Dokumentacja Libripolis

Witold Nowakowski 131506, Jakub Polak 131573

Ogólny opis projektu

Libripolis to aplikacja internetowa dla lokalnej biblioteki publicznej, mająca na celu ułatwienie zarządzania zasobami bibliotecznymi oraz usprawnienie interakcji z czytelnikiem. Aplikacja umożliwia przeglądanie dostępnych tytułów książek oraz wypożyczenie ich przez zarejestrowanego użytkownika.

Funkcjonalność

Rejestracja i logowanie

- Umożliwienie użytkownikom rejestracji konta w celu uzyskania dostępu do dodatkowych funkcji.

Katalog książek online

- Publiczne przeglądanie dostępnych książek w bibliotece z możliwością wyszukiwania po nazwie oraz wyświetlania szczegółowych informacji o książkach.

Zarządzanie wypożyczeniami i rezerwacjami

- Zarejestrowani użytkownicy mogą rezerwować książki online oraz zarządzać swoimi wypożyczeniami, sprawdzając terminy zwrotu i ewentualnie przedłużając wypożyczenia.

Architektura

Aplikacja Libripolis jest napisana w języku programowania C# wykorzystująca MVC (Model-View-Controler) gdzie kontrolery, widoki oraz modele są oddzielone. Ma to na celu ułatwić zarządzanie kodem oraz jego skalowalność.

Backend

- Język programowania: C#
- Framework: ASP.NET MVC
- Baza danych: SQL Server

Frontend

- CSHTML
- CSS

Modele

Model Book

```
public class Book
{
    public int Id { get; set; }
    public string? ImageUrl { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string Author { get; set; }
    public int Pages { get; set; }

    public int BookTypeId { get; set; }
    public virtual BookType? BookType { get; set; }

    public string? userId { get; set; }
    public virtual IdentityUser? user { get; set; }

}
```

Model BookType

```
public class BookType
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string? userId { get; set; }
    public virtual IdentityUser? user { get; set; }
}
```

Model Borrow

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


    public int BookId { get; set; }
    public virtual Book? Book { get; set; }

    public string? userId { get; set; }
    public virtual IdentityUser? user { get; set; }

    public DateTime Start { get; set; }
    public DateTime End { get; set; }
}
```

Kontrolery

Do każdego w wyżej wymienionych modeli został utworzony kontroler

BooksController

Indeks

- Lista książek.
- Obsługuje opcjonalny parametr searchString do filtrowania książek po tytule.
- Zwraca widok z listą książek.

Details:

- Wyświetla szczegóły konkretnej książki na podstawie przekazanego identyfikatora (id).
- Zwraca widok z detalami książki.

Create:

- Dostępna tylko dla użytkowników z rolą "Admin".
- Obsługuje zarówno żądanie GET, jak i POST.
- W przypadku GET, zwraca formularz do utworzenia nowej książki.
- W przypadku POST, dodaje nową książkę do bazy danych po sprawdzeniu poprawności modelu.

Edit:

- Dostępna tylko dla użytkowników z rolą "Admin".
- Obsługuje zarówno żądanie GET, jak i POST.
- W przypadku GET, zwraca formularz edycji dla konkretnej książki.
- W przypadku POST, aktualizuje dane książki w bazie danych po sprawdzeniu poprawności modelu.

Delete:

- Dostępna tylko dla użytkowników z rolą "Admin".
- Wyświetla potwierdzenie usuwania dla konkretnej książki.
- Zwraca widok potwierdzenia usuwania.

Metoda DeleteConfirmed:

- Obsługuje potwierdzenie usunięcia książki.
- Usuwa książkę z bazy danych i zapisuje zmiany.

Metoda BookExists:

- Sprawdza, czy książka istnieje w bazie danych na podstawie przekazanego identyfikatora.

Atrybuty Autoryzacji:

- Akcje Create, Edit, i Delete mają atrybut [Authorize(Roles = "Admin")], co oznacza, że są dostępne tylko dla użytkowników z rolą "Admin".

Zabezpieczenie przed NULL:

- W akcji Index kontroler sprawdza, czy kolekcja Book w kontekście bazy danych nie jest nullem i zwraca odpowiedni problem, jeśli tak.

Ustawianie userId:

- W akcjach Create i Edit, kontroler ustawia userId na podstawie zalogowanego użytkownika.

BookTypesController

Details:

- Wyświetla szczegóły danego typu książek na podstawie przekazanego identyfikatora (id).
- Zwraca widok z detalami typu książek.

Create:

- Obsługuje zarówno żądanie GET, jak i POST.
- W przypadku GET, zwraca formularz do utworzenia nowego typu książek.
- W przypadku POST, dodaje nowy typ książek do bazy danych po sprawdzeniu poprawności modelu.

Edit:

- Obsługuje zarówno żądanie GET, jak i POST.
- W przypadku GET, zwraca formularz edycji dla konkretnej kategorii książek.
- W przypadku POST, aktualizuje dane kategorii książek w bazie danych po sprawdzeniu poprawności modelu.

Delete:

- Wyświetla potwierdzenie usuwania dla konkretnej kategorii książek.
- Zwraca widok potwierdzenia usuwania.

Metoda DeleteConfirmed:

- Obsługuje potwierdzenie usunięcia kategorii książek.
- Usuwa kategorię książek z bazy danych i zapisuje zmiany.

Metoda BookTypeExists:

- Sprawdza, czy dany typ książek istnieje w bazie danych na podstawie przekazanego identyfikatora.

Atrybut Autoryzacji:

- Kontroler ma atrybut [Authorize(Roles = "Admin")], co oznacza, że wszystkie akcje w tym kontrolerze są dostępne tylko dla użytkowników z rolą "Admin".

Zabezpieczenie przed NULL:

- W akcji Index kontroler sprawdza, czy kolekcja BookType w kontekście bazy danych nie jest nullem i zwraca odpowiedni problem, jeśli tak.

Ustawianie userId:

- W akcjach Create i Edit, kontroler ustawia userId na podstawie zalogowanego użytkownika.

BorrowsController

Index:

- Wyświetla listę wypożyczeń dla zalogowanego użytkownika.
- Zwraca widok z listą wypożyczeń.

allBorows:

- Dostępna tylko dla użytkowników z rolą "Admin".
- Wyświetla listę wszystkich wypożyczeń we wszystkich kontekstach.
- Zwraca widok z listą wszystkich wypożyczeń.

Details:

- Wyświetla szczegóły danego wypożyczenia na podstawie przekazanego identyfikatora (id).
- Zwraca widok z detalami wypożyczenia.

Create:

- Wyświetla formularz do utworzenia nowego wypożyczenia.
- Sprawdza, czy użytkownik już ma wypożyczoną tę samą książkę z ważnym terminem wypożyczenia.
- Dodaje nowe wypożyczenie do bazy danych po sprawdzeniu poprawności modelu.

Edit:

- Wyświetla formularz edycji dla danego wypożyczenia.
- Aktualizuje dane wypożyczenia w bazie danych po sprawdzeniu poprawności modelu.

Delete:

- Wyświetla potwierdzenie usunięcia dla danego wypożyczenia.
- Zwraca widok potwierdzenia usuwania.

Metoda DeleteConfirmed:

- Usuwa wypożyczenie z bazy danych i zapisuje zmiany.

Metoda BorrowExists:

- Sprawdza, czy dane wypożyczenie istnieje w bazie danych na podstawie przekazanego identyfikatora.

Atrybut Autoryzacji:

- Akcja allBorows ma atrybut [Authorize(Roles = "Admin")], co oznacza, że jest dostępna tylko dla użytkowników z rolą "Admin".

Zabezpieczenie przed NULL:

- W akcjach Index, allBorows, Details, Create, Edit, i Delete kontroler sprawdza, czy kolekcja Borrow w kontekście bazy danych nie jest nullem.

Ustawianie userId:

- W akcjach Index, Create, i Edit, kontroler ustawia userId na podstawie zalogowanego użytkownika.

Widoki

Każdy z modeli i jego kontrolerów ma widoki (domyślne jest ich 5: Create, Delete, Details, Edit, Index)

Dla modelu Borrows jest dodatkowy widok allBorrows

Views/Books

- Create - jest to formularz dodający książkę i składa się z pól Label, Input by wprowadzić pola takie jak w modelu Book z przyciskiem „Create” by dodać książkę.
- Delete – wyświetla się informacja jaki obiekt jest usuwany, ponownie wyświetla wszystkie informacje z modelu Book i dwa przyciski „Delete” lub „Back To List”.
- Details – wyświetla szczegółowe informacje na temat dodanej książki z widoku create oraz daje możliwość zalogowanemu użytkownikowi na wypożyczenie książki przez przycisk połączony z widokiem Views/Borrows/Create lub wrócić „Back To List”.
- Edit - wyświetla to samo co widok „Create” natomiast nadpisuje obecną książkę a nie dodaje nowej książki lub wrócić „Back To List”.
- Index – Główna strona dla widoku Books. Na niej znajduje się tabela z dodanymi książkami wraz z wyszukiwarką po nazwie. Do każdej książki dołączone są 3 przyciski od widoków (Detalis, Edit, Delete).
- Widoki takie jak Create,Edit,Delete dla View/Books są ograniczone i tylko administrator aplikacji może z nich korzystać

Views/BookTypes

- Create- W tym widoku tworzymy poprzez formularz typ książki (Atrybut Name), użytkownik, który tworzy typ jest automatycznie wybrany w „select” poprzez userId oraz potwierdzamy przyciskiem „submit”
- Delete- wyświetlamy wszystkie informacje odnośnie obiektu z modelu BookType i możemy go usunąć
- Details- Wyświetla się szczegółowa informacja na temat obiektu dodanego przez administratora.
- Edit – Edycja obecnych typów książek
- Index- Wyświetla się tabel z listą wszystkich obecnie dodanych typów książek, do każdego typu jest opcja edycji i usunięcia

Views/Borrows

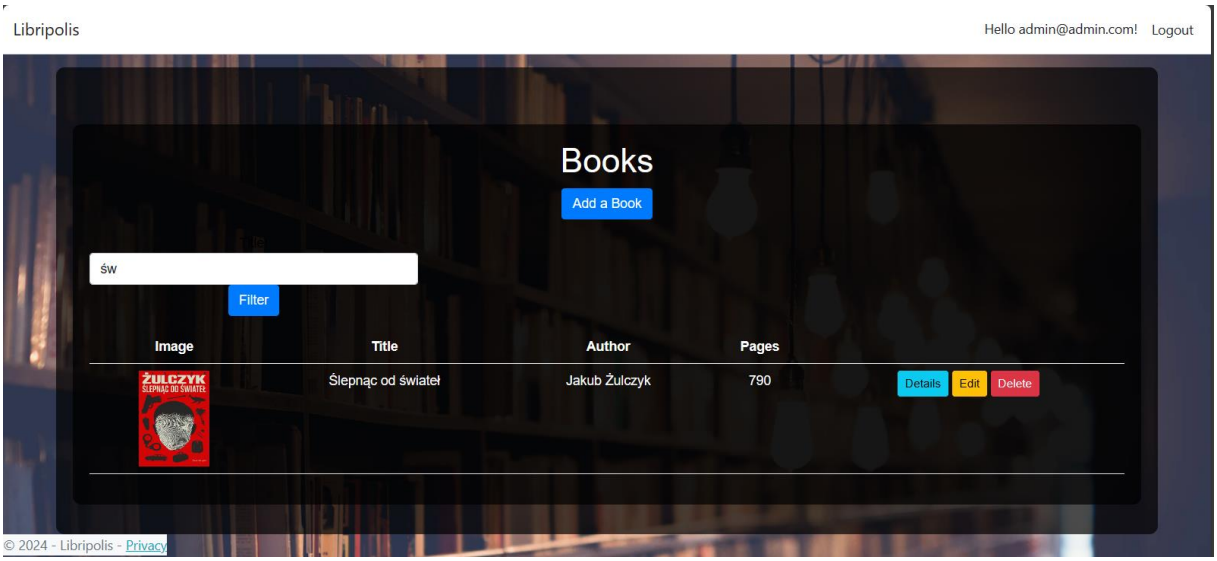
- allBorrows - Lista wszystkich wypożyczeń w postaci tabeli (Widok stworzony tylko dla administratora aplikacji, możliwość usunięcia wypożyczenia, edycji i zobaczenia szczegółów).
- Create - Widok dla stworzenia wypożyczenia, automatycznie wybiera użytkownika i książkę bo ich ID i w formularzu mają atrybut hidden, pozostaje tylko ustalić przedział czasowy Start i End wypożyczenia. Zatwierdzamy przyciskiem „Create” lub wychodzimy „Back To List”.
- Delete – Wyświetla się stworzone wypożyczenie wraz z informacją i możliwością usunięcia przyciskiem „Delete”.
- Details – Wyświetla wszystkie informacje z modelu Borrow.
- Edit – Pole edytujące dodane dane.
- Index – Widok ograniczony tylko dla zalogowanego użytkownika, jest tabela pokazująca wszystkie wypożyczenia (tytuł, kto, start, end) – wraz z przyciskami „Change Date” oraz „Details”.

Zabezpieczenia i Dodatki

Search bar

- W widoku Views/Books/Index.cshtml jest dodany search bar, który wyświetla książki na wpisaną przez nas nazwę, przykładowo mając w bazie książek tytuł „Ślepnąc od

Świateł” po wpisaniu w search bar „św” ten tytuł nam się pokaże wraz z innymi, które maja „św”



Authorize

- Zabezpieczenia metodą [Authorize(Roles=...)]

Wszystkie widoki BookTypes ale też i w modelu Books „Create”, „Edit”, „Delete” oraz w Borrows widok „allBorrows” – zostały ograniczone tylko dla poziomu administratora aplikacji.

A tak wygląda kod generujący role i administratora dla aplikacji.

```
using (var scope = app.Services.CreateScope()) // Tworzenie ról na stronie
{
    var roleManager =
scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

    var roles = new[] { "Admin", "Member", "Guest" };
```

```
    foreach (var role in roles)
    {
        if (!await roleManager.RoleExistsAsync(role))
            await roleManager.CreateAsync(new IdentityRole(role));



    }
}

using (var scope = app.Services.CreateScope()) // Tworzenie ról na stronie
{
    var userMember =
scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

    string email = "admin@admin.com";
    string password = "B00ks1$";

    if (await userMember.FindByEmailAsync(email) == null)
    {
        var user = new IdentityUser();
        user.UserName = email;
        user.Email = email;


        await userMember.CreateAsync(user, password);

        await userMember.AddToRoleAsync(user, "Admin");
    }
      }
```

Zabezpieczenie przed wypożyczeniem drugi raz tej samej książki kiedy jeszcze nie przekroczyła terminu oddania End

- W BorrowsController przy Create metodą GET/POST dodana jest część kodu, która ma za zadanie sprawdzić czy do tej książki ten user nie posiada jeszcze ważnego wypożyczenia. Jeśli tak to wraca z powrotem do swoich wypożyczeń Borrow/Index

```
  var existingBorrow = _context.Borrow
.Where(b => b.userId == userId && b.BookId == bookId && b.End > DateTime.Now)
.FirstOrDefault();
```

Logowanie do aplikacji

- Użytkownik może założyć konto i następnie się zalogować dzięki wybraniu opcji w tworzeniu projektu „Pojedyncze Konta”. Dzięki temu w _Layout.cshtml pojawia się _LoginPartial odpowiedzialny za te przyciski Register oraz Login

```
<ul class="navbar-nav">
```

```
    <partial name="_LoginPartial" />
</ul>
```

Informacje dodatkowe

- Plik readme.txt zawiera krótką instrukcję jak uruchomić aplikację u siebie na komputerze

Spis treści