Modelowanie i analiza systemów informatycznych

dokumentacja projektu system biblioteczny

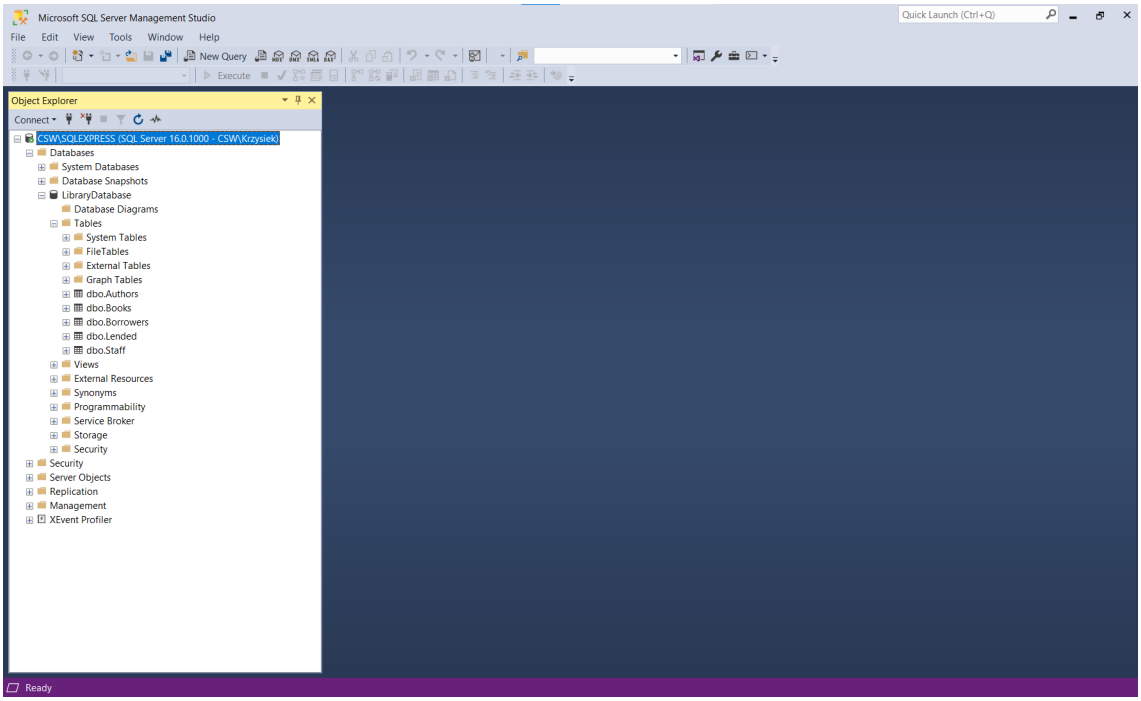
Krzysztof Kalita grupa 3, Tymoteusz Januła-Gniadek grupa 2

19 grudnia 2023

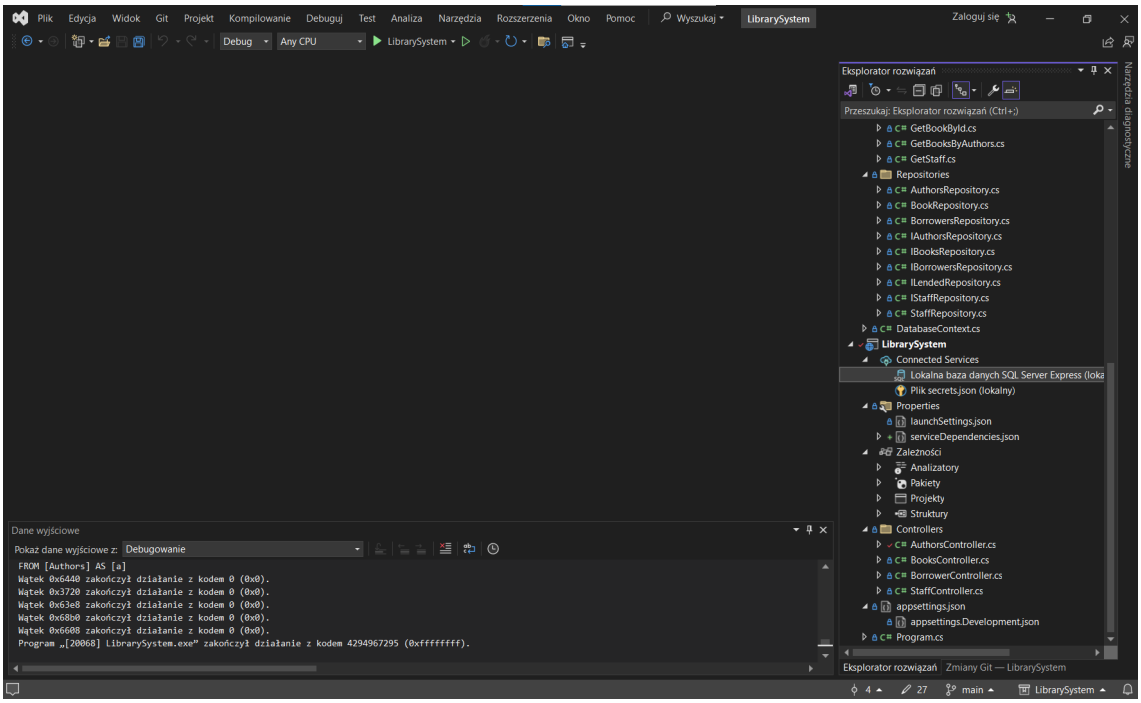
Część I

Opis programu

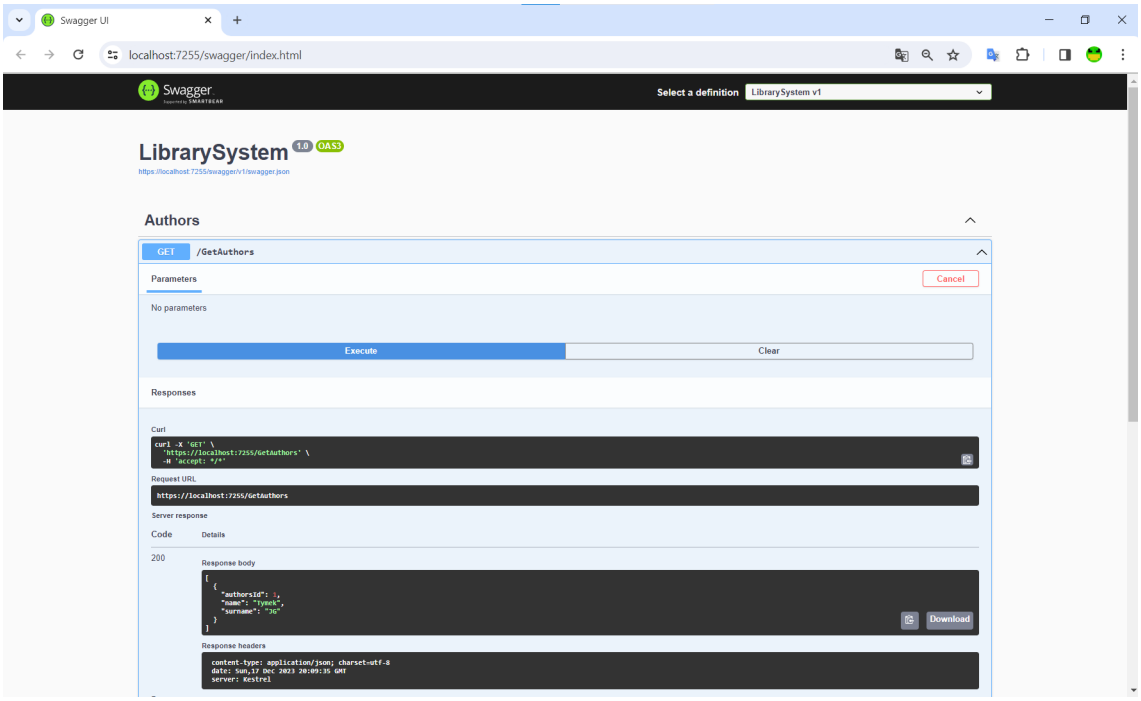
System Biblioteczny. System składa się z bazy danych SQL która zawiera dane na temat książek, pracowników, autorów książek oraz osób pożyczających.Klient używa aplilkacji webowej napisanej w języku angular w celu korzystanie z aplikacji. Użytkownik może tam dodawać książki, wyporzyczać książki, oddawać książi, dodawać pracowników nowe osoby wyporzyczające.Aplikacja webowa połączona jest z bazą danych za pomocą serwera asp.Net.Core napisanym w wzorcu MediatR. APlikacja używa systemu elekcji w celu zapewnienia że tylko jeden użytkownik w danej chwili może sprawdzać specyficzną książkę.



Rysunek 1: SQL Server Management Studio



Rysunek 2: Visual studio 2022

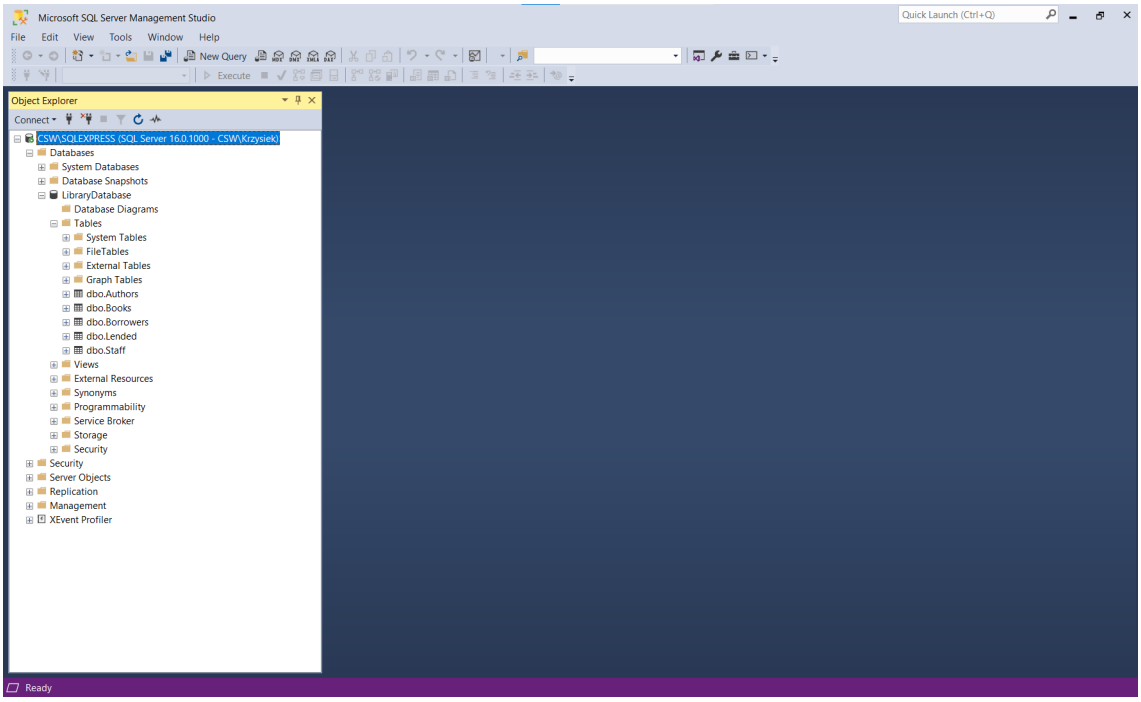


Rysunek 3: Strona internetowa

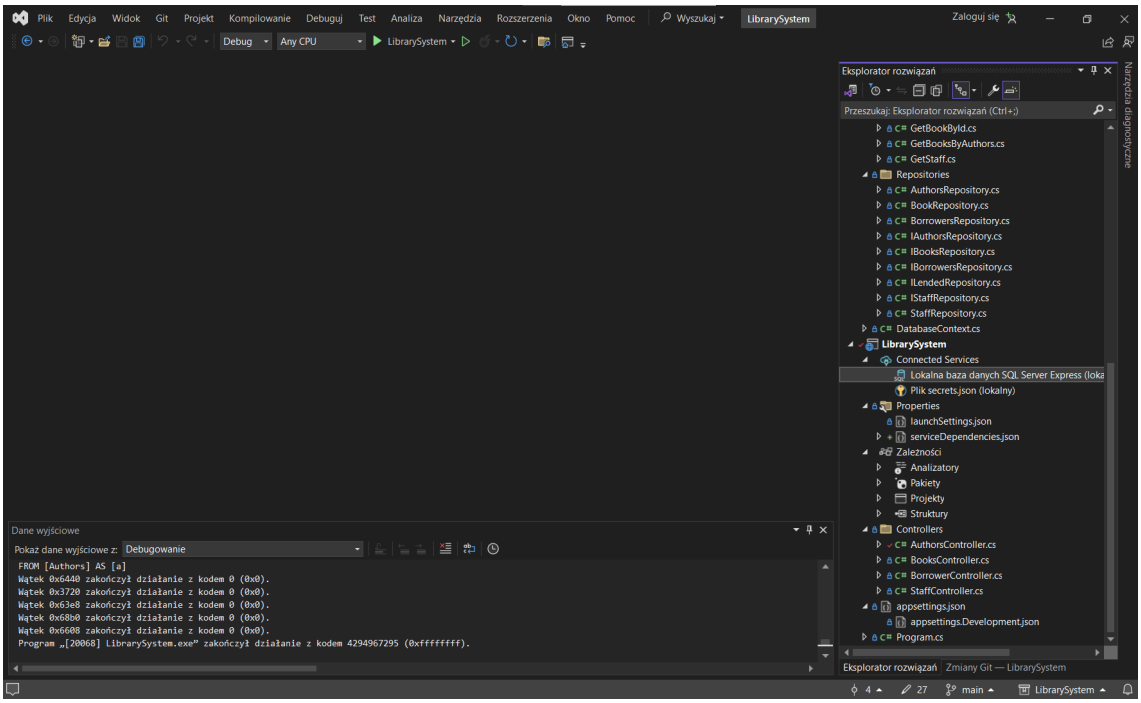
Instrukcja obsługi

W celu uruchomienia projeku w trybe developerskim potrzebne jest:

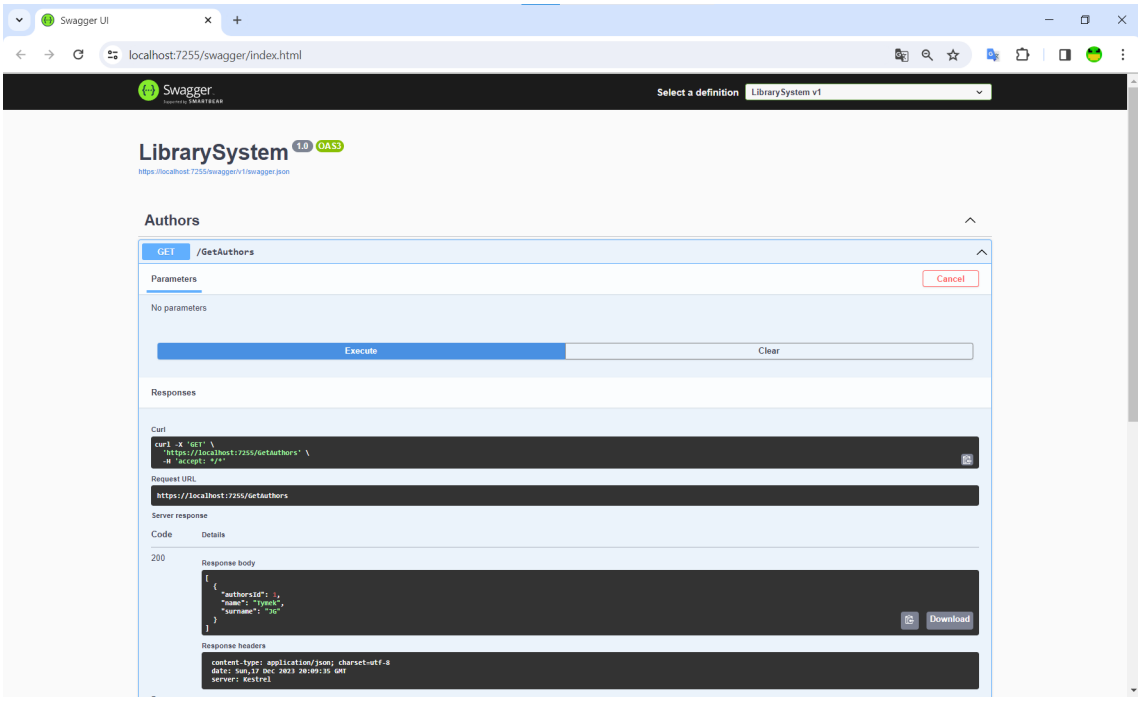
- Dodanie odpowiedniej bazy danych
- Połączenie bazy danych z aplikacją w C#
- Uruchomienie tej aplikacji
- Uruchomienie pojektu angular
  - Uruchomienie Visual Studio code
  - Zainstalowanie Angulara
  - Zainstalowanie angular materials, SignalR oraz crypto.js
  - Uruchomienie projektu w konsoli komendą ng serve



Rysunek 4: SQL Server Management Studio

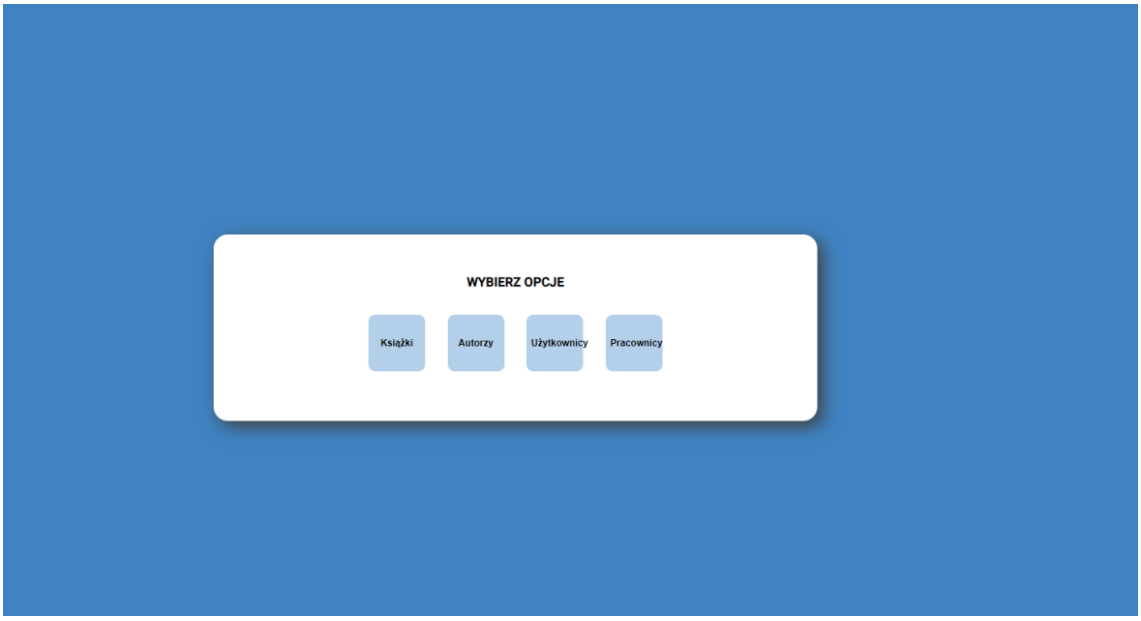


Rysunek 5: Visual studio 2022



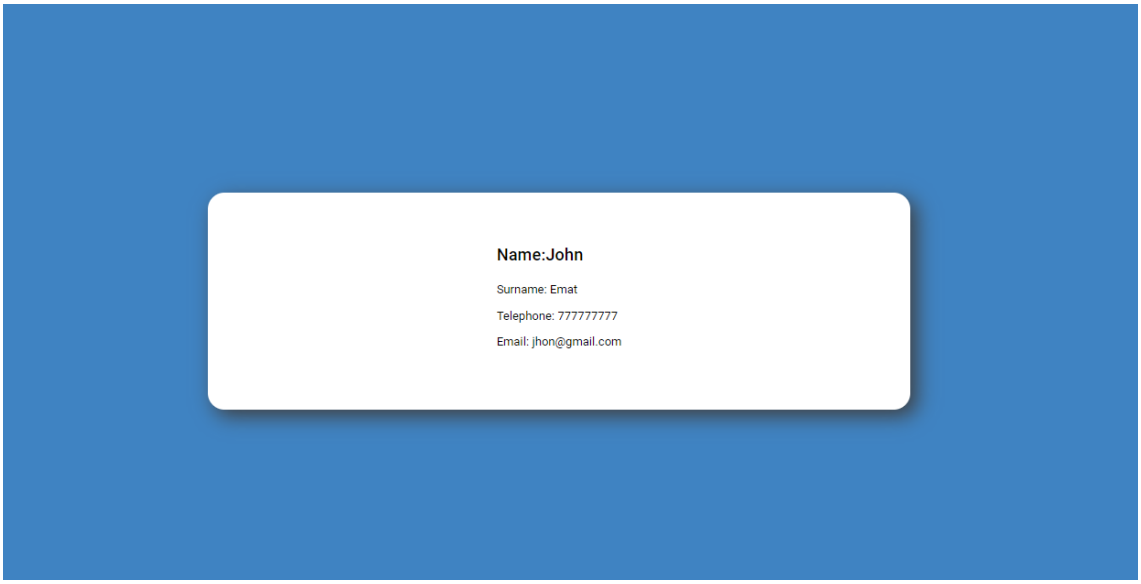
Rysunek 6: Strona internetowa

Rysunek 7: Logowanie do aplikacji.

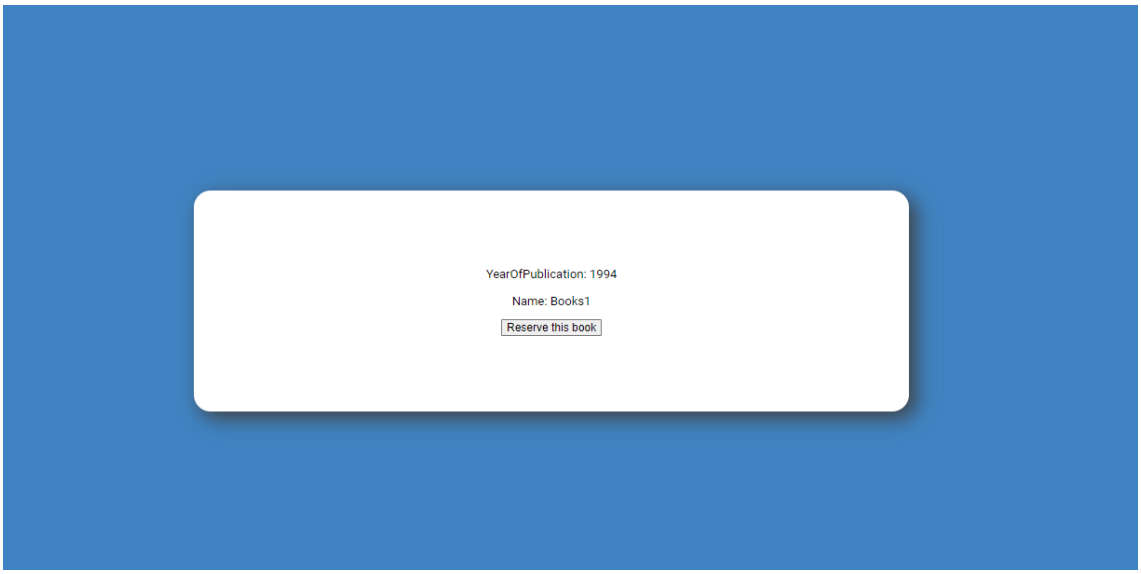


Rysunek 8: Wybranie opcji z 4 przycisków: książki, autorzy, użytkownicy, pracownicy.

Rysunek 9: Wybranie opcji książki i wypisanie listy książek.



Rysunek 10: Dane wypożyczającego książkę.



Rysunek 11: Rezerwacja książki.

0.1 Opis wdrożenia

Dodatkowe informacje

Wymagania:

- System Elekcji
- System Rozproszony
- Wyciąganie danych z bazy danych
- Modyfikacjia elementów bazy danych
- Dodawanie elementów do bazy danych
- Blokowanie dostępu do danych za pomocą systemu elekcji
- Szyfrowanie danych

Podział pracy: Tymoteusz Januła-Gniadek:
Implementacjia aplikacji Kszysztof kalita:
Utworzenie Bazy danych SQL. Wykonanie rozeznania w tematyce projektu. Utworzenie Dokumentacji

Część II

Opis działania

MediatR to wzorzec stosowany głównie w aplikacjach .NET, bazujący na koncepcji pośrednika. W naszej aplikacja został wykorzystany. Jego głównym zadaniem jest minimalizacja bezpośrednich powiązań między klasami, co przyczynia się do uproszczenia zarządzania złożonością aplikacji oraz jej utrzymaniem.

W ramach tego wzorca, wszelka komunikacja między obiektami jest kierowana przez centralny punkt, którym jest mediator. Mediator ten odpowiada za przekazywanie informacji – takich jak zdarzenia, polecenia czy żądania – między nadawcami a ich adresatami. Dzięki temu, zamiast bezpośredniego połączenia między elementami systemu, występuje jedynie ich powiązanie z mediatorem, co sprzyja modularności i ułatwia testowanie.

Główne zalety wykorzystania MediatR w środowisku .NET obejmują:

- Prostsze Testowanie Jednostkowe: Ze względu na słabsze powiązania między komponentami, tworzenie testów jednostkowych staje się łatwiejsze.
- Większa Elastyczność Systemu: System staje się bardziej elastyczny i łatwiejszy w rozbudowie o nowe funkcje, bez konieczności ingerencji w istniejący kod.
- Redukcja Bezpośrednich Zależności: Ograniczenie bezpośrednich powiązań między klasami przyczynia się do lepszej organizacji i struktury kodu.

MVC, czyli Model-View-Controller, to wzorzec projektowy często stosowany w tworzeniu aplikacji internetowych i innych interaktywnych aplikacji. W naszej aplikacja został wykorzystany. Oto kluczowe elementy wzorca MVC:

- Model: Reprezentuje logikę biznesową i dane aplikacji. Model odpowiada za dostęp do bazy danych, manipulowanie danymi oraz ich przetwarzanie. Jest to "serce"aplikacji, które przechowuje dane, reguły i logikę biznesową.
- View: Odpowiada za prezentację danych użytkownikowi. Są to interfejsy użytkownika, które prezentują dane i umożliwiają interakcję z użytkownikiem.
- Controller: Działa jako pośrednik między modelem a widokiem. Kontroler odbiera wejście od użytkownika, przetwarza je (często z wykorzystaniem modelu), a następnie zwraca odpowiedni widok.

Bezpieczeństwo

Podczas logowania się wysłanie dane są wpierw hashowanie za pomocą SHA256. Dzięki temu przesłane dane są jawne więc newet w przypadku przechwycenia przez osąbę trzecią nie mogą one zostać wykorzystanie. Dane bazy danych są także zaszyfrowanie. Dzięki temu nawet jeśli nastąpi wyciek bazy danych i dane zostaną odszyfrowanie, zahashowanie hasła nie mogą zostać użyte. Aplikacja posiada też zabezpiecznie w postaci AuthGuard która pozwala przechodzić przez kolejne komponenty aplikacji tlyko jeśli jest się zalogowanym.

Algorytmy

Data: Brak danych wejściowych
Result: Proces nasłuchiwania i odpowiadania na żądania
while *true* **do**
  WyślijŻądanieWszystkichKlientów();
  odpowiedź := NasłuchujNaOdpowiedź();
  if *odpowiedź zawiera odpowiedniąWartość* **then**
    Zakończ;
  else
    Kontynuuj nasłuch;
  end
  if *OtrzymanoNoweŻądanie()* **then**
    WyślijOdpowiedź(z własnąWartość);
  end
end

Algorithm 1: Algorytm elekcji: Algorytm nasłuchiwania i odpowiadania na żądania

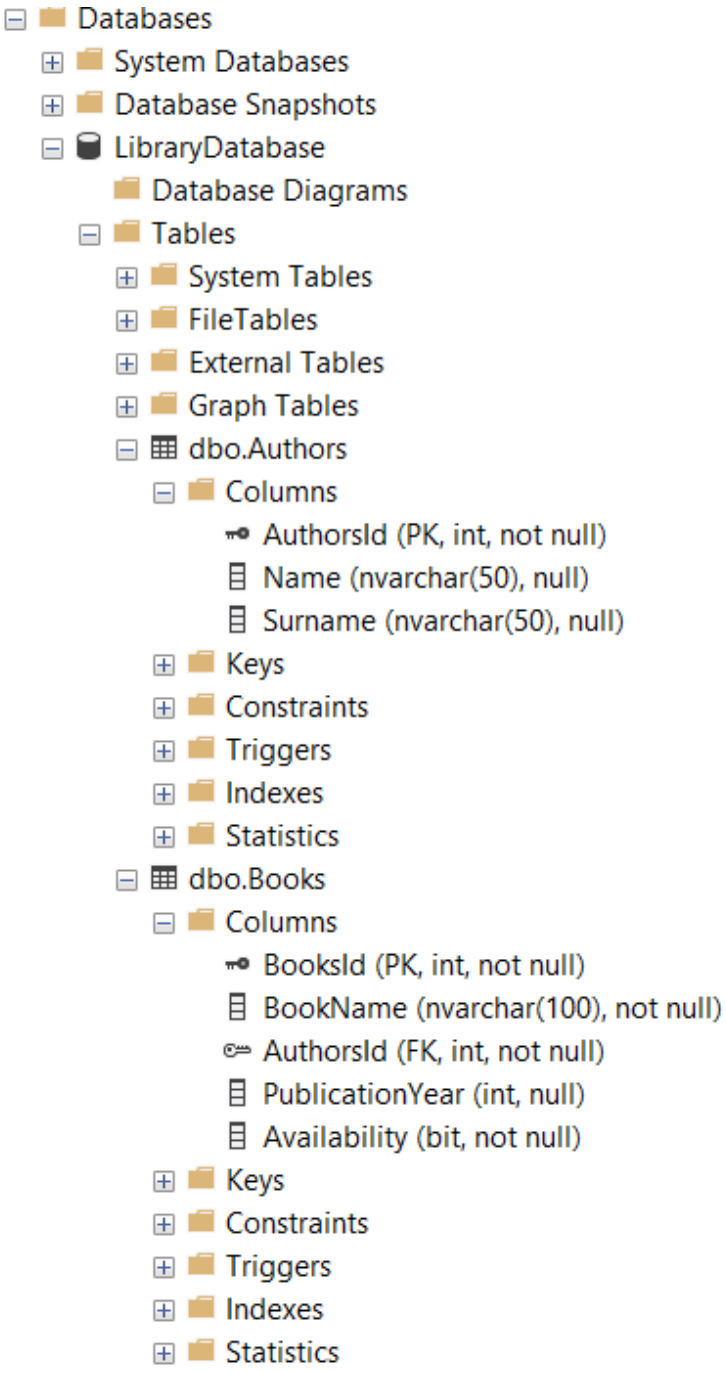
Bazy danych

Baza jest napisana w sql i jest wykorzystywana w SQL Server Management Studio. Baza składa się z 5 tablic:

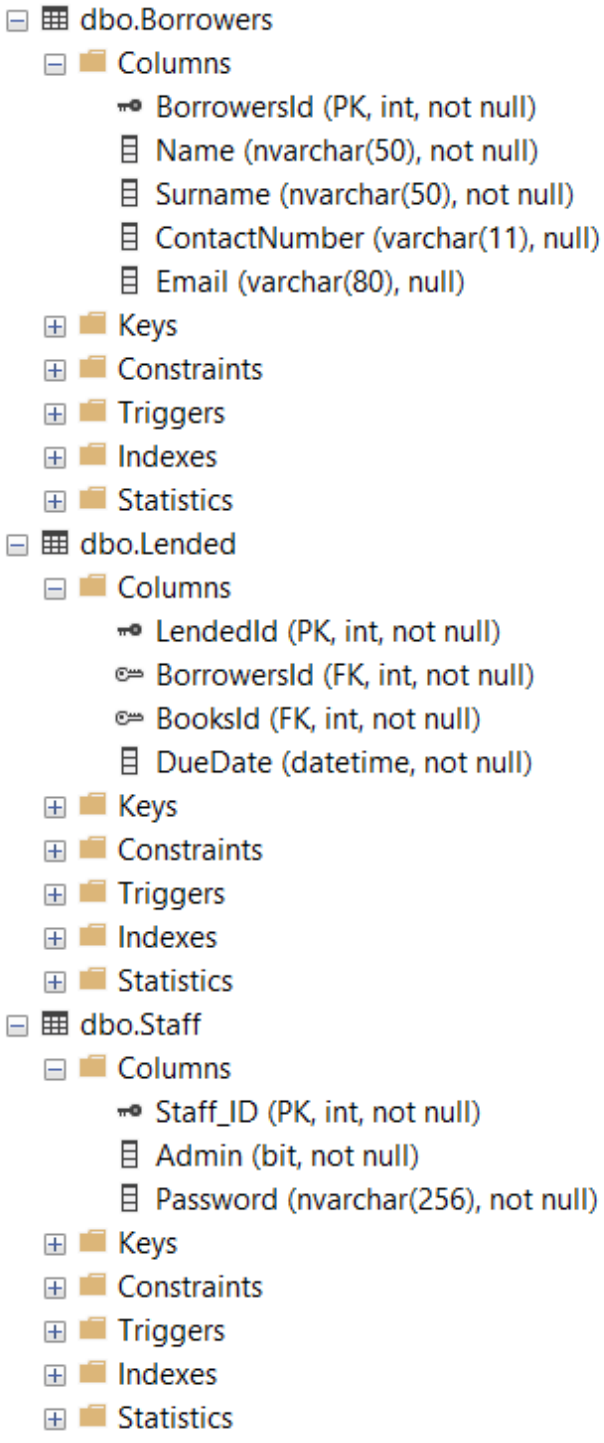
1. Authors - jest to tablica z autorami książek i ma 3 kolumny:
   - AuthorsId - jest to identyfikator autora(jako klucz główny w bazie danych).
   - Name - imie autora.
   - Surname - nazwisko autora.
2. Books - jest to tablica z książkami i ma 5 kolumn:
   - BooksId - jest to identyfikator książki(jako klucz główny w bazie danych).
   - BookName - tytuł ksiązki.
   - AuthorsId - jest to identyfikator autora(jako klucz obcy w bazie danych).
   - PublicationYear - rok publikacji.
   - Availability - dostepność książki w bibliotece.
3. Borrowers - jest to tablica z danymi pożyczających książki(klientów) i ma 5 kolumn:
   - BorrowersId - jest to identyfikator klienta(jako klucz główny w bazie danych).

- Name - imie klienta.
- Surname - nazwisko klienta.
- ContactNumber - numer kontaktowy do klienta.
- Email - email do klienta.

4. Lended - jest to tablica z wypożyczeniami i ma 4 kolumny:
    - LendedId - jest to identyfikator wypożyczenia(jako klucz główny w bazie danych).
    - BorrowersId - jest to identyfikator klienta(jako klucz obcy w bazie danych).
    - BooksId - jest to identyfikator książki(jako klucz obcy w bazie danych).
    - DueDate - data do oddania.
5. Staff - jest to tablica z personelem i ma 3 kolumny:
    - Staff_ID - jest to identyfikator pracownika(jako klucz główny w bazie danych).
    - Admin - nazwa pracownika.
    - Password - hasło pracownika.

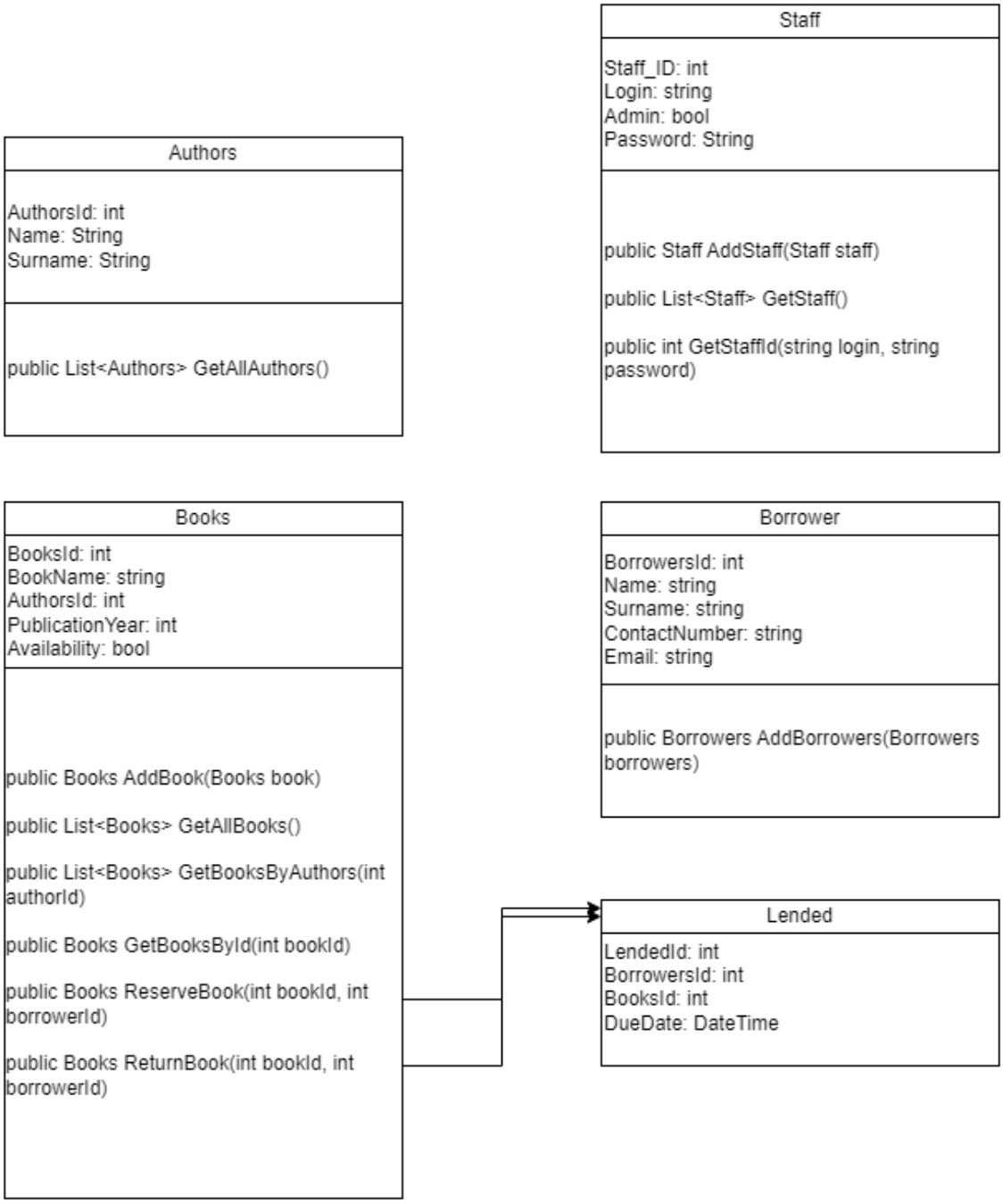


Rysunek 12: Tablice: Authors, Books.

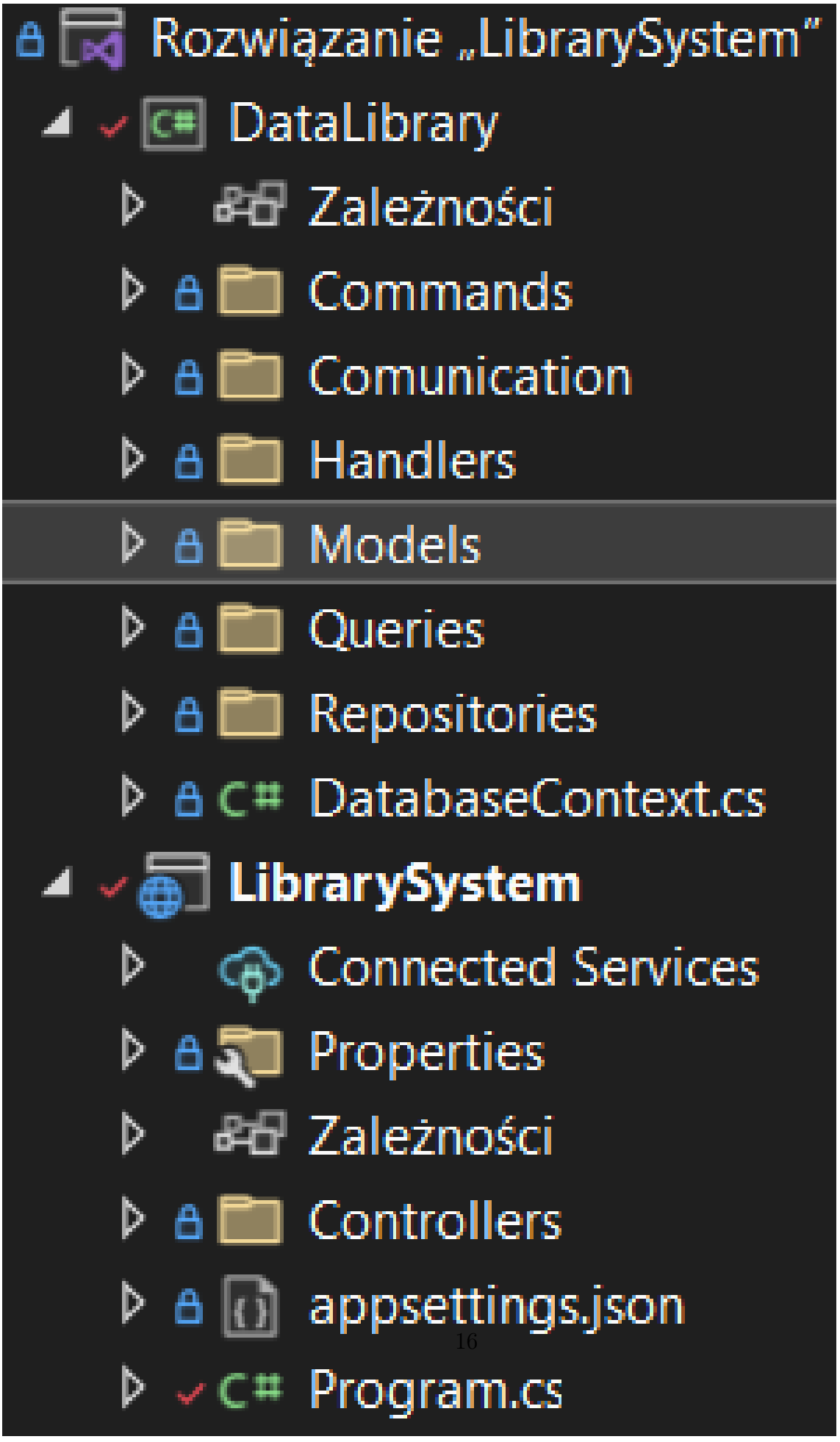


Rysunek 13: Tablice: Borrowers, Lended, Staff.

Implementacja systemu



Rysunek 14: Wizualizacja klas/metod (Diagram UML)



Rysunek 15: Struktura plików w visula studio 2022.

Baza danych jest przechowywana na serwerze SQL Server Management Studio. Jest pobierana przez backend(program w c# otwierany przez visual studio 2022). Backend ten jest zabezpieczony certyfikatem. Backend ten używa dane z bazy danych do zarzadzania nimi za pomocą różnych funkcji takich jak dodaj książkę, dodawanie nowych książek, dodawanie pracowników (pracownik), dodawanie autorów do książek, dodawanie klientów, rezerwowanie i zwracanie książek przez klientów. Backend jest pomostem(połączeniem) miedzy bazą danych, a aplikacją w angularze. Aplikacja w angularze wykorzystuje funkcje z backend ze strony internetowej za pomocą adresów url(np: https://localhost:7255/GetStaff) i w aplikacji widzimy interfejs graficzny, w którym jest wiele funkcjonalności. Między innymi w angularze zaimplementowany jest system blokowaine zasobów który wykorzystuje SignalR chatHub. Angular złorzony jest z komponentów pomiędzy którymi pozwala się poruszać za pomocą routingu. W aplikacji posiadamy komponent logowanie, komponent książek, komponent szczegółów książki który pozwala także na jej rezerwacje, komponent użytkowników, komponent pracownikó, komponent książek wypożyczonych przez użytkowników który pozwala zwrócić książki, serwis do komunikacji z backendem i więcej.

Models- przechowuje dane, pliki z klasami(tablicami z bazy danych) z różnymi parametrami i funkcjami. Np:

```
namespace DataLibrary.Models
{
    public class Authors
    {
        [Key]
        public int AuthorsId { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public Authors( string name, string surname)
        {
            Name = name;
            Surname = surname;
        }
        public Authors() { }
    }
}
```

Plik Authors.cs: Klasa autorzy przechowuje parametry AuthorsId, Name, Surname i ma konstruktor z dwoma parametrami name i surname i konstruktor domyślny.

```
namespace DataLibrary.Models
{
    public class Books
    {
        [Key]
        public int BooksId { get; set; }
        public string BookName { get; set; }
        public int AuthorsId { get; set; }
        public int PublicationYear { get; set; }
        public bool Availability { get; set; }
```

```
11          public Books( string bookName, int authorsId, int
                publicationYear, bool availability)
12          {
13              BookName = bookName;
14              AuthorsId = authorsId;
15              PublicationYear = publicationYear;
16              Availability = availability;
17          }
18          public Books() { }
19      }
20  }
```

Plik Books.cs: Klasa książki przechowuje parametry BooksId, BookName, AuthorsId, PublicationYear, Availability i ma konstruktor z czterema parametrami bookName i authorsId i publicationYear i availability i konstruktor domyślny.

```
1  namespace DataLibrary.Commands
2  {
3      public record AddBook(Books book) : IRequest<Books>;
4  }
```

AddBook.cs w Commands - Kod reprezentuje komendę dodania nowej książki do systemu. IRequest<Books> to interfejs z biblioteki MediatR, który definiuje typ żądania zwracającego obiekt typu Books. W tym przypadku AddBook jest żądaniem (komendą), które po przetworzeniu zwróci obiekt typu Books. Użycie MediatR w tym kontekście pozwala na odseparowanie logiki wykonującej komendę od reszty systemu, co ułatwia testowanie, utrzymanie i skalowanie aplikacji.

```
1  namespace DataLibrary.Queries
2  {
3
4      public record GetAllBooks() : IRequest<List<Books>>;
5  }
```

GetAllBooks.cs w Queries. Rekord GetAllBooks w kontekście aplikacji .NET reprezentuje zapytanie o pobranie wszystkich książek.

```
1  namespace DataLibrary.Repositories
2  {
3      public class AuthorsRepository : IAuthorsRepository
4      {
5          private readonly DatabaseContext _context;
6          public AuthorsRepository(DatabaseContext context)
7          {
8              _context = context;
9          }
10
11         public List<Authors> GetAllAuthors()
12         {
13             var results = _context.Authors.ToList();
14             return results;
15         }
16
17
```

```
18
19      }
20
21
22  }
```

AuthorsRepository.cs w Repositories. Klasa AuthorsRepository jest odpowiedzialna za operacje dotyczące danych autorów w bazie danych. Jest to implementacja repozytorium dla encji Authors.

```
1  namespace DataLibrary.Repositories
2  {
3    public class BookAccessRepository : IBookAccessRepository
4    {
5      private readonly Dictionary<int, List<BookAccess>> _bookAccessState
           = new Dictionary<int, List<BookAccess>>();
6
7      public bool TryLockBook(int bookId, int userId)
8      {
9        if (!_bookAccessState.TryGetValue(bookId, out var accessList))
10       {
11         // If the book is not in the dictionary, create a new entry
12         accessList = new List<BookAccess>();
13         _bookAccessState[bookId] = accessList;
14       }
15
16       // Check if any user is already accessing the book
17       if (accessList.Any(access => access.IsLocked))
18       {
19         // Book is already locked
20         return false;
21       }
22
23       // Lock the book for the current user
24       accessList.Add(new BookAccess { BookId = bookId, UserId = userId,
             IsLocked = true });
25       return true;
26     }
27
28     public void ReleaseBookLock(int bookId)
29     {
30       if (_bookAccessState.TryGetValue(bookId, out var accessList))
31       {
32         // Release the lock on the book for all users
33         accessList.RemoveAll(access => access.IsLocked);
34       }
35     }
36   }
37 }
```

BookAccessRepository.cs w Repositories. Reprezentuje repozytorium zarządzające dostępem do książek. Klasa BookAccessRepository zarządza blokadami na książkach, pozwalając na zarezerwowanie książki przez użytkownika i późniejsze jej zwolnienie. Jest to przykład wzorca projektowego repozytorium.

```
1     namespace DataLibrary.Repositories
2 {
3     public class BookRepository : IBooksRepository
4     {
5         private readonly DatabaseContext _context;
6         public BookRepository(DatabaseContext context)
7         {
8             _context = context;
9         }
10        public Books AddBook(Books book)
11        {
12            _context.Add(book);
13            _context.SaveChanges();
14            return book;
15        }
16
17        public List<Books> GetAllBooks()
18        {
19            var results = _context.Books.ToList();
20            return results;
21        }
22
23        public List<Books> GetBooksByAuthors(int authorId)
24        {
25            var result = _context.Books.Where(x => x.AuthorsId ==
                  authorId).ToList();
26            return result;
27        }
28
29        public Books GetBooksById(int bookId)
30        {
31            var result = _context.Books.FirstOrDefault
32       (x => x.BooksId == bookId);
33            return result;
34        }
35
36        public Books ReserveBook(int bookId, int borrowerId)
37        {
38            var result = _context.Books.FirstOrDefault
39        (x => x.BooksId == bookId);
40            if (result == null) return null;
41            result.Availability = false;
42            DateTime dueDate = DateTime.Now.AddMonths(3);
43            Lended newLend = new Lended(borrowerId,bookId,dueDate);
44
45
46            _context.Lended.Add(newLend);
47            _context.SaveChanges();
48            return result;
49        }
50
51        public Books ReturnBook(int bookId, int borrowerId)
52        {
53            var result = _context.Books.FirstOrDefault
54        (x => x.BooksId == bookId);
```

```
55              var result2 = _context.Lended.FirstOrDefault
56             (x => x.BooksId == bookId && x.BorrowersId == borrowerId);
57
58
59              _context.Lended.Remove(result2);
60              _context.SaveChanges();
61              return result;
62          }
63      }
64
65
66  }
```

BookRepository.cs w Repositories. BookRepository zawiera metodę do dodawania, pobierania i aktualizowania danych książek w bazie danych. Umożliwia również zarządzanie procesem wypożyczania i zwrotu książek. Jest to typowe repozytorium w aplikacjach korzystających z wzorca architektonicznego repozytorium, które abstrahuje logikę dostępu do danych, umożliwiając łatwiejsze testowanie i utrzymanie kodu.

```
1  namespace DataLibrary.Repositories
2  {
3      public class BorrowersRepository : IBorrowersRepository
4      {
5          private readonly DatabaseContext _context;
6          public BorrowersRepository(DatabaseContext context)
7          {
8              _context = context;
9          }
10
11         public Borrowers AddBorrowers(Borrowers borrowers)
12         {
13             _context.Borrowers.Add(borrowers);
14             _context.SaveChanges();
15             return borrowers;
16         }
17
18
19     }
20
21
22 }
```

BorrowersRepository.cs w Repositories. BorrowersRepository umożliwia dodanie nowych wypożyczających (borrowers) do bazy danych. Jest to typowe repozytorium w aplikacjach, które stosują wzorzec architektoniczny repozytorium. W tym przypadku, repozytorium to zajmuje się operacjami związanymi z wypożyczającymi w bazie danych.

```
1  namespace DataLibrary.Repositories
2  {
3
4      public class StaffRepository : IStaffRepository
5      {
6          private readonly DatabaseContext _context;
7          public StaffRepository(DatabaseContext context)
```

```
8           {
9               _context = context;
10          }
11
12          public Staff AddStaff(Staff staff)
13          {
14               _context.Staff.Add(staff);
15              _context.SaveChanges();
16              return staff;
17          }
18
19          public List<Staff> GetStaff()
20          {
21              var result = _context.Staff.ToList();
22              return result;
23          }
24
25          public int GetStaffId(string login, string password)
26          {
27            if (_context.Staff.Any(u => u.Login == login && u.Password ==
                  password))
28            {
29              return _context.Staff.Single(u => u.Login == login && u.
                    Password == password).Staff_ID;
30            }
31            else
32            {
33              return 0;
34            }
35      }
36    }
37 }
```

StaffRepository.cs w Repositories. Klasa StaffRepository w przestrzeni nazw DataLibrary.Repositories jest implementacją repozytorium dla obiektów typu Staff. StaffRepository umożliwia dodawanie nowych pracowników do bazy danych, pobieranie listy wszystkich pracowników oraz weryfikację tożsamości pracownika na podstawie danych logowania. Repozytorium to zajmuje się operacjami związanymi z pracownikami w bazie danych.

```
1 namespace DataLibrary.Repositories
2 {
3     public interface IAuthorsRepository
4     {
5         List<Authors> GetAllAuthors();
6
7     }
8 }
```

IAuthorsRepository.cs w Repositories. Interfejsy w programowaniu służą jako kontrakt, określający zestaw metod, które muszą zostać zaimplementowane przez klasę, która implementuje ten interfejs. IAuthorsRepository jest interfejsem, który określa wymagania dotyczące funkcji do zarządzania danymi autorów w aplikacji.

```
1 namespace DataLibrary.Handlers
```

```
{

    public class AddBookHandler : IRequestHandler<AddBook, Books>
    {
        private readonly IBooksRepository _booksRepository;

        public AddBookHandler(IBooksRepository booksRepository)
        {
            _booksRepository = booksRepository;
        }
        public async Task<Books> Handle(AddBook request,
            CancellationToken cancellationToken)
        {
            var result = _booksRepository.AddBook(request.book);
            return result;
        }
    }
}
```

AddBookHandler.cs w Handlers. Klasa AddBookHandler w przestrzeni nazw DataLibrary.Handlers jest implementacją wzorca projektowego Mediator, używając biblioteki MediatR, popularnej w aplikacjach .NET. Klasa ta jest obsługiwaczem (handlerem) żądania AddBook i odpowiada za przetwarzanie tego żądania. Zajmuje się dodawaniem nowych książek do systemu.

```
namespace DataLibrary.Handlers
{
    public class AddBorrowerHandler : IRequestHandler<AddBorrower,
        Borrowers>
    {
        private readonly IBorrowersRepository _borrowersRepository;

        public AddBorrowerHandler(IBorrowersRepository
            borrowersRepository)
        {
            _borrowersRepository = borrowersRepository;
        }
        public async Task<Borrowers> Handle(AddBorrower request,
            CancellationToken cancellationToken)
        {
            var result = _borrowersRepository.AddBorrowers(request.
                borrower);
            return result;
        }
    }
}
```

AddBorrowerHandler.cs w Handlers. Klasa AddBorrowerHandler w przestrzeni nazw DataLibrary.Handlers to również implementacja wzorca projektowego Mediator za pomocą biblioteki MediatR, podobnie jak poprzednio opisana klasa AddBookHandler. Jest to obsługiwacz (handler) żądania AddBorrower, który odpowiada za przetwarzanie tego żądania. Służy do dodawania nowych wypożyczających do systemu.

```
namespace LibrarySystem.Controllers
{
```

```
3       public class AuthorsController : ControllerBase
4       {
5           private readonly IMediator _mediator;
6           public AuthorsController(IMediator mediator)
7           {
8               _mediator = mediator;
9           }
10          [HttpGet("/GetAuthors")]
11          public async Task<ActionResult> GetAuthors()
12          {
13              var result = await _mediator.Send(new GetAuthors());
14              return Ok(result);
15          }
16
17      }
18  }
```

AuthorsController.cs w Controllers. AuthorsController w aplikacji ASP.NET Core służy do obsługi żądań HTTP związanych z autorami. Używa wzorca Mediator za pośrednictwem MediatR. W tym przypadku, kontroler obsługuje żądanie GET do pobrania listy autorów.

```
1  namespace LibrarySystem.Controllers
2  {
3
4    public class BookAccessController : ControllerBase
5    {
6      private readonly IMediator _mediator;
7
8      public BookAccessController(IMediator mediator)
9      {
10       _mediator = mediator;
11     }
12
13     [HttpPost("/lock")]
14     public async Task<ActionResult<bool>> LockBook([FromBody]
           LockBookCommand command)
15     {
16       var result = await _mediator.Send(command);
17       return Ok(result);
18     }
19
20     [HttpPost("/release")]
21     public async Task<ActionResult> ReleaseBook([FromBody]
           ReleaseBookLock command)
22     {
23       await _mediator.Send(command);
24       return NoContent();
25     }
26   }
27 }
```

BookAccessController.cs w Controllers. BookAccessController w aplikacji ASP.NET Core służy do obsługi żądań HTTP związanych z blokowaniem i zwalnianiem dostępu do książek. Używa wzorca Mediator przez MediatR. Kontroler ten obsługuje konkretne komendy

dotyczące zarządzania dostępem do książek w bibliotece.

0.2 Angular

Angular złożony jest componentów, serwisów klas oraz intefejsów Komponenty mogą być widziane jako osobne strony http wraz z kodem.

```
<div class="login-containter">
  <div class="login-card">
    <h3 class="login-title">WITAJ PONOWNIE</h3>
    <form [formGroup]="LoginForm" (ngSubmit)="onSubmit()" class="login-
       form">
      <label for="registry_no" class="login-label">
        LOGIN
      </label>
      <div class="field-container">
        <input class="login-input" placeholder="login" id="registry_no"
           type="text" formControlName="registry_no">
      </div>
      <label for="password" class="login-label">
        HASLO
      </label>
      <div class="field-container">
        <input class="login-input" placeholder="password" id="password"
           matInput [type]="hide ? 'password' : 'text'" formControlName=
           "password" >
      </div>
      <div class="field-container">
        <button class="login-button" type="submit" [disabled]="!
           LoginForm.valid" >ZALOGUJ SIE</button>
      </div>
      <div class="login-error" *ngIf="error">WYSTAPIL BLAD LOGOWANIA</
         div>
    </form>
  </div>
  </div>
```

```

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  hide = true;
  LoginForm = new FormGroup({
    registry_no: new FormControl('',Validators.required),
    password: new FormControl('')
   })
   error:boolean = false;
  constructor(private service: DatabaseAccessServiceService,private
     route: ActivatedRoute,private router: Router, private authService:
     AuthenticateService,  private formBuilder: FormBuilder) { }
  onSubmit(): void {
```

```
16      const userlogin = this.LoginForm.value.registry_no as string;
17      const userPassword = CryptoJS.SHA256(this.LoginForm.value.password
            as string).toString(CryptoJS.enc.Hex);
18      this.service.GetStaffId(userlogin, userPassword).subscribe(resp =>{
19        if (resp != 0) {
20          const userId = resp
21          this.authService.succesfulLogIn(userId)
22          this.router.navigate(['/main']);
23        }
24        else {
25          this.error=true;
26        }
27
28    }
29
30      );
31    }
32    ngOnInit() {
33    }
34
35  }
```

Przykładowo to jest strona logowanie . Funkcja ngOnInit odbywa się w momęcie włączenia componentu. Funkcja onSubmit jest aktywowana poprzez kliknięcie na element strony i jeśli podane dane zgadzają się z danymi z bazy danych (this.service.GetStaffId(userlogin, userPassword)) Pozwala na routi czyli przejście do innego komponentu

Serwisy to elementy niezależne od componentów które działają od momentu inicializacji, mogą na przykład służyć do przetrzywania danych poniższy serwis służył do blokowanie zasobó za pomocą algorytmu elekcji poprzez przesyłanie zapytań do innych aktywnych klientów czy używają zasobu który chcę teraz użyc

```
1   startConnection(): void {
2     this.hubConnection = new HubConnectionBuilder()
3       .withUrl('https://localhost:7255/chatHub')
4       .build();
5
6     this.hubConnection
7       .start()
8       .then(() => {
9         console.log('SignalR connection started.');
10        this.getConnectionId();
11        this.setupEventHandlers();
12      })
13      .catch((error) => {
14        console.error('Error starting SignalR connection:', error);
15      });
16  }
17
18  private getConnectionId(): void {
19    this.hubConnection
20      .invoke('GetConnectionId')
21      .then((connectionId: string) => {
22        this.connectionId = connectionId;
23        console.log('Connection ID:', connectionId);
24      })
25      .catch((err) => console.error('Error getting connection ID:', err)
          );
26  }
27
28  private setupEventHandlers(): void {
29    this.hubConnection.on('RequestReceived', (userId: string, bookId:
        string) => {
30
31      this.sendResponse(userId, bookId);
32    });
33
34  }
35
36
37  sendRequest(bookId: string): Promise<boolean> {
38    this.bookid=bookId
39    return new Promise<boolean>(async (resolve) => {
40      const trySendRequest = async () => {
41        if (this.hubConnection) {
42          if (this.hubConnection.state === 'Disconnected') {
43            await this.startConnection();
44          } else if (this.hubConnection.state === 'Connected') {
45            const hasMatchingBookId = await this.invokeRequest(bookId);
46            this.responseResult = hasMatchingBookId;
47            resolve(this.responseResult);
48          } else if (this.hubConnection.state === 'Connecting') {
```

```
            setTimeout(trySendRequest, 100);
          }
        } else {
          console.error('SignalR connection is undefined.');
          resolve(false);
        }
      };

      trySendRequest();
    });
  }

private dewid(){
  console.log("tempsadasdasd")
}

private async _startConnection(): Promise<void> {
  try {
    await this.hubConnection.start();
    console.log('SignalR connection started.');
    this.getConnectionId();
    this.setupEventHandlers();
  } catch (error) {
    console.error('Error starting SignalR connection:', error);
  }
}

private invokeRequest(bookId: string): Promise<boolean> {
  return new Promise<boolean>((resolve) => {
    let hasMatchingBookId = false;

    const responseHandler = (bookIds: string) => {
      console.log(bookId)
      console.log(bookIds)
      if (bookId == bookIds) {
        hasMatchingBookId = true;
        this.responseResult = true;
      }
      console.log("Here" + this.responseResult);
      resolve(hasMatchingBookId);
    };

    this.hubConnection.on('ResponseReceived', responseHandler);

    this.hubConnection
      .invoke('SendRequest', this.connectionId, bookId)
      .catch((error) => {
        console.error('Error sending request:', error);

        resolve(false);
      });
  });
}
```

```
104
105
106
107
108
109   sendResponse(userId: string, bookId: string): void {
110     if (this.hubConnection && this.hubConnection.state === 'Connected')
          {
111       this.hubConnection.invoke('SendResponse', userId, this.bookid)
112         .catch((error) => {
113           console.error('Error sending response:', error);
114         });
115     } else {
116       console.error('SignalR connection is not in the Connected state.')
            ;
117     }
118   }
119 }
```

W aplikacji istneje jeszcze jeden ważny serwis database acces który pozwala na komunikacją z c# backendem

```
1  export class DatabaseAccessServiceService {
2      readonly databaseAccesssURL = "https://localhost:7255";
3  constructor(private http:HttpClient) { }
4  lockBook(bookId: number, userId: number): Observable<boolean> {
5    console.log(this.http.post<boolean>(this.databaseAccesssURL + `/lock`,
         { bookId, userId }));
6    return this.http.post<boolean>(this.databaseAccesssURL + `/lock`, {
       bookId, userId });
7  }
8
9  releaseBook(bookId: number): Observable<void> {
10   const url = `${this.databaseAccesssURL}/release`;
11   return this.http.post<void>(url, { bookId });
12 }
13 GetAuthors(): Observable<AuthorsInterface[]>
14 {
15   return this.http.get<any>(this.databaseAccesssURL + `/GetAuthors`);
16 }
17 GetBooksByAuthors(authorId:number):Observable<BooksClass[]>
18 {
19   return this.http.get<BooksClass[]>(this.databaseAccesssURL + `/
       GetBookByAuthors` + `?authorId=${authorId}`);
20 }
21 GetStaffId(login: string, password: string): Observable<number>
22 {
23   return this.http.get<number>(this.databaseAccesssURL + `/GetStaffId`
       +`?login=${login}` + `&password=${password}`);
24 }
25 GetAllBookss():Observable<BooksClass[]>
26 {
27   return this.http.get<BooksClass[]>(this.databaseAccesssURL + `/
       GetAllBooks` );
28 }
```

```
29 GetBookById(bookId:number):Observable<BooksClass>
30 {
31   return this.http.get<BooksClass>(this.databaseAccesssURL + `/
       GetBookById`+`?bookId=${bookId}` );
32 }
```