefault()** zwraca pierwszego aktora spełniającego podany warunek, gdzie sprawdza, czy **AktorId** aktora jest równy **id**. Następnie zwraca widok o nazwie "Edit" z modelem zawierającym wybranego aktora.

Metoda **public ActionResult Edit(Aktor aktor)** to metoda kontrolera, która obsługuje żądanie HTTP POST. Przyjmuje parametr **Aktor aktor**, który jest modelem aktora przekazanym z formularza w widoku "Edit". Sprawdza, czy stan modelu (**ModelState**) jest prawidłowy. Jeśli dane nie są prawidłowe, zwraca widok "Edit" z modelem aktora, aby użytkownik mógł poprawić dane. Następnie tworzy nową instancję klasy **DatabaseContext**, aby uzyskać dostęp do bazy danych. Ustawia stan obiektu **aktor** na **Modified**, co oznacza, że obiekt ten został zmodyfikowany i należy zaktualizować odpowiednią pozycję w bazie danych i zapisuje zmiany. Po zapisaniu zmian, przekierowuje do akcji "ViewAll", która wyświetla wszystkich aktorów.

```
[HttpGet]
0 references
public ActionResult Delete(int? id)
{
    Aktor aktor;
    using (DatabaseContext db = new DatabaseContext())
    {
        aktor = db.Aktorzy.FirstOrDefault(x => x.AktorId == id);
    }


    return View(aktor);
}

[HttpPost, ActionName("Delete")]
0 references
public ActionResult DeleteConfirm(int? id)
{
    Aktor aktor;
    using (DatabaseContext db = new DatabaseContext())
    {
        aktor = db.Aktorzy.FirstOrDefault(x => x.AktorId == id);
        db.Aktorzy.Remove(aktor);
        db.SaveChanges();
    }

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

Metoda **public ActionResult Delete(int? id)** pobiera aktora o określonym identyfikatorze z bazy danych poprzez utworzenie nowej instancji klasy **DatabaseContext** i wywołanie metody **FirstOrDefault()** na właściwości **Aktorzy** kontekstu. Metoda **FirstOrDefault()** zwraca pierwszego aktora spełniającego podany warunek, gdzie sprawdza, czy **AktorId** aktora jest równy **id**. Następnie zwraca widok o nazwie "Delete" z modelem zawierającym wybranego aktora.

Metoda **public ActionResult DeleteConfirm(int? id)** pobiera aktora o określonym identyfikatorze z bazy danych poprzez utworzenie nowej instancji klasy **DatabaseContext** i wywołanie metody **FirstOrDefault()** na właściwości **Aktorzy** kontekstu. Metoda **FirstOrDefault()** zwraca pierwszego aktora spełniającego podany warunek, gdzie sprawdza, czy **AktorId** aktora jest równy **id**. Następnie usuwa wybranego aktora z bazy danych poprzez wywołanie metody **Remove(aktor)** i zapisuje zmiany w bazie danych poprzez wywołanie **db.SaveChanges()**. Na koniec przekierowuje do akcji "ViewAll", która wyświetla wszystkich aktorów.

BibliotekaController.cs

```
private DatabaseContext db = new DatabaseContext();

// GET: Biblioteka
0 references
public ActionResult Index()
{
    return View(db.Biblioteki.ToList());
}

// GET: Biblioteka/Details/5
0 references
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Biblioteka biblioteka = db.Biblioteki.Find(id);
    if (biblioteka == null)
    {
        return HttpNotFound();
    }
    return View(biblioteka);
}

// GET: Biblioteka/Create
0 references
public ActionResult Create()
{
    return View();
}
```

Metoda **public ActionResult Index()** wraca widok o nazwie "Index" z listą wszystkich obiektów **Biblioteka** pobranych z bazy danych przy użyciu **db.Biblioteki.ToList()**.

Metoda **public ActionResult Details(int? id)** sprawdza, czy **id** jest null. Następnie pobiera obiekt **Biblioteka** o określonym **id** z bazy danych za pomocą **db.Biblioteki.Find(id)**. Jeśli obiekt **biblioteka** nie jest null zwraca widok "Details" z modelem **biblioteka**.

Metoda **public ActionResult Create()** zwraca widok "Create", który jest formularzem do tworzenia nowego obiektu **Biblioteka**.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Create([Bind(Include = "BibliotekaId,Nazwa")] Biblioteka biblioteka)
{
    if (ModelState.IsValid)
    {
        db.Biblioteki.Add(biblioteka);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(biblioteka);
}

// GET: Biblioteka/Edit/5
0 references
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Biblioteka biblioteka = db.Biblioteki.Find(id);
    if (biblioteka == null)
    {
        return HttpNotFound();
    }
    return View(biblioteka);
}
```

Metoda **public ActionResult Create([Bind(Include = "BibliotekaId,Nazwa")] Biblioteka biblioteka)** przyjmuje parametr **Biblioteka biblioteka**, który jest modelem biblioteki przekazanym z formularza w widoku "Create". Jeśli stan modelu (**ModelState**) jest prawidłowy, dodaje nową bibliotekę do bazy danych poprzez wywołanie **db.Biblioteki.Add(biblioteka)** i zapisuje zmiany w bazie danych za pomocą **db.SaveChanges()**. Następnie przekierowuje do akcji "Index", która wyświetla listę bibliotek. Jeśli dane modelu nie są prawidłowe, zwraca widok "Create" z modelem biblioteki, aby użytkownik mógł poprawić dane.

Metoda **public ActionResult Edit(int? id)** sprawdza, czy **id** jest null. Następnie pobiera obiekt **Biblioteka** o określonym **id** z bazy danych za pomocą **db.Biblioteki.Find(id)**. Jeśli obiekt **biblioteka** nie jest null zwraca widok "Edit" z modelem **biblioteka**.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Edit([Bind(Include = "BibliotekaId,Nazwa")] Biblioteka biblioteka)
{
    if (ModelState.IsValid)
    {
        db.Entry(biblioteka).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(biblioteka);
}

// GET: Biblioteka/Delete/5
0 references
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Biblioteka biblioteka = db.Biblioteki.Find(id);
    if (biblioteka == null)
    {
        return HttpNotFound();
    }
    return View(biblioteka);
}
```

Metoda **public ActionResult Edit([Bind(Include = "BibliotekaId,Nazwa")] Biblioteka biblioteka)** przyjmuje parametr **Biblioteka biblioteka**, który jest modelem biblioteki przekazanym z formularza w widoku "Edit". Jeśli stan modelu (**ModelState**) jest prawidłowy, ustawia stan obiektu **biblioteka** na **EntityState.Modified**, co oznacza, że obiekt jest zmodyfikowany w kontekście bazy danych. Następnie zapisuje zmiany w bazie danych za pomocą **db.SaveChanges()**. Na koniec przekierowuje do akcji "Index", która wyświetla listę bibliotek. Jeśli dane modelu nie są prawidłowe, zwraca widok "Edit" z modelem biblioteki, aby użytkownik mógł poprawić dane.

Metoda **public ActionResult Delete(int? id)** sprawdza, czy **id** jest null. Następnie pobiera obiekt **Biblioteka** o określonym **id** z bazy danych za pomocą **db.Biblioteki.Find(id)**. Jeśli obiekt **biblioteka** nie jest null zwraca widok "Delete" z modelem **biblioteka**.

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult DeleteConfirmed(int id)
{
    Biblioteka biblioteka = db.Biblioteki.Find(id);
    db.Biblioteki.Remove(biblioteka);
    db.SaveChanges();
    return RedirectToAction("Index");
}

0 references
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
```

Metoda **public ActionResult DeleteConfirmed(int id)** przyjmuje parametr **int id**, który jest identyfikatorem biblioteki do usunięcia. Pobiera obiekt **Biblioteka** o określonym **id** z bazy danych za pomocą **db.Biblioteki.Find(id)**. Następnie usuwa ten obiekt z kontekstu bazy danych za pomocą **db.Biblioteki.Remove(biblioteka)**. Po zapisaniu zmian w bazie danych za pomocą **db.SaveChanges()**, przekierowuje do akcji "Index", która wyświetla listę bibliotek.

Metoda **protected override void Dispose(bool disposing)** to metoda, która zastępuje metodę **Dispose** z klasy bazowej. W przypadku, gdy **disposing** ma wartość **true** wywołuje **db.Dispose()** w celu zwolnienia zasobów kontekstu bazy danych. Następnie wywołuje metodę bazową **base.Dispose(disposing)** w celu wykonania dodatkowych czynności zwalniania zasobów.

FilmController.cs

Metoda **public ActionResult Index()** pobiera listę filmów z bazy danych za pomocą **db.Filmy.Include(f => f.Aktor).Include(f => f.Biblioteka)**. Następnie lista filmów jest przekazywana do widoku "Index" za pomocą **View(filmy.ToList())**.

Metoda **public ActionResult Details(int? id)** sprawdza, czy **id** jest null. Następnie pobiera obiekt **Film** o określonym **id** z bazy danych za pomocą **db.Filmy.Find(id)**. Jeśli obiekt **film** nie jest null zwraca widok "Details" z modelem **film**.

```
public class FilmController : Controller
{
    private DatabaseContext db = new DatabaseContext();

    // GET: Film
    0 references
    public ActionResult Index()
    {
        var filmy = db.Filmy.Include(f => f.Aktor).Include(f => f.Biblioteka);
        return View(filmy.ToList());
    }

    // GET: Film/Details/5
    0 references
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Film film = db.Filmy.Find(id);
        if (film == null)
        {
            return HttpNotFound();
        }
        return View(film);
    }
```

Metoda **public ActionResult Create()** tworzy listę aktorów posortowaną alfabetycznie i przypisuje ją do **ViewBag.Aktorzy** w celu przekazania jej do widoku. Następnie tworzy **ViewBag.AktorId** i **ViewBag.BibliotekaId**, które zawierają listy wyboru (**SelectList**) aktorów i bibliotek. Obie listy są używane w widoku "Create" do wyboru aktora i biblioteki dla nowego filmu. Na koniec zwraca widok "Create".

Metoda **public ActionResult Create([Bind(Include = "FilmId,Tytul,Gatunek,AktorId,BibliotekaId")] Film film)** przyjmuje parametr **Film film**, który jest modelem filmu przekazanym z formularza w widoku "Create". Jeśli stan modelu (**ModelState**) jest prawidłowy, dodaje ten obiekt **film** do kontekstu bazy danych za pomocą **db.Filmy.Add(film)**. Następnie zapisuje zmiany w bazie danych za pomocą **db.SaveChanges()**. Po zapisaniu przekierowuje do akcji "Index", która wyświetla listę filmów. Jeśli dane modelu nie są prawidłowe, ponownie przekazuje model **film** do widoku "Create", aby użytkownik mógł poprawić dane.

```
public ActionResult Create()
{
    var aktorzy = db.Aktorzy.OrderBy(a => a.Nazwisko).ToList();
    ViewBag.Aktorzy = aktorzy;
    ViewBag.AktorId = new SelectList(db.Aktorzy, "AktorId", "Imie");
    ViewBag.BibliotekaId = new SelectList(db.Biblioteki, "BibliotekaId", "Nazwa");
    return View();
}

// POST: Film/Create
// Aby zapewnić ochronę przed atakami polegającymi na przesyłaniu dodatkowych danych, włącz określone właśc
// Aby uzyskać więcej szczegółów, zobacz https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Create([Bind(Include = "FilmId,Tytul,Gatunek,AktorId,BibliotekaId")] Film film)
{
    if (ModelState.IsValid)
    {
        db.Filmy.Add(film);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    ViewBag.AktorId = new SelectList(db.Aktorzy, "AktorId", "Imie", film.AktorId);
    ViewBag.BibliotekaId = new SelectList(db.Biblioteki, "BibliotekaId", "Nazwa", film.BibliotekaId);
    return View(film);
}
```

```
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Film film = db.Filmy.Find(id);
    if (film == null)
    {
        return HttpNotFound();
    }
    var aktorzy = db.Aktorzy.OrderBy(a => a.Nazwisko).ToList();
    ViewBag.Aktorzy = aktorzy;
    ViewBag.AktorId = new SelectList(db.Aktorzy, "AktorId", "Imie", film.AktorId);
    ViewBag.BibliotekaId = new SelectList(db.Biblioteki, "BibliotekaId", "Nazwa", film.BibliotekaId);
    return View(film);
}

// POST: Film/Edit/5
// Aby zapewnić ochronę przed atakami polegającymi na przesyłaniu dodatkowych danych, włącz określone właściwo
// Aby uzyskać więcej szczegółów, zobacz https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Edit([Bind(Include = "FilmId,Tytul,Gatunek,AktorId,BibliotekaId")] Film film)
{
    if (ModelState.IsValid)
    {
        db.Entry(film).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    ViewBag.AktorId = new SelectList(db.Aktorzy, "AktorId", "Imie", film.AktorId);
    ViewBag.BibliotekaId = new SelectList(db.Biblioteki, "BibliotekaId", "Nazwa", film.BibliotekaId);
    return View(film);
}
```

Metoda **public ActionResult Edit(int? id)** sprawdza, czy **id** jest null. Następnie wyszukuje obiekt **Film** o określonym **id** w bazie danych za pomocą **db.Filmy.Find(id)**. Jeśli obiekt **film** jest null, to oznacza, że nie znaleziono filmu o podanym **id**. Następnie tworzy listę aktorów posortowaną alfabetycznie i przypisuje ją do **ViewBag.Aktorzy** w celu przekazania jej do widoku. Tworzy również **ViewBag.AktorId** i **ViewBag.BibliotekaId**, które zawierają listy wyboru (**SelectList**) aktorów i bibliotek. Obie listy są używane w widoku "Edit" do wyboru aktora i biblioteki dla edytowanego filmu. Na koniec zwraca widok "Edit" z modelem **film**.

Metoda **public ActionResult Edit([Bind(Include = "FilmId,Tytul,Gatunek,AktorId,BibliotekaId")] Film film)** przyjmuje parametr **Film film**, który jest modelem filmu przekazanym z formularza w widoku "Edit". Jeśli stan modelu (**ModelState**) jest prawidłowy, ustawia stan obiektu **film** na **EntityState.Modified**. Następnie zapisuje zmiany w bazie danych za pomocą **db.SaveChanges()**. Po zapisaniu przekierowuje do akcji "Index", która wyświetla listę filmów. Jeśli dane modelu nie są prawidłowe, ponownie przekazuje model **film** do widoku "Edit", aby użytkownik mógł poprawić dane.

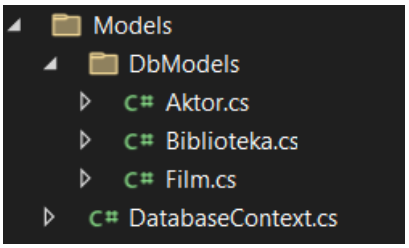
```
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Film film = db.Filmy.Find(id);
    if (film == null)
    {
        return HttpNotFound();
    }
    return View(film);
}

// POST: Film/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult DeleteConfirmed(int id)
{
    Film film = db.Filmy.Find(id);
    db.Filmy.Remove(film);
    db.SaveChanges();
    return RedirectToAction("Index");
}

0 references
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
```

Metoda **public ActionResult Delete(int? id)** sprawdza, czy **id** jest null. Następnie wyszukuje obiekt **Film** o określonym **id** w bazie danych za pomocą **db.Filmy.Find(id)**. Jeśli obiekt **film** nie jest null zwraca widok "Delete" z modelem **film**.

Metoda **public ActionResult DeleteConfirmed(int id)** przyjmuje parametr **int id**, który jest identyfikatorem filmu do usunięcia. Wyszukuje obiekt **Film** o określonym **id** za pomocą **db.Filmy.Find(id)**. Następnie usuwa ten obiekt z bazy danych za pomocą **db.Filmy.Remove(film)** i zapisuje zmiany w bazie danych za pomocą **db.SaveChanges()**. Po usunięciu przekierowuje do akcji "Index", która wyświetla listę filmów.

Metoda **protected override void Dispose(bool disposing)** to przesłonięta metoda z klasy bazowej **Controller**. Jest wywoływana podczas usuwania kontrolera. W przypadku wartości **disposing** równych true, metoda zwalnia zasoby bazodanowe przez wywołanie **db.Dispose()**. Następnie wywołuje metodę bazowej klasy **Controller** za pomocą **base.Dispose(disposing)** w celu zwolnienia innych zasobów. Ta metoda jest używana do zarządzania zasobami i pamięcią w kontrolerze.

Opis modeli




```
namespace podejsciedwa.Models.DbModels
{
    13 references
    public class Aktor
    {

        5 references
        public int AktorId { get; set; }
        1 reference
        public string Imie { get; set; }
        3 references
        public string Nazwisko { get; set; }
        1 reference
        public List<Film> Filmy { get; set; } = new List<Film>();

        2 references
        public Aktor() { }
        0 references
        public Aktor(int aktorId, string imie, string nazwisko, List<Film> filmy)
        {
            AktorId = aktorId;
            Imie = imie;
            Nazwisko = nazwisko;
            Filmy = filmy;
        }
    }
}
```

W klasie Aktor zostały dodane następujące właściwości publiczne: int AktorId, string Imie, string Nazwisko oraz lista Filmów List Filmy ustawiona na pustą listę. Następnie zostały utworzone konstruktory: nieparametryczny oraz parametryczny ustawiający wszystkie właściwości.

```
namespace podejsciedwa.Models.DbModels
{
    10 references
    public class Biblioteka
    {

        1 reference
        public int BibliotekaId { get; set; }
        1 reference
        public string Nazwa { get; set; }
        1 reference
        public List<Film> Filmy { get; set; } = new List<Film>();

        0 references
        public Biblioteka() { }
        0 references
        public Biblioteka(int bibliotekaId, string nazwa, List<Film> filmy)
        {
            BibliotekaId = bibliotekaId;
            Nazwa = nazwa;
            Filmy = filmy;
        }
    }
}
```

W klasie Biblioteka dodane zostały następujące właściwości publiczne: int BibliotekaId, string Nazwa, oraz lista List Filmy. Lista będzie zawierała Filmy, które są dodane do kolekcji przez użytkownika. Następnie dodano konstruktory: nieparametryczny oraz parametryczny ustawiający wszystkie właściwości.

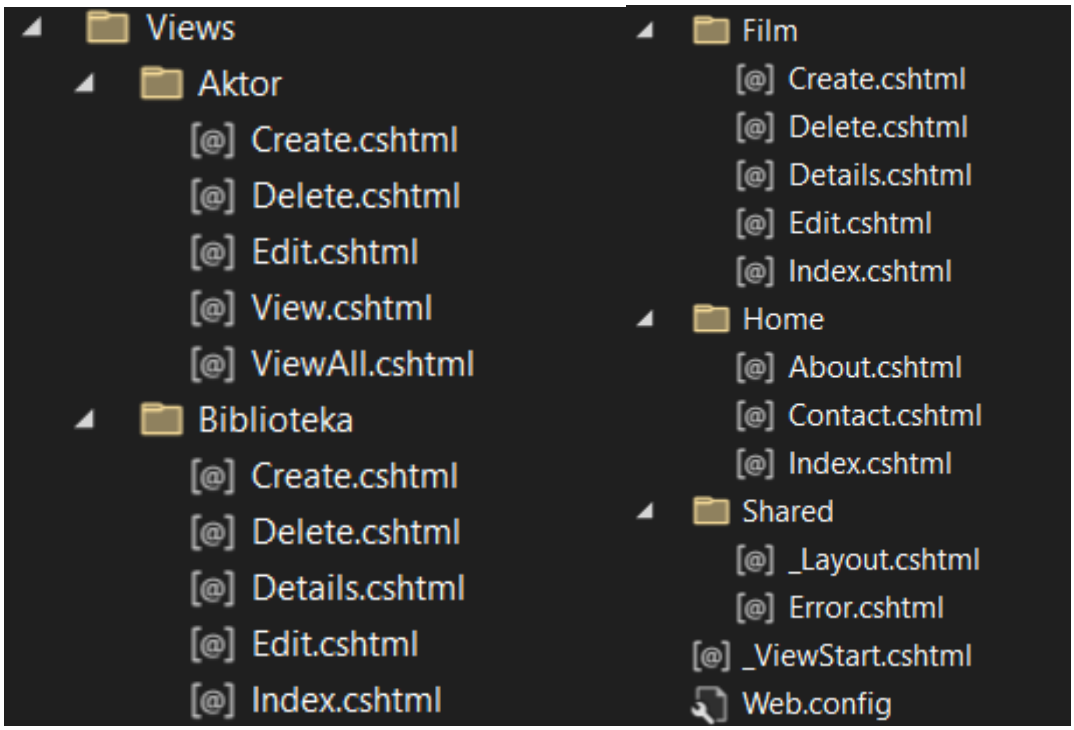
```
namespace podejsciedwa.Models.DbModels
{
    2 references
    public enum Gatunek { Komedia, Horror, Fantasy, SciFi, Kryminal, Thriller, Akcji, Dramat, Romans }
    15 references
    public class Film
    {


        1 reference
        public int FilmId { get; set; }
        1 reference
        public string Tytul { get; set; }

        1 reference
        public Gatunek Gatunek { get; set; }
        0 references
        public Film() { }
        0 references
        public Film(int filmId, string tytul, Gatunek gatunek)
        {
            FilmId = filmId;
            Tytul = tytul;
            Gatunek = gatunek;
        }
        3 references
        public int? AktorId { get; set; }
        1 reference
        public virtual Aktor Aktor { get; set; }
        3 references
        public int? BibliotekaId { get; set; }
        1 reference
        public virtual Biblioteka Biblioteka { get; set; }
    }
}
```

W klasie Film zostały dodane następujące właściwości publiczne: int FilmId, string Tytul, Gatunek Gatunek (enum) przyjmujący wartości wymienione powyżej. Następnie dodano konstruktory: nieparametryczny oraz parametryczny ustawiający wszystkie właściwości. Dodano również dodatkowe właściwości: public int AktorId, public virtual Aktor Aktor, public int BibliotekaId oraz public virtual Biblioteka Biblioteka.

```
namespace podejsciedwa.Models
{
    23 references
    public class DatabaseContext : DbContext
    {
        10 references
        public DatabaseContext() : base("podejsciedwaConnectionString") { }
        7 references
        public DbSet<Film> Filmy { get; set; }
        13 references
        public DbSet<Aktor> Aktorzy { get; set; }
        11 references
        public DbSet<Biblioteka> Biblioteki { get; set; }
    }
}
```

Klasa DatabaseContext.cs służy do przechowywania wszystkich niezbędnych informacji o bazie danych. Dziedziczy po klasie DbContext. Następnie dodano klasy DbSet: dla klas Film, Aktor, Biblioteka.

Opis widoków:



Dla każdego kontrolera w projekcie utworzone zostały widoki. Pozwalają one na odczyt, ale także zmiany w bazie danych. Widoki Create, Edit oraz Delete pozwalają kolejno na dodanie nowego rekordu do bazy, edycję oraz usunięcie elementów już istniejących. Widoki View i Details oraz ViewAll i Index są parami pokrewne. Pierwsze dwa pozwalają na wyświetlenie szerszej gamy informacji zawartych w poszczególnych elementach list. Kolejne dwa natomiast umożliwiają wyświetlenie pełnej listy utworzonych rekordów (listy aktorów, bibliotek, filmów - w zależności od wyboru).

W folderze Home zawarte zostały elementy strony takie jak Landing Page (Index) oraz krótki opis strony (About).

Folder Shared, a dokładniej plik _Layout odpowiada za wygląd oraz zawartość menu górnego oraz stopki. Tam zawarte zostały także najważniejsze style dla widoków strony.

Poniżej zawartość kilku elementów widoków.

Error.cshtml | **Create.cshtml** | Contact.cshtml | Index.cshtml | Edit.cshtml | FilmController

```
@model podejsciedwa.Models.DbModels.Aktor

@{
    ViewBag.Title = "Create";
}

<h2>Dodawanie Aktora</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Aktor</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Imie, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Imie, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Imie, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Nazwisko, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Nazwisko, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Nazwisko, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Dodaj aktora" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Wróć do listy aktorów", "ViewAll")
</div>
```

Delete.cshtml | Error.cshtml | Create.cshtml | Contact.cshtml | Index

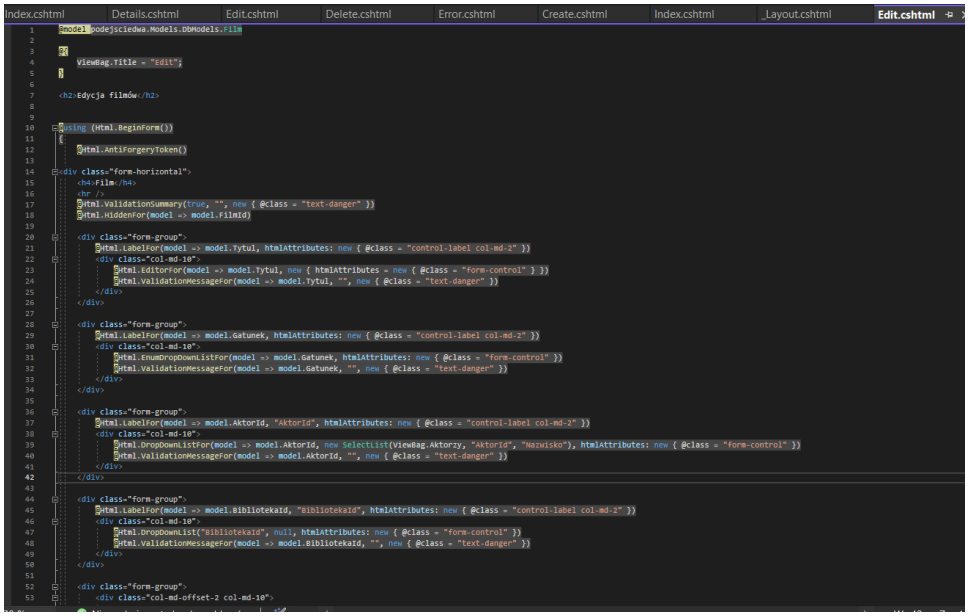
```
@model podejsciedwa.Models.DbModels.Biblioteka

@{
    ViewBag.Title = "Delete";
}

<h2>Usuwanie biblioteki</h2>

<h3>Czy jesteś pewien, że chcesz usunąć bibliotekę??</h3>
<div>
    <h4>Biblioteka</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Nazwa)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Nazwa)
        </dd>

    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Usuń bibliotekę" class="btn btn-default" /> |
            @Html.ActionLink("Wróć do listy bibliotek", "Index")
        </div>
    }
</div>
```

Details.cshtml | Edit.cshtml | Delete.cshtml | Error.cshtml | Create.cshtml

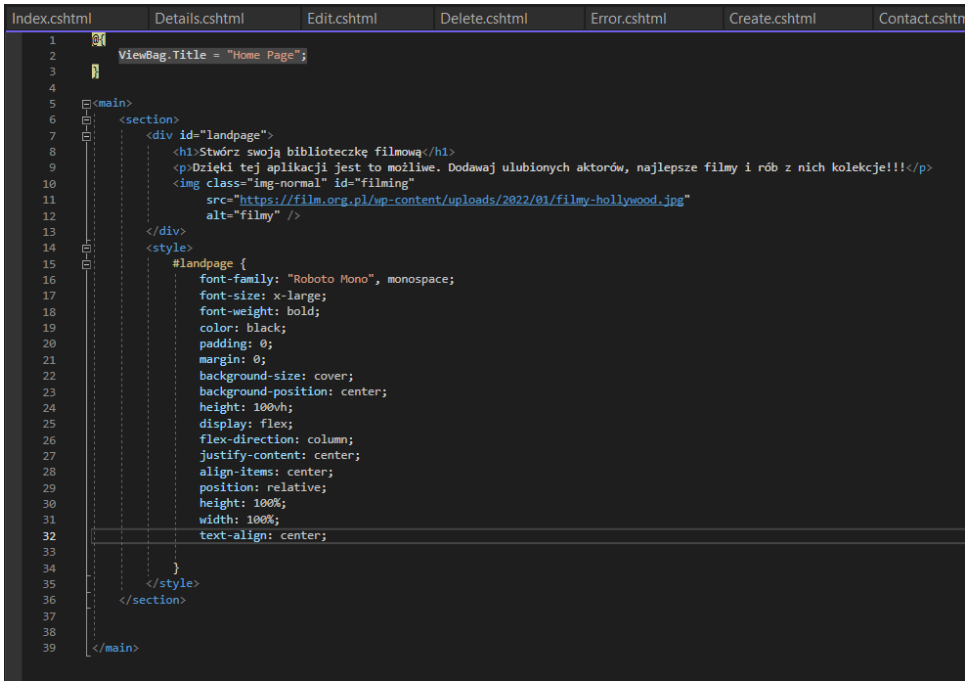
```
@model podejsciedwa.Models.DbModels.Biblioteka

@{
    ViewBag.Title = "Details";
}

<h2>Szczegóły</h2>

<div>
    <h4>Biblioteka</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.BibliotekaId)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.BibliotekaId)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Nazwa)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Nazwa)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edytuj", "Edit", new { id = Model.BibliotekaId }) |
    @Html.ActionLink("Wróć do listy bibliotek", "Index")
</p>

```

Index.cshtml | Details.cshtml | Edit.cshtml | Delete.cshtml | Error.cshtml | Cr

```
@model IEnumerable<podejsciedwa.Models.DbModels.Biblioteka>

@{
    ViewBag.Title = "Index";
}

<h2>Lista bibliotek</h2>

<p>
    @Html.ActionLink("Dodaj nową bibliotekę", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Nazwa)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Nazwa)
        </td>
        <td>
            @Html.ActionLink("Edytuj", "Edit", new { id=item.BibliotekaId }) |
            @Html.ActionLink("Szczegóły", "Details", new { id=item.BibliotekaId }) |
            @Html.ActionLink("Usuń bibliotekę", "Delete", new { id=item.BibliotekaId })
        </td>
    </tr>
}

</table>

```



```
Index.cshtml | Details.cshtml | Edit.cshtml | Delete.cshtml | Error.cshtml | Create.cshtml | Index.cshtml | _Layout.cshtml | Edit.cshtml
@model Dodejsciedwa.Models.DbModels.Film

@{
    ViewBag.Title = "Edit";
}

<h2>Edycja filmów</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

<div class="form-horizontal">
    <h4>Film</h4>
    <hr />
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    @Html.HiddenFor(model => model.FilmId)

    <div class="form-group">
        @Html.LabelFor(model => model.Tytul, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Tytul, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Tytul, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Gatunek, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EnumDropDownListFor(model => model.Gatunek, htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.Gatunek, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.AktorId, "AktorId", htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.DropDownListFor(model => model.AktorId, new SelectList(ViewBag.Aktorzy, "AktorId", "Nazwisko"), htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.AktorId, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.BibliotekaId, "BibliotekaId", htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.DropDownList("BibliotekaId", null, htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.BibliotekaId, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
```



```
Index.cshtml | Details.cshtml | Edit.cshtml | Delete.cshtml | Error.cshtml | Create.cshtml | Contact.cshtml
@{
    ViewBag.Title = "Home Page";
}

<main>
    <section>
        <div id="landpage">
            <h1>Stwórz swoją biblioteczkę filmową</h1>
            <p>Dzięki tej aplikacji jest to możliwe. Dodawaj ulubionych aktorów, najlepsze filmy i rób z nich kolekcje!!!</p>
            <img class="img-normal" id="filming"
                 src="https://film.org.pl/wp-content/uploads/2022/01/filmy-hollywood.jpg"
                 alt="filmy" />
        </div>
        <style>
            #landpage {
                font-family: "Roboto Mono", monospace;
                font-size: x-large;
                font-weight: bold;
                color: black;
                padding: 0;
                margin: 0;
                background-size: cover;
                background-position: center;
                height: 100vh;
                display: flex;
                flex-direction: column;
                justify-content: center;
                align-items: center;
                position: relative;
                height: 100%;
                width: 100%;
                text-align: center;

            }
        </style>
    </section>


</main>
```

Index.cshtml | Details.cshtml | Edit.cshtml | Delete.cshtml | Error.cshtml | Create.cshtml | Contact.cshtml | Index.cshtml | **_Layout.cshtml**

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - moja aplikacja platformy ASP.NET</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    <style>
        div {
            font-family: "Roboto Mono", monospace;
            font-size: x-large;
            font-weight: bold;
            color: black;
            margin-top: 20px;
            padding-left: 15px;
            padding-right: 15px;

        }

        div p {
            display: flex;
            justify-content: center;
            align-items: center;
            position: relative;
            text-align: center;
        }

        div dt, dd {
            font-size: large;
            font-weight: normal;
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark">
        <div class="container">
            @Html.ActionLink("Biblioteczka filmowa", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" title="Przełącz nawigację" aria-controls="navbarSupportedContent"
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse d-sm-inline-flex justify-content-between">
                <ul class="navbar-nav flex-grow-1">
                    <li>@Html.ActionLink("O aplikacji", "About", "Home", new { area = "" }, new { @class = "nav-link" })</li>
                    <li>@Html.ActionLink("Lista filmów", "Index", "Film", new { area = "" }, new { @class = "nav-link" })</li>
                    <li>@Html.ActionLink("Lista aktorów", "Viewall", "Aktor", new { area = "" }, new { @class = "nav-link" })</li>
                    <li>@Html.ActionLink("Lista bibliotek", "Index", "Biblioteka", new { area = "" }, new { @class = "nav-link" })</li>
                </ul>
            </div>
        </div>
    </nav>
```

Baza danych

W ramach projektu została utworzona baza danych oparta na klasach, która zawiera trzy tabele. Każda tabela umożliwia tworzenie nowych list.

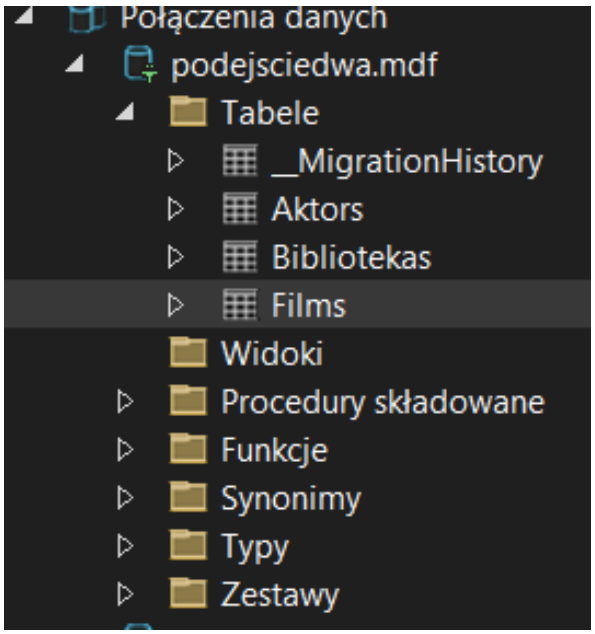


Tabela "Aktors" służy do dodawania aktorów do bazy danych. Tutaj przechowywane są informacje dotyczące aktorów, takie jak imię i nazwisko.

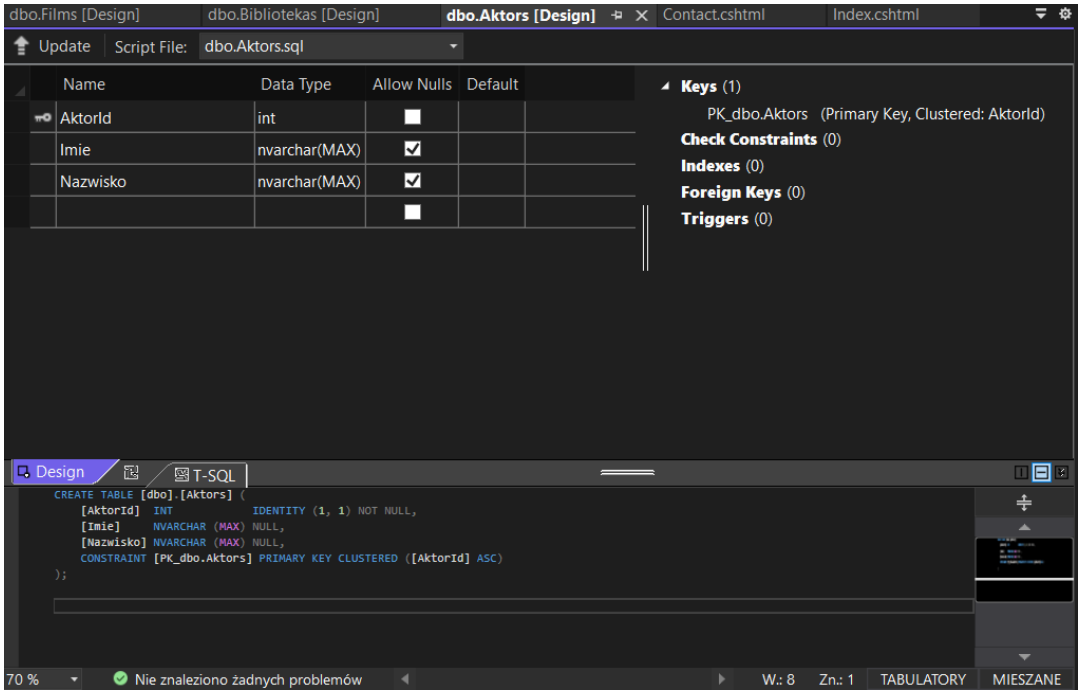
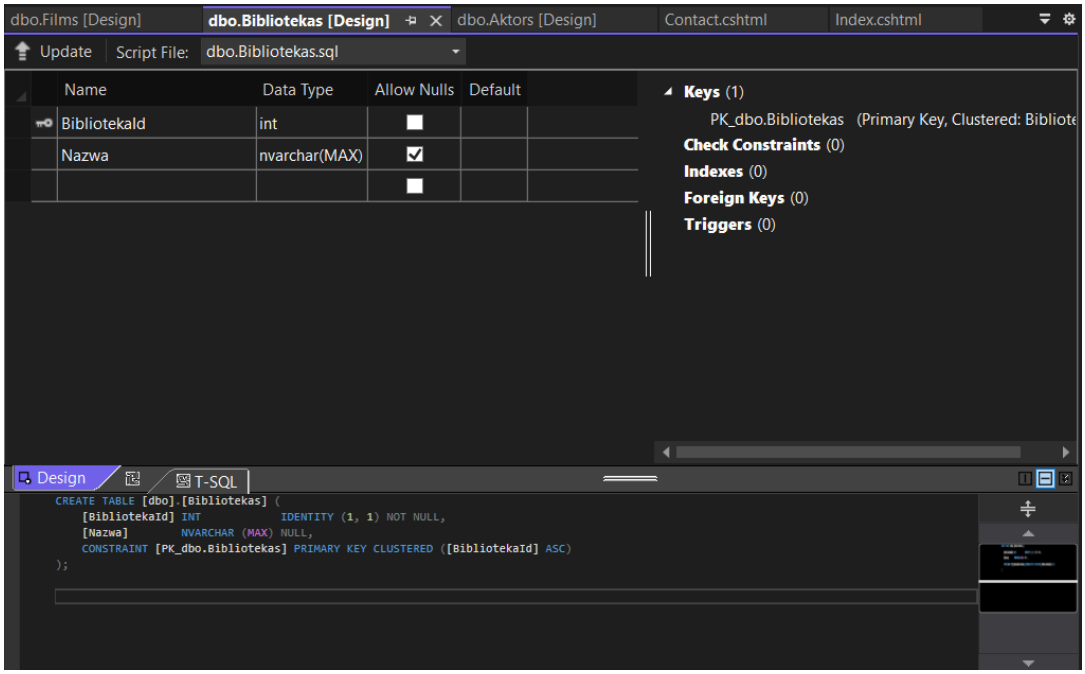
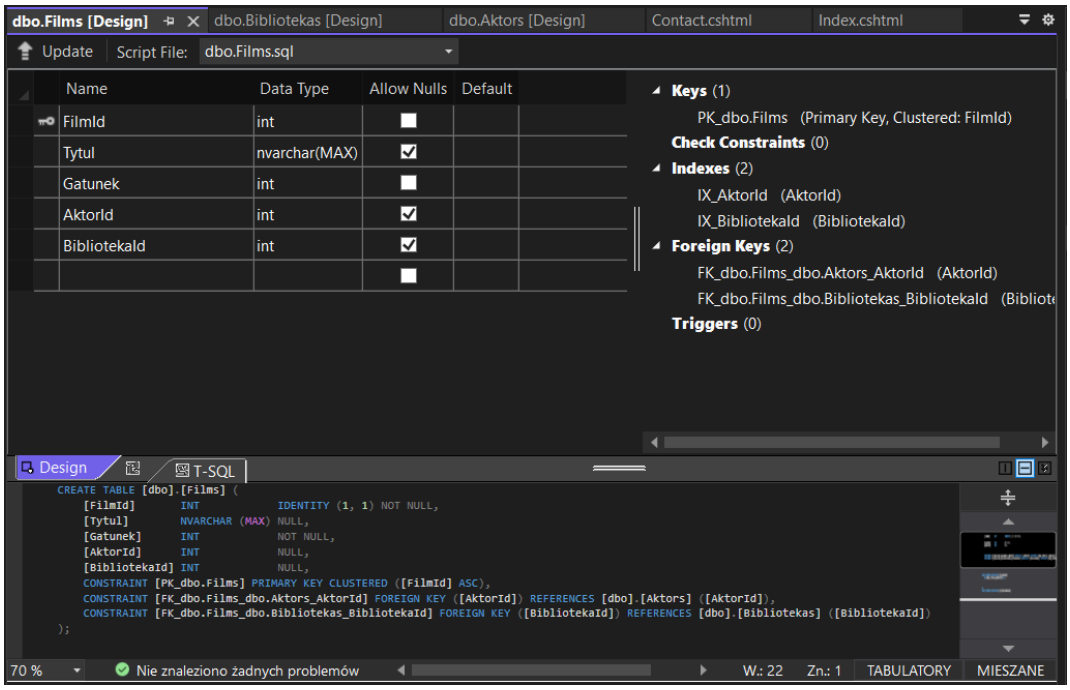


Tabela "Bibliotekas" umożliwia tworzenie list bibliotek. Przechowywane są w niej nazwy bibliotek.



Ostatnia tabela "Films" zawiera klucz główny oraz dwa klucze obce. Pozwala to na powiązanie aktora i biblioteki z filmem. Klucze obce umożliwiają przypisanie jednego aktora lub biblioteki do więcej niż jednego filmu. Dzięki temu można tworzyć listy filmów i powiązywać z nimi odpowiednie informacje o aktorach i bibliotekach.



Dzięki tym tabelom i powiązaniom można efektywnie zarządzać informacjami o aktorach, bibliotekach i filmach w bazie danych.

W ramach projektu można również sprawdzić zawartość list utworzonych w aplikacji.

dbo.Films [Data] | dbo.Bibliotekas [Data] | **dbo.Aktors [Data]**

Max Rows: 1000

AktorId	Imie	Nazwisko
2	Ilona	Ostrowska
3	Leonardo	DiCaprio
4	Adam	Sandler
NULL	*NULL*	*NULL*

dbo.Films [Data] | **dbo.Bibliotekas [Data]**

Max Rows: 1000

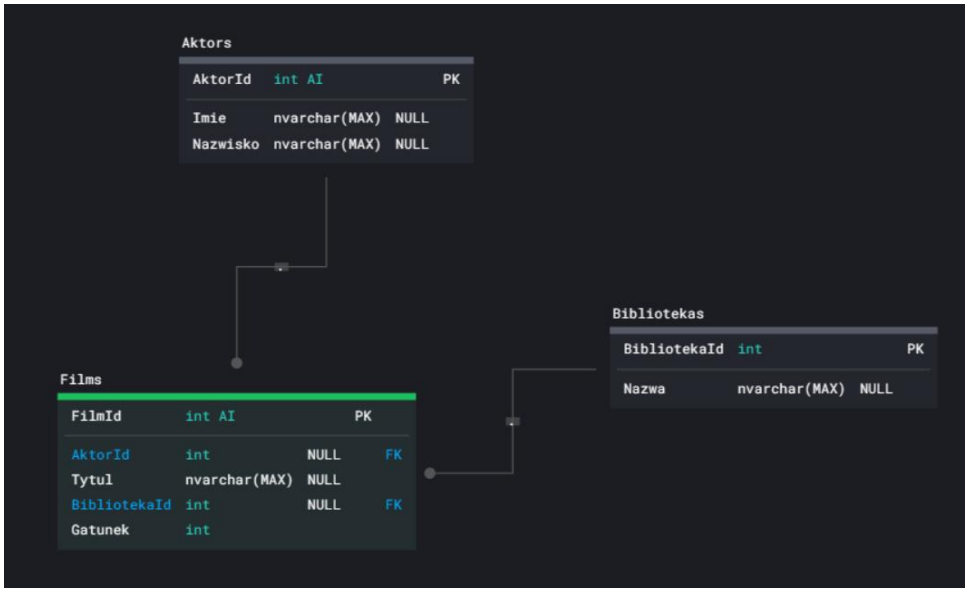
BibliotekaId	Nazwa
3	Netflix
4	SkyShowTime
5	HBO Max
NULL	*NULL*

dbo.Films [Data] | dbo.Bibliotekas [Data] | dbo.Aktors [Data] | dbo.Films [Desig

Max Rows: 1000

FilmId	Tytul	Gatunek	AktorId	BibliotekaId
2	Iluzja	5	2	5
3	Harry Potter	2	4	3
4	Wilk z Wall Stre...	4	3	4
NULL	*NULL*	*NULL*	*NULL*	*NULL*

Diagram bazy danych


Aktors
AktorId int AI PK
Imie nvarchar(MAX) NULL
Nazwisko nvarchar(MAX) NULL
Bibliotekas
BibliotekaId int PK
Nazwa nvarchar(MAX) NULL
Films
FilmId int AI PK
AktorId int NULL FK
Tytul nvarchar(MAX) NULL
BibliotekaId int NULL FK
Gatunek int