System katalogów bibliotecznych

Projekt baz danych

Autor sprawozdania: Michał Dziedziak 263901, Michał Zapała 263935
Imię i Nazwisko prowadzącego kurs: Dr inż. Marcin Łopuszyński
Dzień i godzina zajęć: Wtorek, 17:05 - 18:45

Spis treści

Spis tabel

Spis rysunków

1 Opis wycinka rzeczywistości

1.1 Opis rzeczywistości

Biblioteka jest instytucją kultury, która gromadzi, przechowuje i udostępnia materiały biblioteczne oraz informuje o nich. Każdy obywatel może wypożyczyć dany zasób biblioteczny na określony okres. Jednocześnie ma możliwość podglądu dostępnych zasobów w bibliotece oraz informacji na ile dni może je wypożyczyć. Każdy zasób biblioteczny określany jest przez kilka cech:

- typ (książka, artykuł, list, kaseta),
- tytuł,
- autor,
- rok wydania
- numer egzemplarza danego tytułu,
- status (czy jest wypożyczony i przez kogo),
- informację o liczbie dni, na które można wypożyczyć egzemplarz,
- ilość dostępnych sztuk na stanie.

1.2 Zasoby ludzkie

- Pracownik biblioteki:
  - wypożyczanie pozycji klientowi
  - dodawanie pozycji
  - usuwanie pozycji
  - katalogowanie pozycji
  - śledzenie/modyfikowanie stanu (ilość) zasobów bibliotecznych
  - sprawdzanie statusu danego egzemplarza (czy jest wypożyczony i przez kogo)
- Klient (czytelnik)
  - sprawdzanie statusu wypożyczonych przez siebie egzemplarzy (na jaki okres dany egzemplarz został mu wypożyczony),
  - możliwość przedłużenia okresu wypożyczenia
  - sprawdzenie dostępności pozycji
  - sprawdzenie informacji o pozycji (tytuł, autor, rok wydania)

1.3 Opisy sytuacji

Sytuacja 1:

Do biblioteki przychodzi osoba, która chce wypożyczyć dany zasób. Prosi pracownika o możliwość wypożyczenia danego materiału. Pracownik sprawdza dostępność danego tytułu oraz informacje o liczbie dni na jaki może zostać wypożyczony. Pracownik przekazuje te informacje klientowi. Jeżeli klient decyduje się na wypożyczenie dostępnego zasobu, wypożyczony egzemplarz nie jest dostępny dla kolejnych klientów, dopóki nie wróci on z powrotem do biblioteki. Jednak kolejny klient może wypożyczyć inny egzemplarz tego samego zasobu.

Sytuacja 2:

Klient zdalnie przegląda dostępne materiały biblioteczne i wypożycza egzemplarz wybranego zasobu. Klient nie widzi egzemplarzy wypożyczonych przez inne osoby.

Sytuacja 3:

Do biblioteki zostaje dostarczony nowy zasób. Pracownik ma zadanie dodać nowy materiał do spisu biblioteki.

Sytuacja 4:

Jedyny egzemplarz danego zasobu biblioteki zostaje zniszczony. Pracownik ma za zadanie usunąć go ze spisu dostępnych materiałów w przypadku, gdy zasób danego tytułu nie będzie uzupełniony.

Sytuacja 5:

Klient zwraca wypożyczony materiał biblioteczny. Pracownik ma za zadanie zmienić status danego egzemplarza. W następstwie dany egzemplarz jest znowu widoczny dla nowych klientów.

2 Model bazy danych

2.1 Wymagania funkcjonalne i niefunkcjonalne

1. Wymagania funkcjonalne
   (a) Użytkownik zakłada konto
   (b) Użytkownik loguję się
   (c) Użytkownik przegląda zasoby
   (d) Użytkownik sprawdza dostępność danego zasobu
   (e) Użytkownik sprawdza informację o zasobie
   (f) Pracownik dodaje zasoby
   (g) Pracownik usuwa zasoby
   (h) Pracownik zarządza zasobami
   (i) Pracownik zmienia ilość dostępnych zasobów
   (j) Pracownik sprawdza status danego egzemplarza
   (k) Pracownik zatwierdza lub odrzuca prośbę o wypożyczenie egzemplarza
   (l) Pracownik zatwierdza lub odrzuca prośbę o przedłużenie terminu wypożyczenia egzemplarza
   (m) Pracownik zarządza egzemplarzami zasobów
   (n) Pracownik odnotowuje zwrot wypożyczonego egzemplarza
   (o) Klient przegląda wypożyczone przez niego zasoby
   (p) Klient zgłasza prośbę o wypożyczenie egzemplarza zasobu
   (q) Klient zgłasza prośbę o przedłużenie terminu wypożyczenia
2. Wymagania niefunkcjonalne
   (a) Aktualizacja zasobów jest rejestrowana w bazie danych
   (b) System jest przyjazny i prosty w obsłudze
   (c) System zapewnia bezpieczeństwo danych użytkownika

(d) System chroni dane klientów

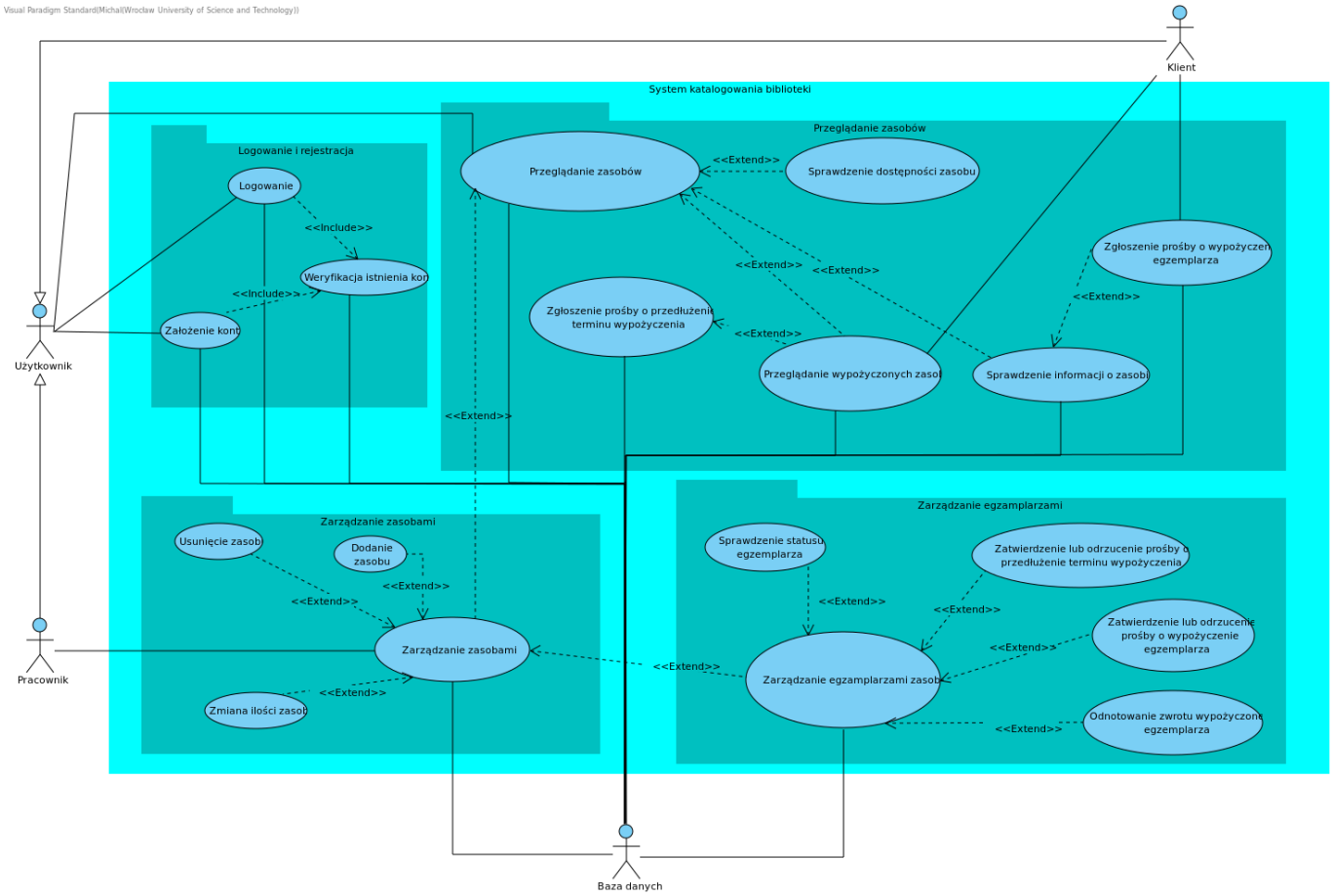
(e) System informuje klienta o odmowie wypożyczenia egzemplarza

(f) Klienci zgłaszają dowolną ilość próśb o wypożyczenie

(g) Użytkownicy mają różne uprawnienia

(h) System jest uruchamiany na popularnych systemach

2.2 Diagram przypadków użycia



Rysunek 1: Diagram przypadków użycia

2.3 Opis scenariuszy

1. **Weryfikacja istnienia kont**
   a. **Cel:** Zweryfikowanie czy konto użytkownika istnieje.
   b. **Aktor:** Baza danych
   c. **Zdarzenie inicjujące:** Wpływa polecenie o weryfikację istnienia konta.
   d. **Warunki wstępne:**
      1) Może być wywołana z Pu Założenie konta lub z Pu Logowanie.
   e. **Warunki końcowe:**
      1) Zwracana jest informacja o pomyślnym zweryfikowany.
      2) Lub: Zwracana jest informacja o niepowodzeniu weryfikacji.
   f. **Scenariusz:**
      1) Wpływa polecenie o weryfikację istnienia konta.
      2) Jeśli konto o danym loginie istnieje w Bazie danych:
         2.1) Zwracana jest informacja o powodzeniu operacji.
      3) W przeciwnym wypadku:
         3.1) Zwracana jest informacja o niepowodzeniu operacji.
      4) Następuje zakończenie scenariusza.

2. **Logowanie**
   a. **Cel:** Zalogowanie użytkownika na jego konto w aplikacji.
   b. **Aktor:**
      1) Użytkownik – osoba korzystająca z systemu.
      2) Baza danych
   c. **Zdarzenie inicjujące:** Użytkownik otwiera panel logowania.
   d. **Warunki wstępne:**
      1) Inicjalizacja poprzez uruchomienie aplikacji.
   e. **Warunki końcowe:**
      1) Użytkownik zostaje zalogowany.
      2) Lub: Użytkownik zaniechuje próby logowania.
   f. **Scenariusz:**
      1) Dopóki użytkownik chce się zalogować
         1.1) Użytkownik podaje login i hasło.
         1.2) Następuje wywołanie PU weryfikacja istnienia konta w celu sprawdzenia, czy podane konto istnieje.
         1.3) Jeżeli weryfikacja konta zakończy się sukcesem:
            1.3.1) Baza danych zwraca informacje o haśle przypisanym do danego loginu.
               1.3.1.1) Jeśli hasło jest poprawne:
                  1.3.1.1.1) Użytkownik zostaje zalogowany.
                  1.3.1.1.2) Następuje zakończenie scenariusza.
               1.3.1.2) W przeciwnym przypadku:
                  1.3.1.2.1) Przejście do kroku 1.4.1
         1.4) W przeciwny wypadku:
            1.4.1) Zwracany jest komunikat o błędnych danych.
            1.4.2) Przejście do kroku 1
      2) Następuje zakończenie scenariusza

3. **Założenie konta przez użytkownika**
   a. **Cel:** Założenie konta przez użytkownika.
   b. **Aktor:**
      1) Użytkownik – osoba korzystająca z systemu.
      2) Baza danych
   c. **Zdarzenie inicjujące:** Użytkownik otwiera panel rejestracji.
   d. **Warunki wstępne:**
      1) Inicjalizacja poprzez uruchomienie aplikacji.
   e. **Warunki końcowe:**
      1) Użytkownik zakłada swoje konto.
      2) Lub: Użytkownik zaniechuje próby założenia konta
   f. **Scenariusz:**
      1) Dopóki użytkownik chce założyć konto
         1.1) Użytkownik podaje login, hasło i jeżeli chce klucz uprawnień pracownika.
         1.2) Następuje wywołanie PU weryfikacja istnienia konta w celu sprawdzenia, czy podane konto istnieje.
         1.3) Jeżeli weryfikacja konta zakończy się sukcesem:
            1.3.1) Zwracany jest komunikat, że dane konto już istnieje.
         1.4) W przeciwny wypadku:
            1.4.1) Jeśli użytkownik podał poprawny klucz uprawnień pracownika:
               1.4.1.1) Stworzone zostaje nowe konto pracownika.
               1.4.1.2) Stworzenie nowego konta pracownika zostaje odnotowane w bazie danych.
               1.4.1.3) Następuje zakończenie scenariusza.
            1.4.2) W przeciwnym wypadku:
               1.4.2.1) Stworzone zostaje nowe konto klienta.
               1.4.2.2) Stworzenie nowego konta klienta zostaje odnotowane w bazie danych.
               1.4.2.3) Następuje zakończenie scenariusza.
      2) Następuje zakończenie scenariusza.

4. **Zarządzanie zasobami**
   a. **Cel:** Zarządzanie zasobami przez pracownika.
   b. **Aktor:**
      1) Pracownik – osoba odpowiedzialna za zasoby w bibliotece
      2) Baza danych
   c. **Zdarzenie inicjujące:** Pracownik wybrał zakładkę zarządzania zasobami
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Przeglądanie zasobów przez pracownika
   e. **Warunki końcowe:**
      1) Dokonane zmiany zostają odnotowane w Bazie danych
   f. **Scenariusz:**
      1) Dopóki pracownik chce zarządzać zasobami
         1.1) Pracownik chce zarządzać zasobami.
         1.2) Jeśli Pracownik chce dodać zasób:
            1.2.1) Przypadek Dodanie zasobu zostaje wywołany
         1.3) Jeśli Pracownik chce usunąć zasób:
            1.3.1) Przypadek Usunięcie zasobu zostaje wywołany
         1.4) Jeśli Pracownik chce zmienić ilość zasobu:
            1.4.1) Przypadek Zmiana ilości zasobu zostaje wywołany
         1.5) Jeśli Pracownik chce zarządzać danym egzemplarzem:
            1.5.1) Przypadek Zarządzanie egzemplarzami zasobu zostaje wywołany
         1.6) Zmiany zostają odnotowane w bazie danych
      2) Następuje zakończenie scenariusza

5. **Dodanie zasobu**
   a. **Cel:** Dodanie nowego zasobu do spisu biblioteki.
   b. **Aktor:** Pracownik
   c. **Zdarzenie inicjujące**: Pracownik chce dodać nowy zasób
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie zasobami
   e. **Warunki końcowe:**
      1) Zasób zostaje dodany.
   f. **Scenariusz:**
      1) Pracownik uzupełnia dane nowego zasobu.
      2) Pracownik potwierdza, że chce dodać zasób.
      3) Następuje zakończenie scenariusza.

6. **Usunięcie zasobu**
   a. **Cel:** Usunięcie zasobu ze spisu biblioteki.
   b. **Aktor:** Pracownik
   c. **Zdarzenie inicjujące**: Pracownik chce usunąć zasób.
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie zasobami
   e. **Warunki końcowe:**
      1) Zasób zostaje usunięty.
   f. **Scenariusz:**
      1) Pracownik wybiera zasób do usunięcia.
      2) Pracownik potwierdza, że chce usunąć zasób.
      3) Następuje zakończenie scenariusza

7. **Zmiana ilości zasobu**
   a. **Cel:** Zmiana ilości zasobu w spisie biblioteki.
   b. **Aktor:** Pracownik
   c. **Zdarzenie inicjujące**: Pracownik chce zmienić dostępność ilość danego zasobu.
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie zasobami
   e. **Warunki końcowe:**
      1) Ilość zasobu zostaje zmieniona.
   f. **Scenariusz:**
      1) Pracownik wybiera zasób do zmiany ilość
      2) Pracownik wpisuję nową ilość zasobu
      3) Pracownik potwierdza, że chce zmienić ilość zasobu
      4) Następuje zakończenie scenariusza

8. **Zarządzanie egzemplarzami zasobu**
   a. **Cel:** Zarządzanie poszczególnymi egzemplarzami danego zasobu.
   b. **Aktor:**
      1) Pracownik
      2) Baza danych
   c. **Zdarzenie inicjujące:** Pracownik chce zarządzić egzemplarzem
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie zasobami
   e. **Warunki końcowe:**
      1) Dokonane zmiany egzemplarzy zostają odnotowane w Bazie danych
   f. **Scenariusz:**
      1) Baza danych zwraca informację o egzemplarzach danego zasobu.
      2) Dopóki pracownik chce zarządzać egzemplarzami
         2.1) Pracownik chce zarządzać egzemplarzami.
         2.2) Jeśli Pracownik chce sprawdzić status egzemplarza:
            2.2.1) Pracownik wybiera egzemplarz z listy
            2.2.2) Przypadek Sprawdzenie statusu egzemplarza zostaje wywołany
         2.3) Jeśli Pracownik chce zatwierdzić lub odrzucić prośbę o wypożyczenie egzemplarza:
            2.3.1) Pracownik wybiera egzemplarz z listy
            2.3.2) Przypadek Zatwierdzenie lub odrzucenie prośby o wypożyczenie egzemplarza zostaje wywołany
         2.4) Pracownik chce zatwierdzić lub odrzucić prośbę o przedłużenie terminu wypożyczenia egzemplarza
            2.4.1) Pracownik wybiera egzemplarz z listy
            2.4.2) Zostaje wywołany przypadek Zatwierdzenie lub odrzucenie prośby o przedłużenie terminu wypożyczenia
         2.5) Jeśli Pracownik chce odnotować zwrot wypożyczonego egzemplarza:
            2.5.1) Pracownik wybiera egzemplarz z listy
            2.5.2) Zostaje wywołany przypadek Odnotowanie zwrotu wypożyczonego egzemplarza
      3) Zmiany zostają zapisane w bazie danych.
      4) Następuje zakończenie scenariusza.

9. **Sprawdzenie statusu egzemplarza**
   a. **Cel:** Sprawdzenie statusu egzemplarza w spisie biblioteki.
   b. **Aktor:** Pracownik
   c. **Zdarzenie inicjujące**: Pracownik chce sprawdzić stan danego egzemplarza.
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie egzemplarzami zasobu
   e. **Warunki końcowe:**
      1) Komunikat o statusie zasobu.
   f. **Scenariusz:**
      1) Baza danych zwraca informację o statusie egzemplarza
      2) Informację zostają wyświetlone
      3) Następuje zakończenie scenariusza

10. **Zatwierdzenie lub odrzucenie prośby o wypożyczanie egzemplarza**
   a. **Cel:** Zatwierdzenie lub odrzucenie prośby o wypożyczenie egzemplarza danego zasobu.
   b. **Aktor:** Pracownik
   c. **Zdarzenie inicjujące**: Pracownik chce zatwierdzić bądź odrzucić prośbę o wypożyczenie.
   d. **Warunki wstępne:**
      1) Może być wywołany z PU Zarządzanie egzemplarzami zasobu
   e. **Warunki końcowe:**
      1) Zaakceptowanie prośby o wypożyczenie.
      2) LUB: Odrzucenie prośby o wypożyczenie.
   f. **Scenariusz:**
      1) Jeśli Pracownik chce zatwierdzić prośbę o wypożyczenie:
         1.1) Pracownik zatwierdza prośbę.
         1.2) Następuje zakończenie scenariusza.
      2) W przeciwnym przypadku:
         2.1) Pracownik odrzuca prośbę.
         2.2) Następuje zakończenie scenariusza.

11. **Zatwierdzenie lub odrzucenie prośby o przedłużenie terminu wypożyczenia**
    a. **Cel:** Zatwierdzenie lub odrzucenie prośby o zatwierdzenia bądź odrzucenia prośby o przedłużenie terminu wypożyczenia.
    b. **Aktor:** Pracownik
    c. **Zdarzenie inicjujące**: Pracownik chce zatwierdzić bądź odrzucić prośbę o przedłużenie terminu wypożyczenia.
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Zarządzanie egzemplarzami zasobu
    e. **Warunki końcowe:**
        1) Zaakceptowanie prośby o przedłużenie wypożyczenia.
        2) LUB: Odrzucenie prośby o przedłużenie wypożyczenia.
    f. **Scenariusz:**
        1) Jeśli Pracownik chce zatwierdzić prośbę o przedłużenie wypożyczenia:
            1.1) Pracownik zatwierdza prośbę.
            1.2) Następuje zakończenie scenariusza.
        2) W przeciwnym przypadku:
            2.1) Pracownik odrzuca prośbę.
            2.2) Następuje zakończenie scenariusza.

12. **Odnotowanie zwrotu wypożyczonego egzemplarza**
    a. **Cel:** Odnotowanie zwrotu konkretnego egzemplarza danego zasobu.
    b. **Aktor:** Pracownik
    c. **Zdarzenie inicjujące**: Klient zwraca dany egzemplarz zasobu
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Zarządzanie egzemplarzami zasobu
    e. **Warunki końcowe:**
        1) Odnotowanie zwrotu.
    f. **Scenariusz:**
        1) Pracownik potwierdza zwrot danego egzemplarza.
        2) Następuje zakończenie scenariusza.

13. **Przeglądanie zasobów**
    a. **Cel:** Przeglądanie zasobów biblioteki
    b. **Aktor:**
        1) Użytkownik
        2) Baza danych
    c. **Zdarzenie inicjujące**: Użytkownik chce przeglądać zasoby biblioteki
    d. **Warunki wstępne:**
        1) Inicjalizacja poprzez uruchomienie okna przeglądania zasobów bibliotecznych
    e. **Warunki końcowe:**
        1) Użytkownik skończył przeglądać zasoby
    f. **Scenariusz:**
        1) Baza danych zwraca informację o zasobach
        2) Informację zostają wyświetlone
        3) Dopóki Użytkownik przegląda zasoby
            3.1) Jeżeli Użytkownik ma uprawnienia Pracownika i użytkownik wybrał zakładkę zarządzania zasobami
                3.1.1) Przypadek Zarządzanie zasobami zostaje wywołany
            3.2) W przeciwnym przypadku
                3.2.1) Przypadek Sprawdzenie informacji o zasobie zostaje wywołany
            3.3) Jeżeli Użytkownik wpisał nazwę zasobu do okna szukania
                3.3.1) Zostaje wywołany przypadek Sprawdzenie dostępności zasobu
            3.4) Jeżeli Użytkownik wybrał akcje "Przeglądanie Wypożyczonych Zasobów" i Użytkownik ma uprawnienia Klient
                3.4.1) Zostaje wywołany przypadek Przeglądanie wypożyczonych zasobów
        4) Następuje zakończenie scenariusza.

14. **Sprawdzenie dostępności zasobu**
    a. **Cel:** Sprawdzenie, czy dany zasób jest dostępny w bibliotece
    b. **Aktor:** Klient
    c. **Zdarzenie inicjujące**: Wpisane nazwy szukanego zasobu w pole wyszukiwania
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Przeglądanie zasobów
    e. **Warunki końcowe:**
        1) Wyświetlenie szukanego zasobu na samej górze listy
        2) LUB: Wyświetlenie informacji o tym, że dany zasób nie istnieje
    f. **Scenariusz:**
        1) Jeżeli Zasób o danej nazwie istnieje
            1.1) Zostaję wyświetlony na samej górze listy zasobów
        2) W przeciwnym przypadku
            2.1) Zostaje wyświetlona informacja o tym, że dany zasób nie istnieje
        3) Następuje zakończenie scenariusza.

15. **Zgłoszenie prośby o przedłużenie terminu wypożyczenia**
    a. **Cel:** Klient chce przedłużyć termin wypożyczenia egzemplarza
    b. **Aktor:**
        1) Klient
        2) Baza danych
    c. **Zdarzenie inicjujące**: Zgłoszenie prośby o przedłużenie wypożyczenia przez Klienta z poziomu aplikacji
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Przeglądanie wypożyczonych zasobów
    e. **Warunki końcowe:**
        1) Zgłoszenie zostaje odnotowane w bazie danych
    f. **Scenariusz:**
        1) Użytkownik wybiera o ile chce przedłużyć termin wypożyczenia (tydzień, 2 tygodnie, 3 tygodnie)
        2) Baza danych odnotowuje zgłoszenie prośby
        3) Następuje zakończenie scenariusza

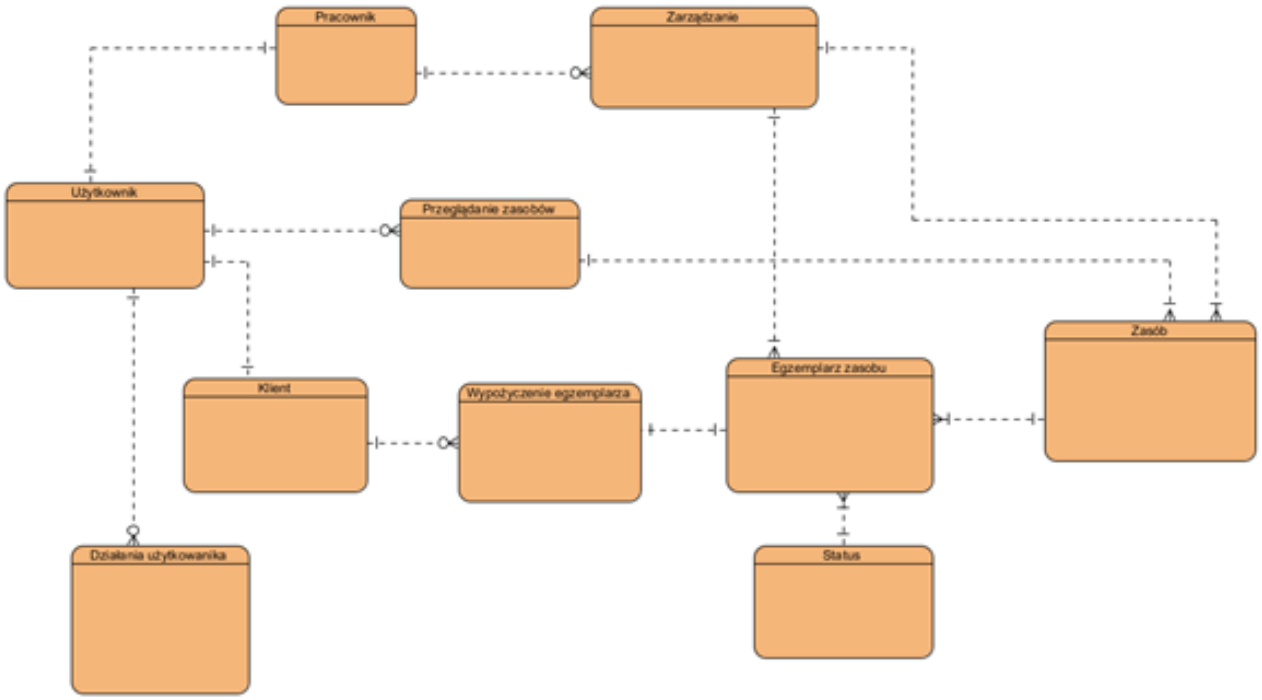
15. **Zgłoszenie prośby o przedłużenie terminu wypożyczenia**
    a. **Cel:** Klient chce przedłużyć termin wypożyczenia egzemplarza
    b. **Aktor:**
        1) Klient
        2) Baza danych
    c. **Zdarzenie inicjujące**: Zgłoszenie prośby o przedłużenie wypożyczenia przez Klienta z poziomu aplikacji
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Przeglądanie wypożyczonych zasobów
    e. **Warunki końcowe:**
        1) Zgłoszenie zostaje odnotowane w bazie danych
    f. **Scenariusz:**
        1) Użytkownik wybiera o ile chce przedłużyć termin wypożyczenia (tydzień, 2 tygodnie, 3 tygodnie)
        2) Baza danych odnotowuje zgłoszenie prośby
        3) Następuje zakończenie scenariusza

16. **Przeglądanie wypożyczonych zasobów**
    a. **Cel:** Klient chce obejrzeć wypożyczone przez niego zasoby
    b. **Aktor:**
        1) Klient
        2) Baza danych
    c. **Zdarzenie inicjujące**: Klient wybiera opcje przeglądania wypożyczonych zasobów
    d. **Warunki wstępne:**
        1) Może być wywołany z PU Przeglądanie zasobów
    e. **Warunki końcowe:**
        1) Klient chce zakończyć przeglądanie wypożyczonych zasobów
    f. **Scenariusz:**
        1) Baza danych zwraca informację o zasobach wypożyczonych przez Klient
        2) Dopóki Klient chce przeglądać wypożyczone pozycje
            2.1) Jeżeli Klient wybierze akcję "Zgłoś prośbę o przedłużenie terminu"
                2.1.1) Wywołany zostaje przypadek Zgłoszenie prośby o przedłużenie terminu wypożyczenia
        3) Scenariusz zostaje zakończony

17. **Sprawdzenie informacji o zasobie**
    a. **Cel: Sprawdzenie szczegółowych informacji o zasobie**
    b. **Aktor:**
        1) Użytkownik
        2) Baza danych
    c. **Zdarzenie inicjujące**: Użytkownik wybrał zasób w celu sprawdzenia informacji o nim
    d. **Warunki wstępne:**
        1) Może być wywołane z PU Przeglądanie zasobów
    e. **Warunki końcowe:**
        1) Informacje o zasobie zostają wyświetlone
    f. **Scenariusz:**
        1) Baza danych zwraca informację o zasobie (typ, tytuł, autor, rok wydania, informacja o liczbie dni, na które można wypożyczyć egzemplarz, ilość dostępnych egzemplarzy)
        2) Informację zostają wyświetlone
        3) Jeżeli Użytkownik ma uprawnienia Klient i Użytkownik chce wypożyczyć zasób
           3.1)Zostaje wywołany przypadek Zgłoszenie prośby o wypożyczenie egzemplarza
        4) Scenariusz zostaje zakończony

18. **Zgłoszenie prośby o wypożyczenie egzemplarza**
    a. **Cel: Zgłoszenie prośby o wypożyczenie egzemplarza**
    b. **Aktor:**
        1) Klient
        2) Baza danych
    c. **Zdarzenie inicjujące**: Klient wybrał opcję zgłoszenia prośby o wypożyczenie egzemplarza
    d. **Warunki wstępne:**
        1) Może być wywołane z PU Sprawdzenie informacji o zasobie
    e. **Warunki końcowe:**
        1) Zgłoszenie prośby o wypożyczenie egzemplarza zostaje odnotowane w bazie danych
    f. **Scenariusz:**
        1) Użytkownik wybiera na ile chce wypożyczyć egzemplarz danego zasobu (tydzień, 2 tygodnie, 3 tygodnie)
        2) Baza danych odnotowuje zgłoszenie prośby
        3) Scenariusz zostaje zakończony

2.4 Diagram ERD



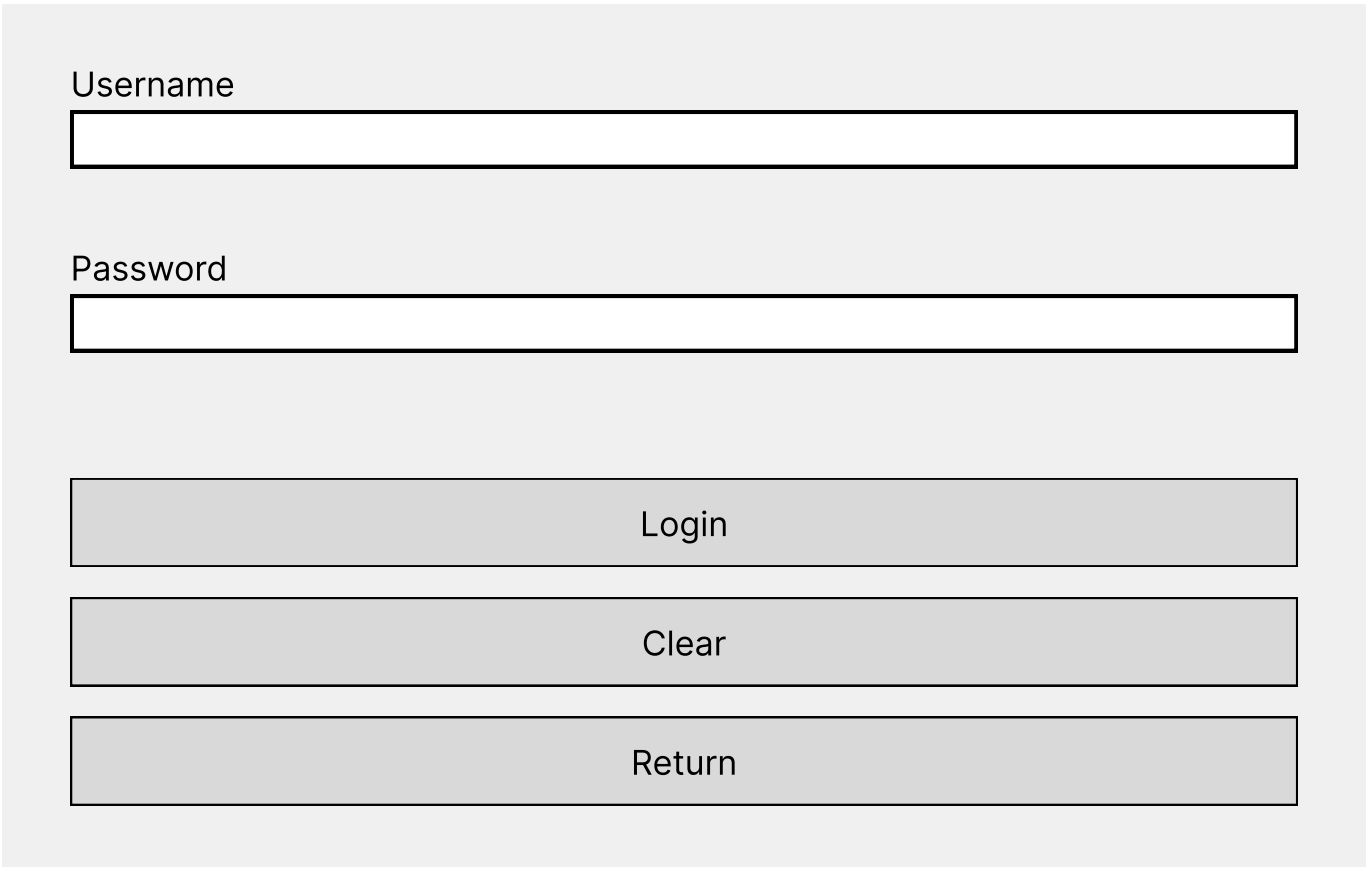
Rysunek 2: Diagram ERD

3 Implementacja bazy danych

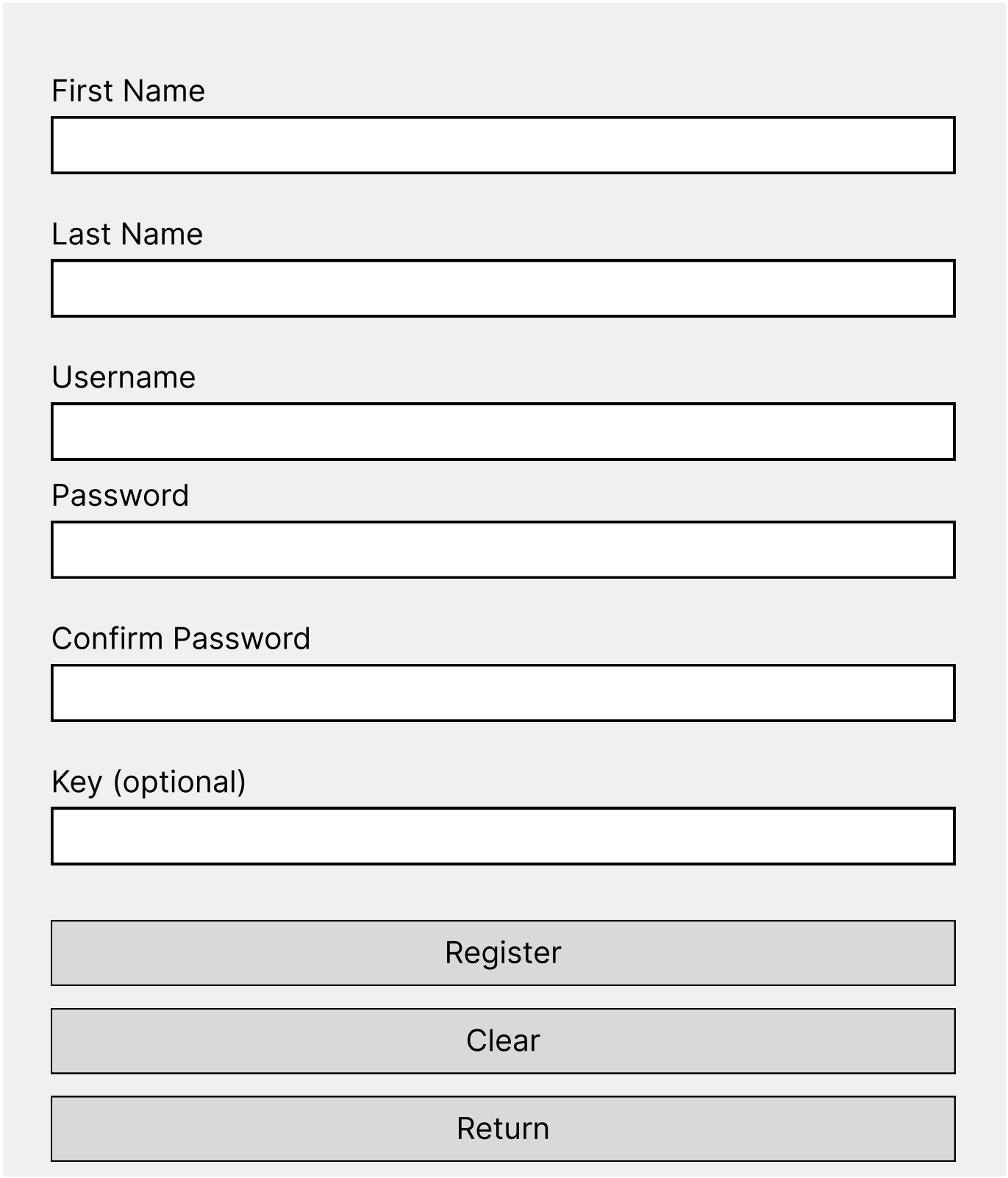
4 Implementacja aplikacji

4.1 Makieta interfejsu graficznego

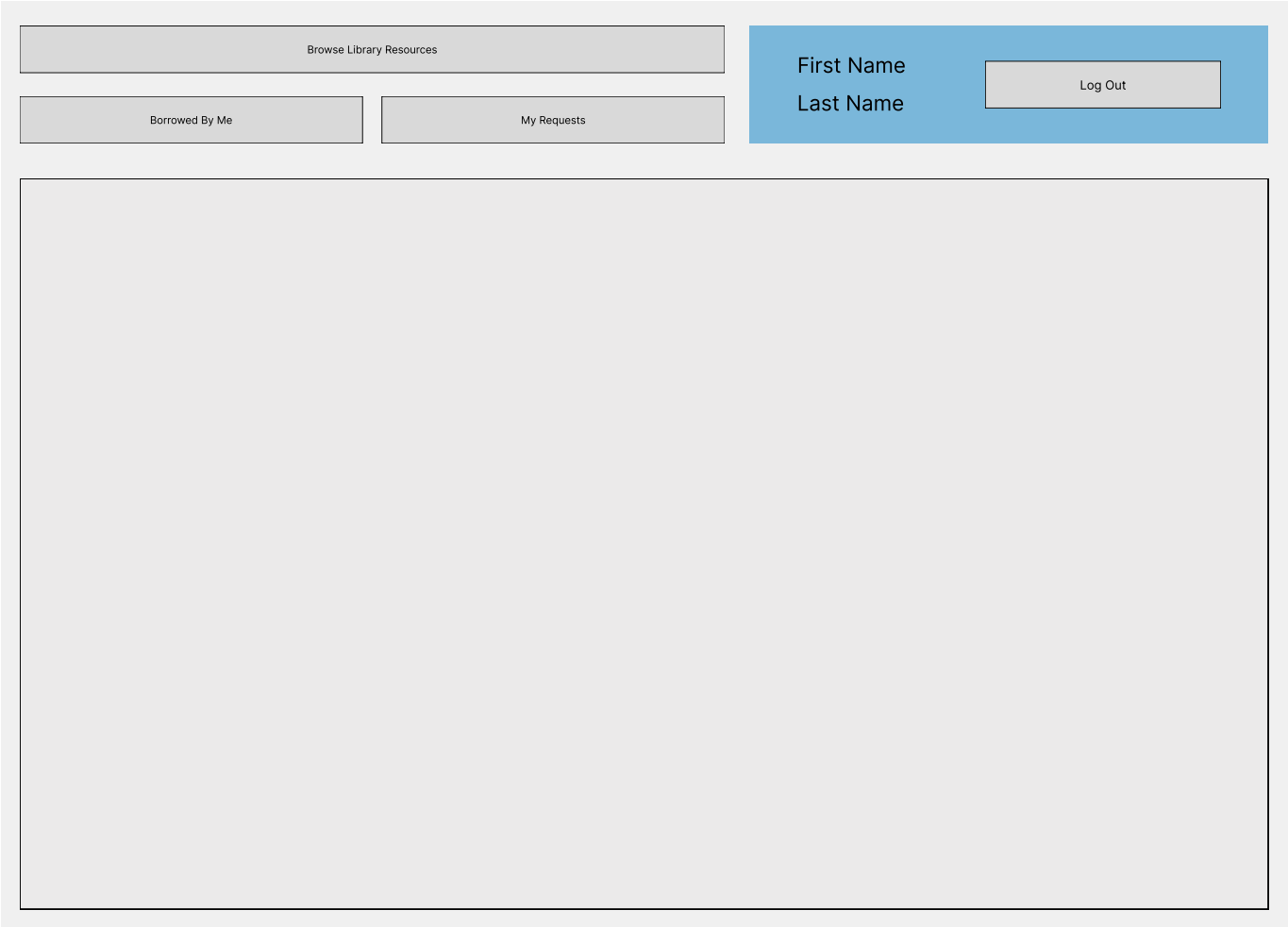
Rysunek 3: Makieta widoku powitalnego.



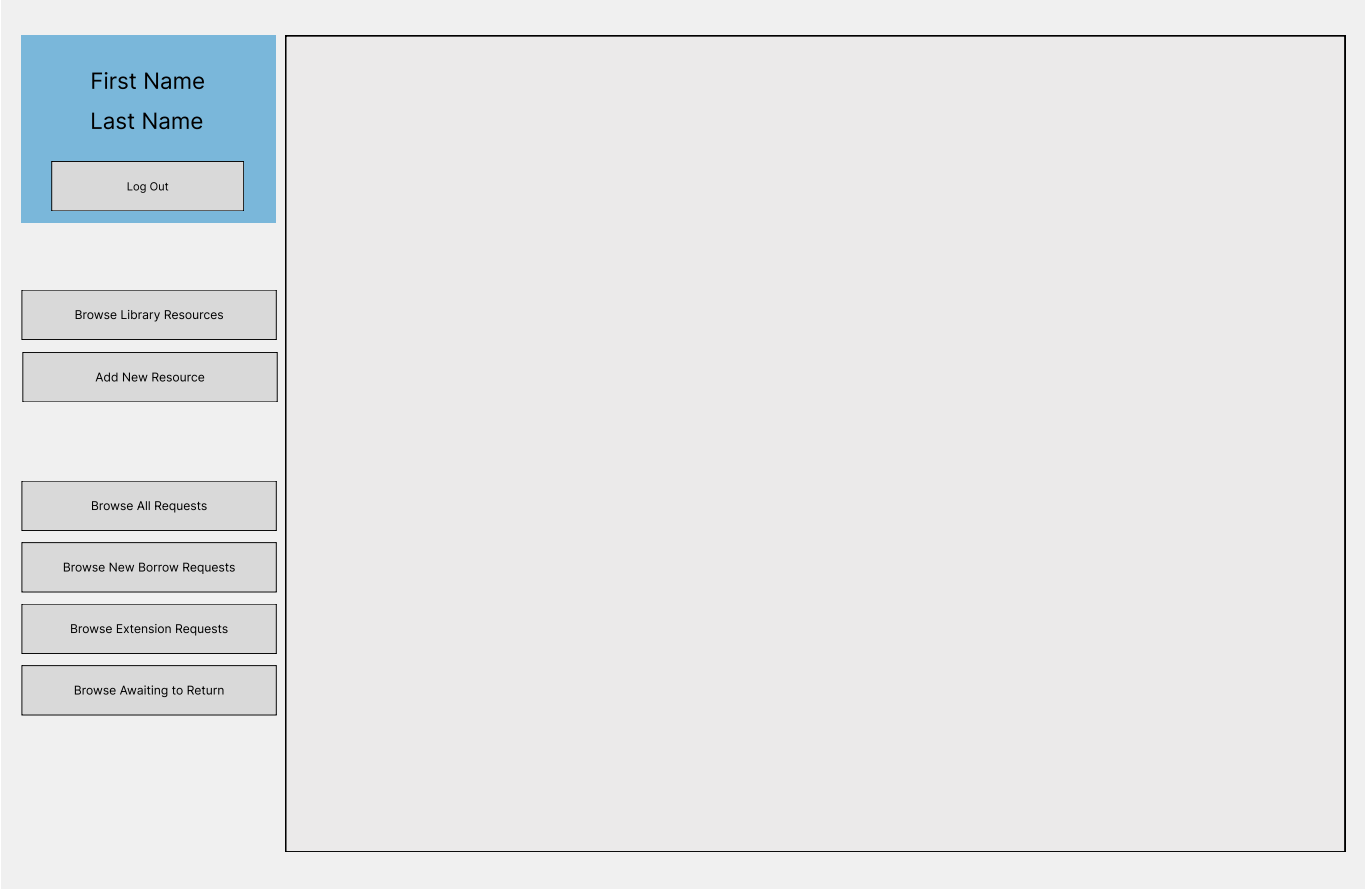
Rysunek 4: Makieta widoku logowania.



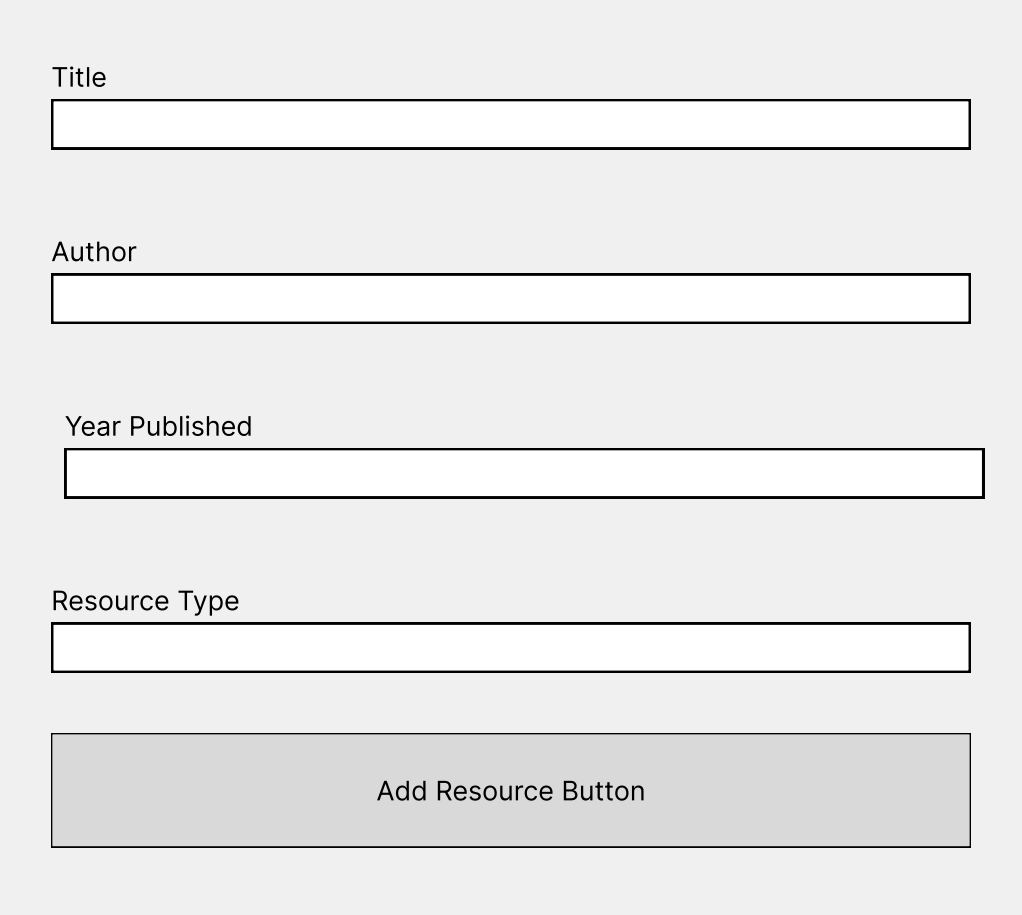
Rysunek 5: Makieta widoku rejestracji.



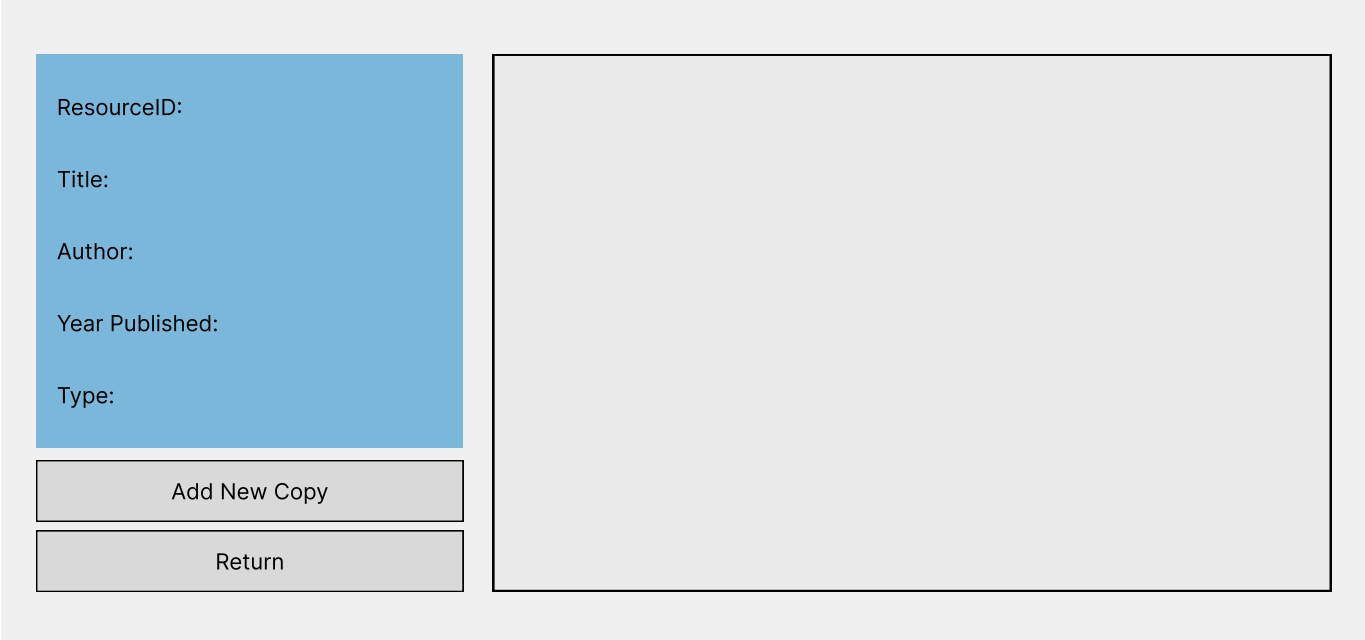
Rysunek 6: Makieta widoku klienta.



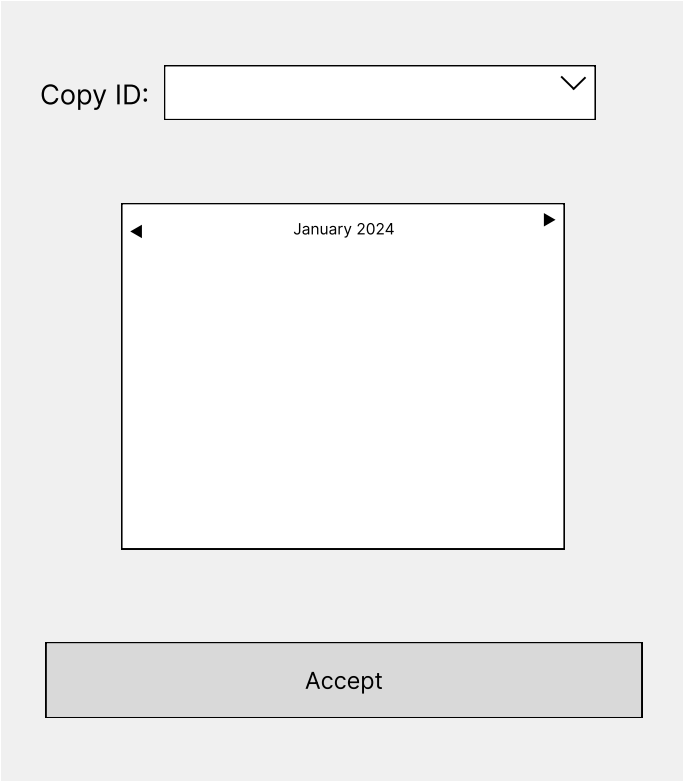
Rysunek 7: Makieta widoku pracownika.



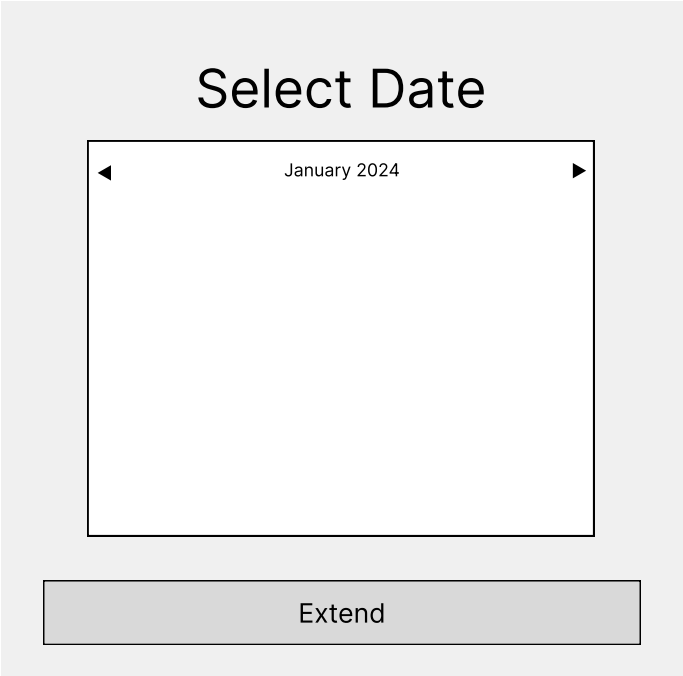
Rysunek 8: Makieta widoku dodawania zasobu.



Rysunek 9: Makieta widoku zarządzania kopiami zasobu.

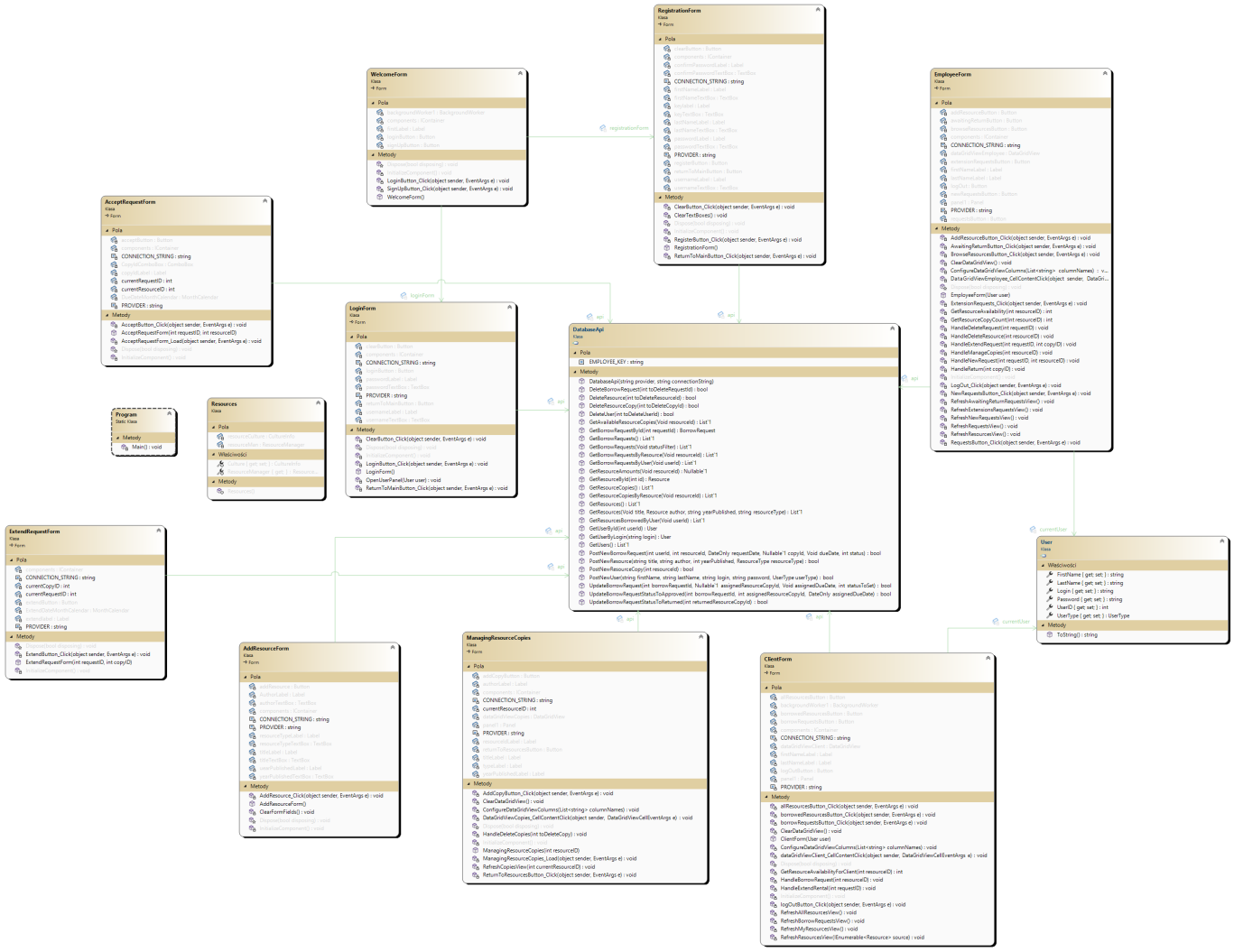


Rysunek 10: Makieta widoku akceptacji prośby o wypożyczenie.

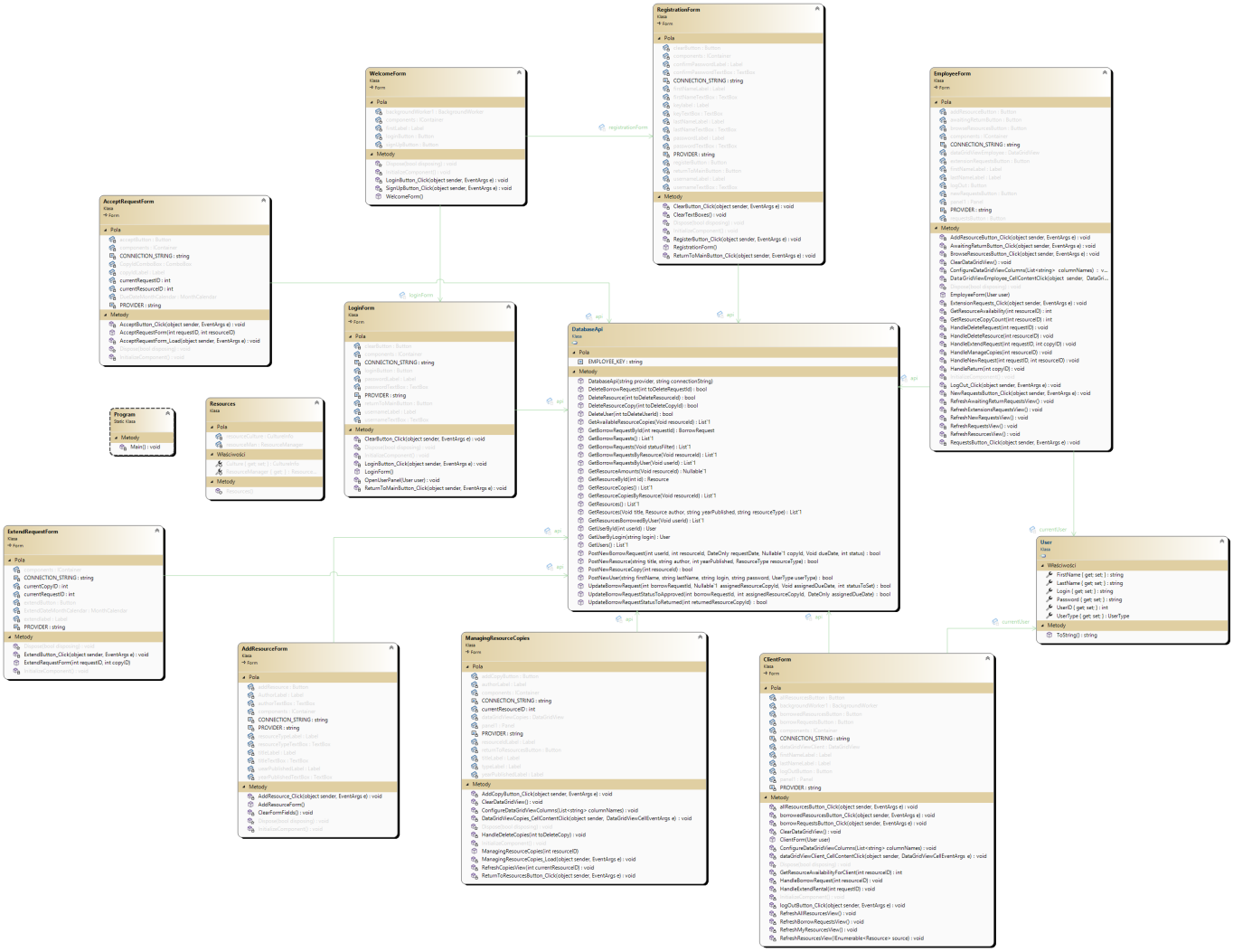


Rysunek 11: MAkieta widoku przedłużenia prośby o wypożyczenie.

4.2 Diagram klas



Rysunek 12: Diagram klas dla Front-End'u.



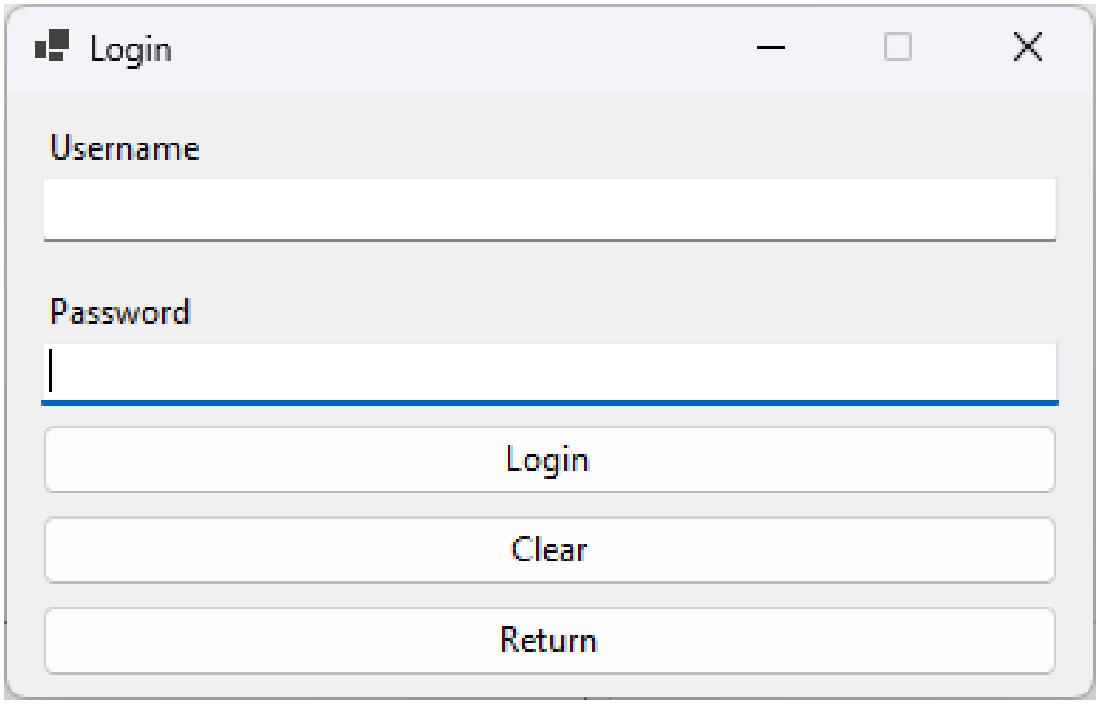
Rysunek 13: Diagram klas dla Back-End'u.

4.3 Wdrożenie aplikacji

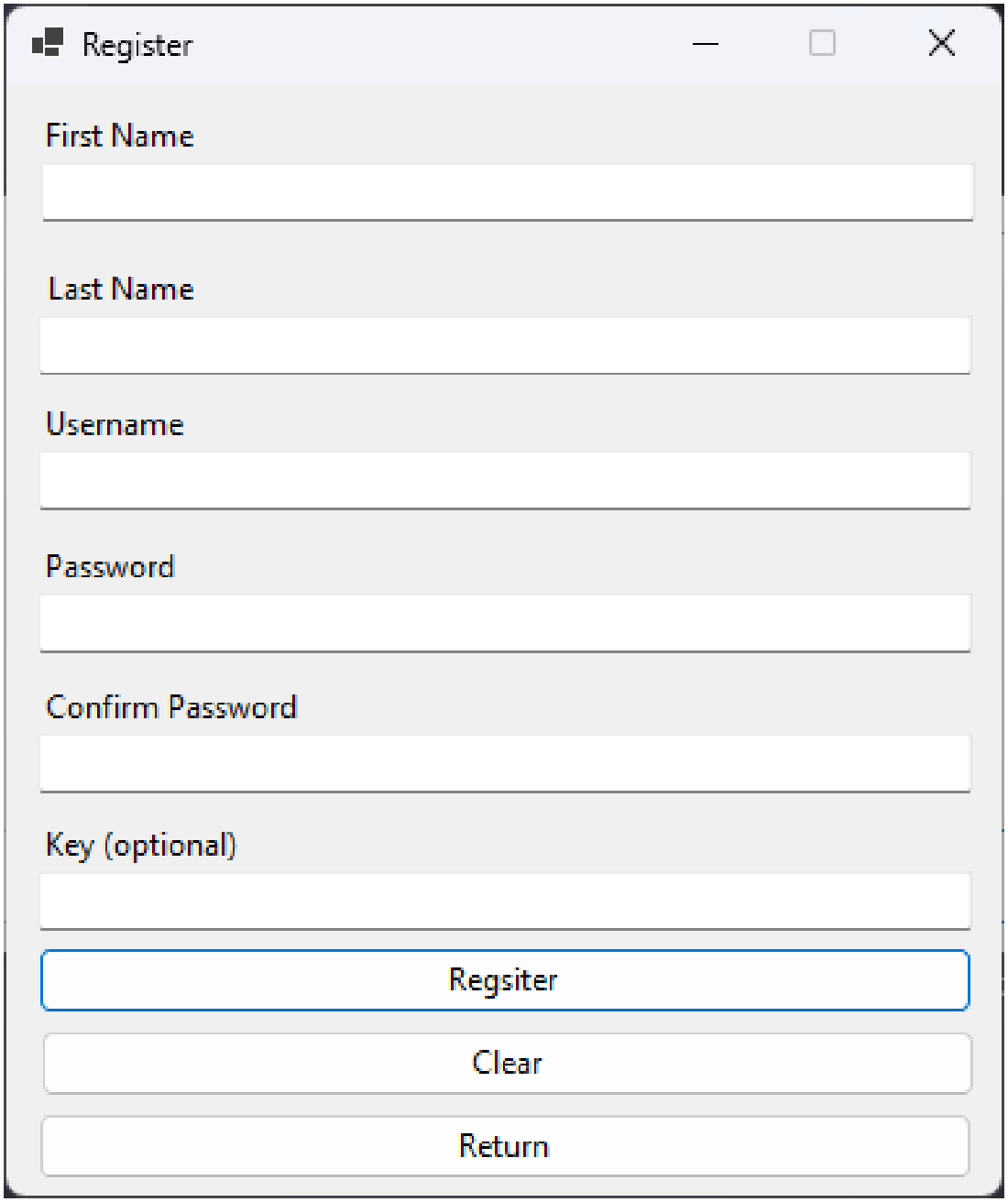
//screeny

4.4 Okna startowe

Rysunek 14: Okno powitalne.

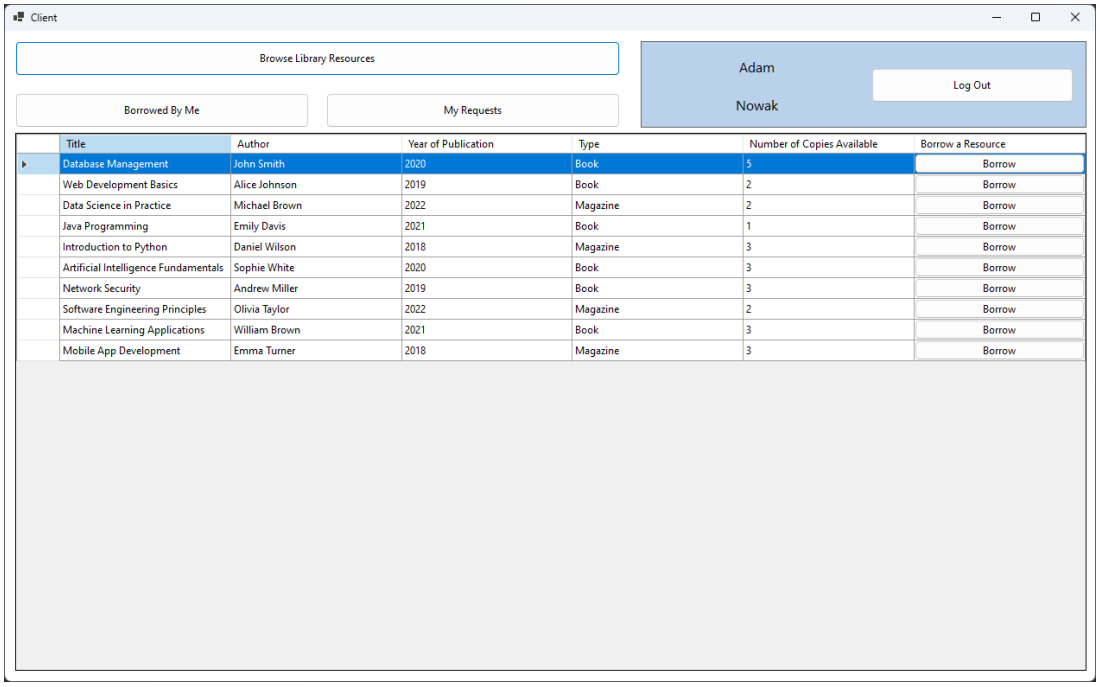


Rysunek 15: Okno logowania.

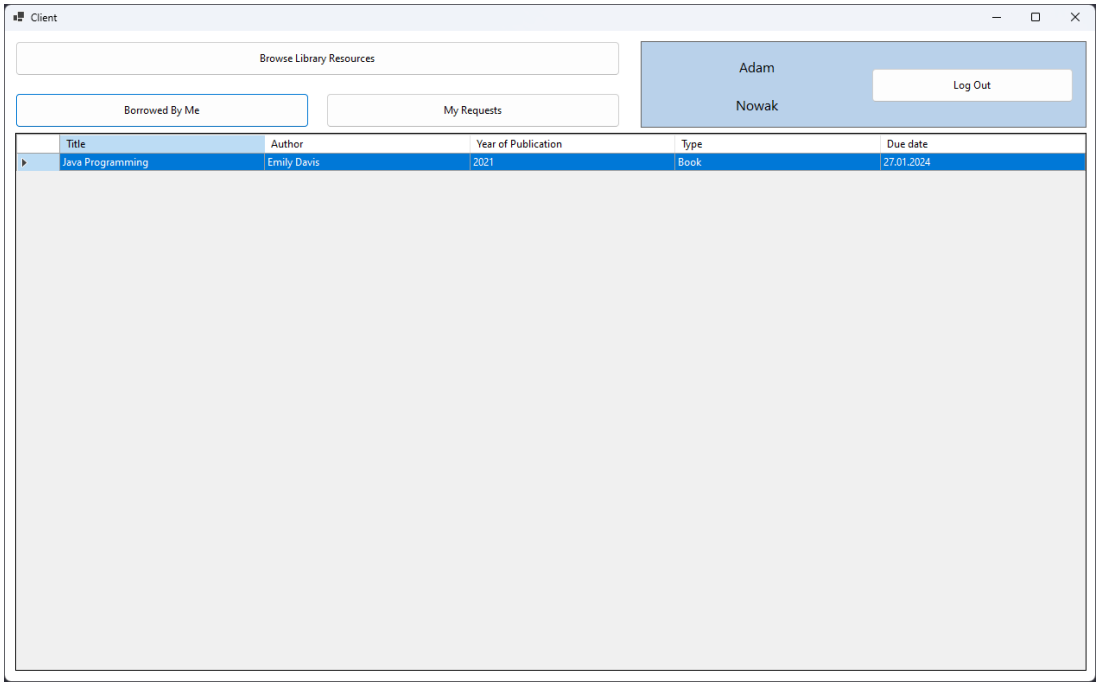


Rysunek 16: Okno rejestracji.

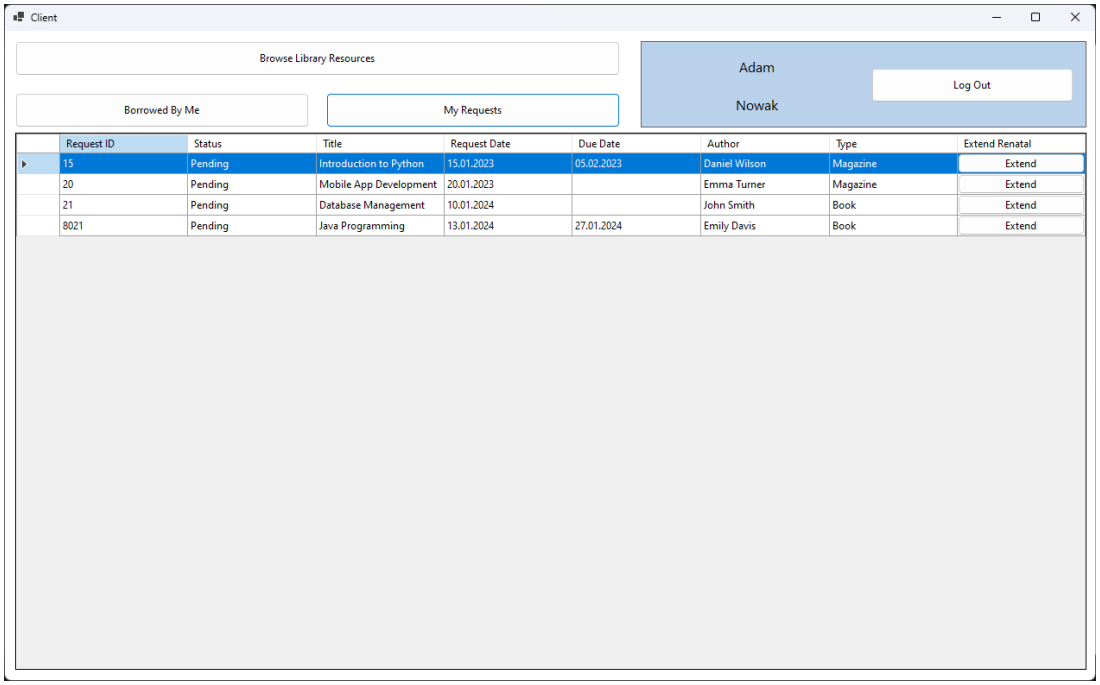
4.5 Okna klienta



Rysunek 17: Okno przeglądania zbioru biblioteki.

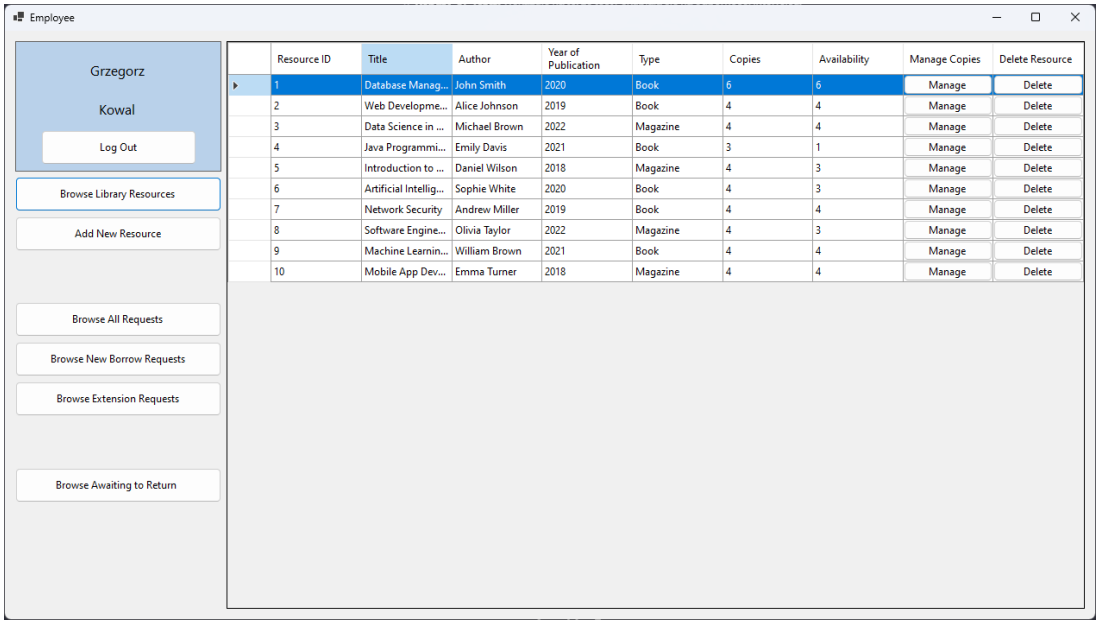


Rysunek 18: Okno przeglądania pożyczonych pozycji.

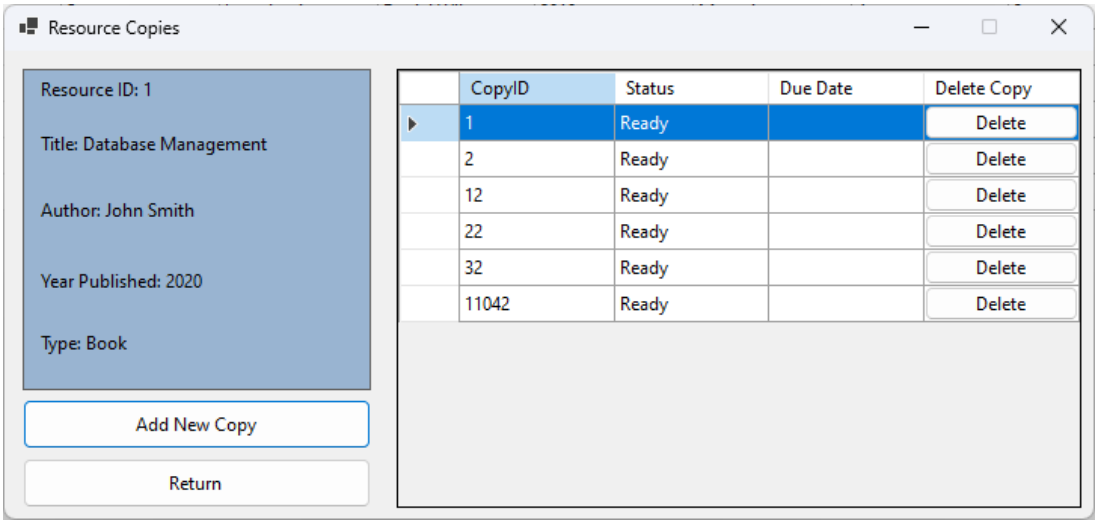


Rysunek 19: Okno przeglądania próśb o wypożyczenie.

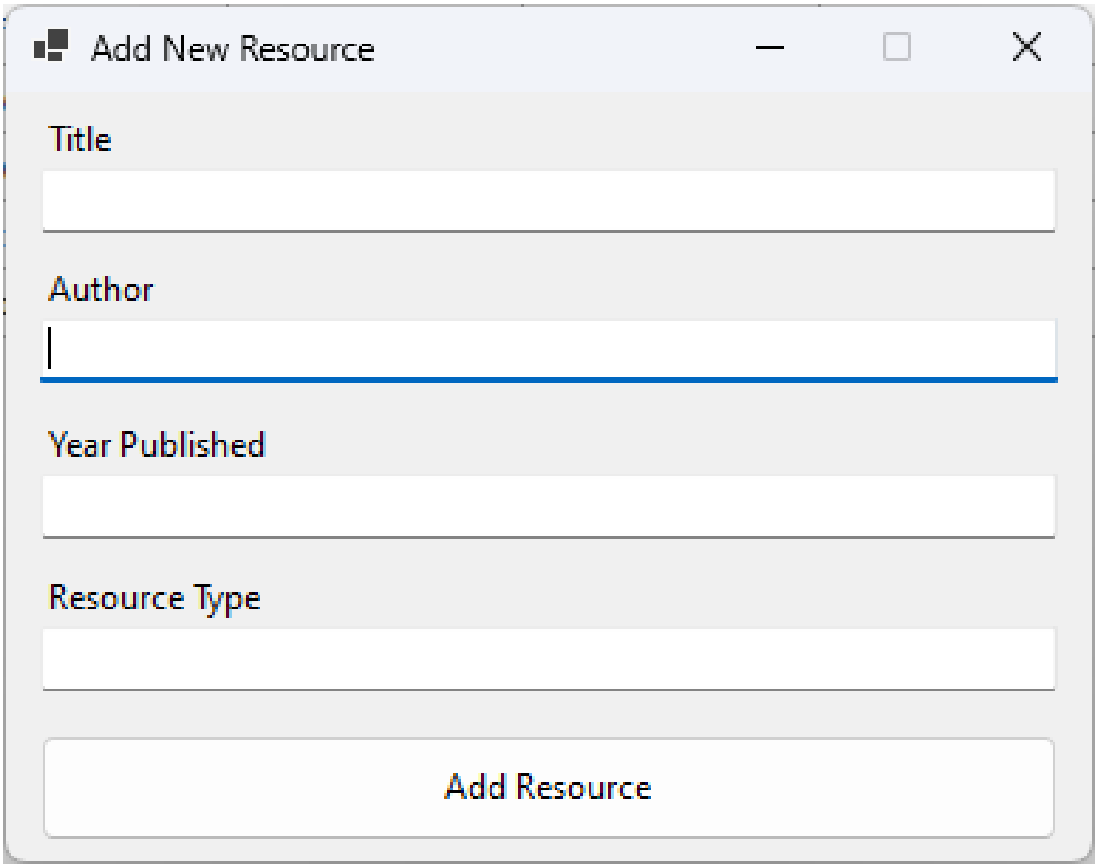
4.6 Okna pracownika



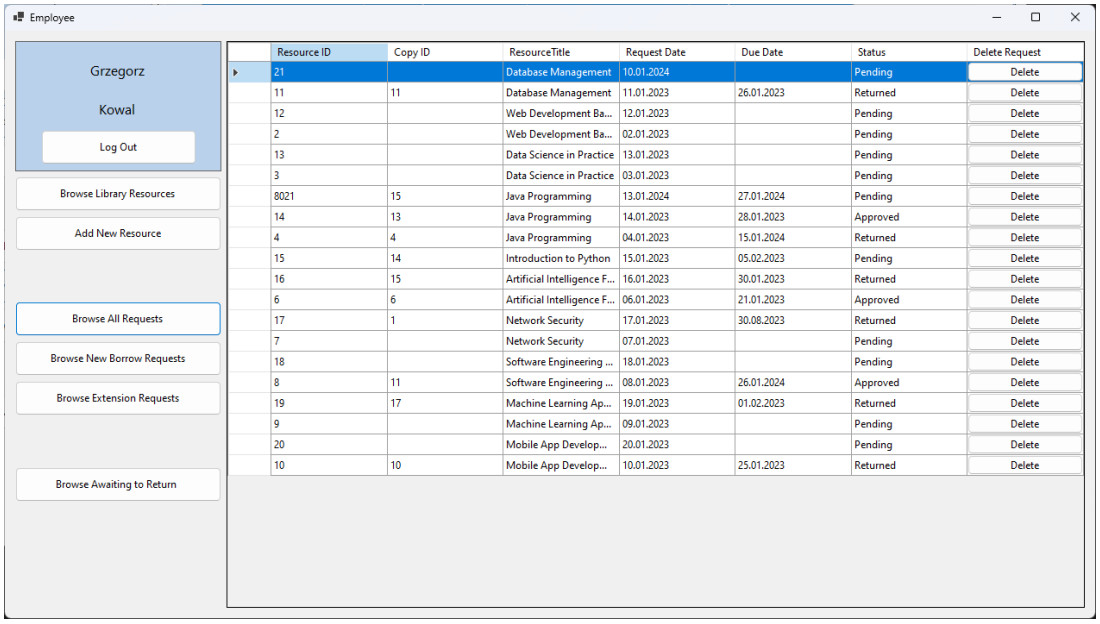
Rysunek 20: Okno przeglądania zasobów biblioteki.



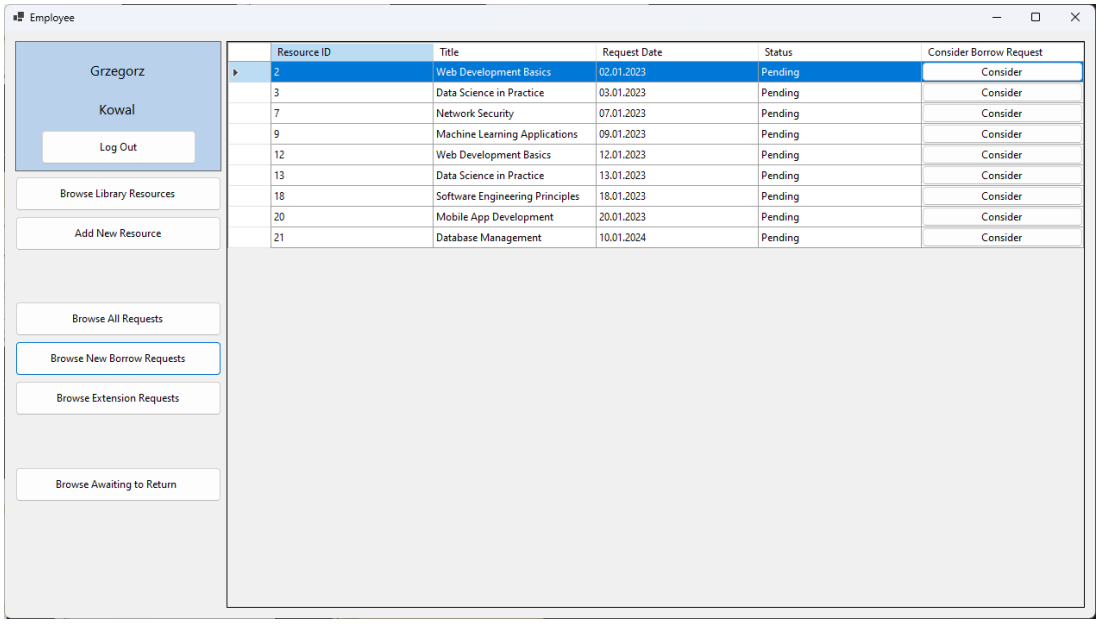
Rysunek 21: Okno zarządzania egzemplarzami zasobu.



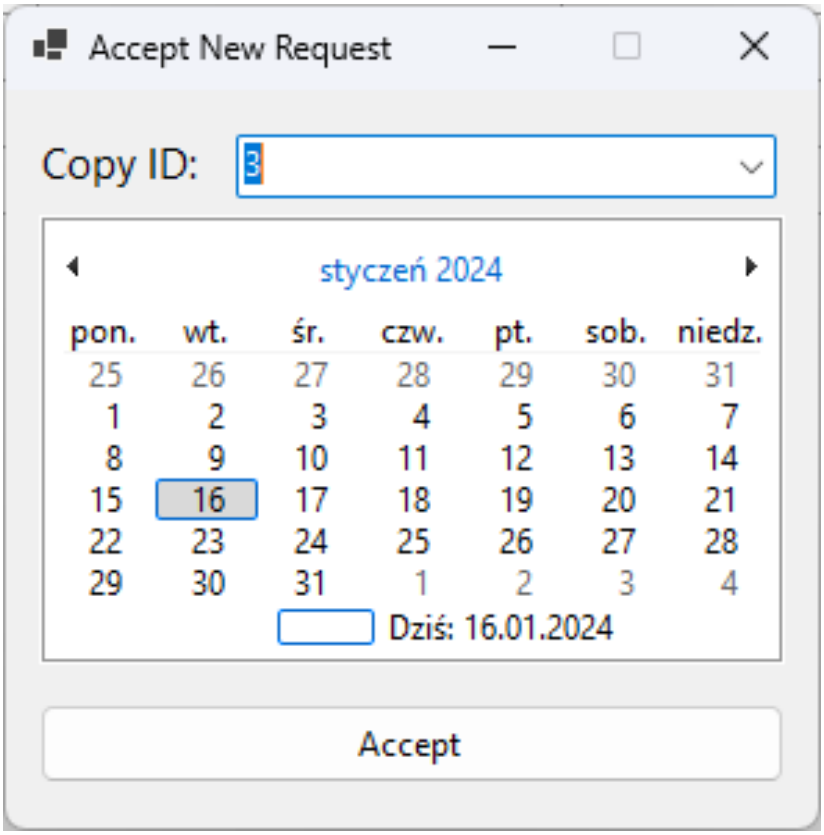
Rysunek 22: Okno dodawania nowego zasobu.



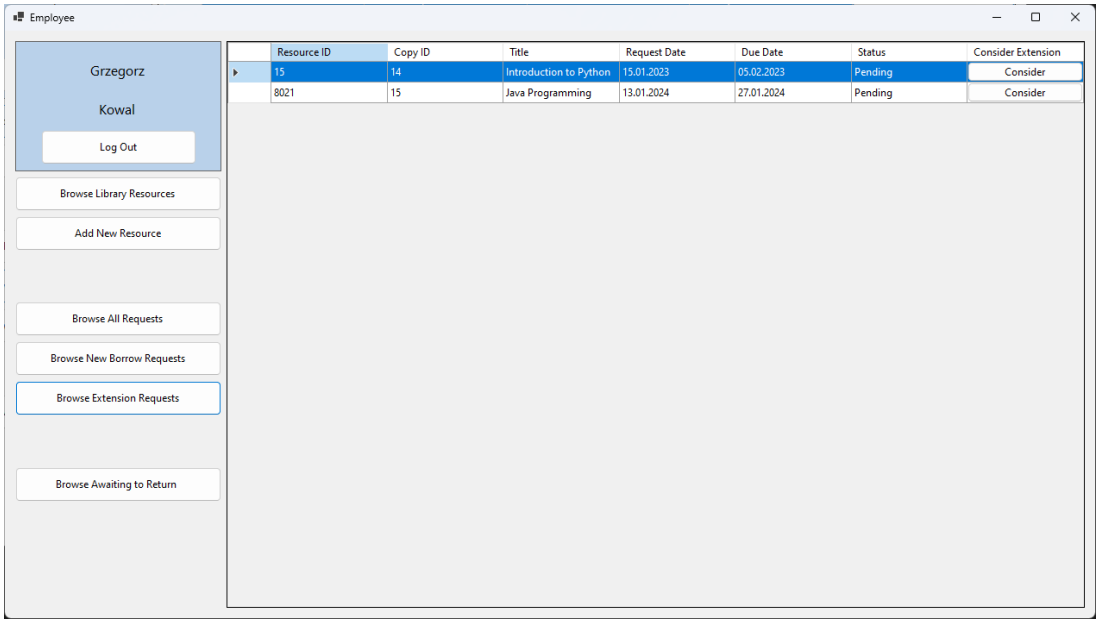
Rysunek 23: Okno przeglądania wszystkich próśb o wypożyczenie.



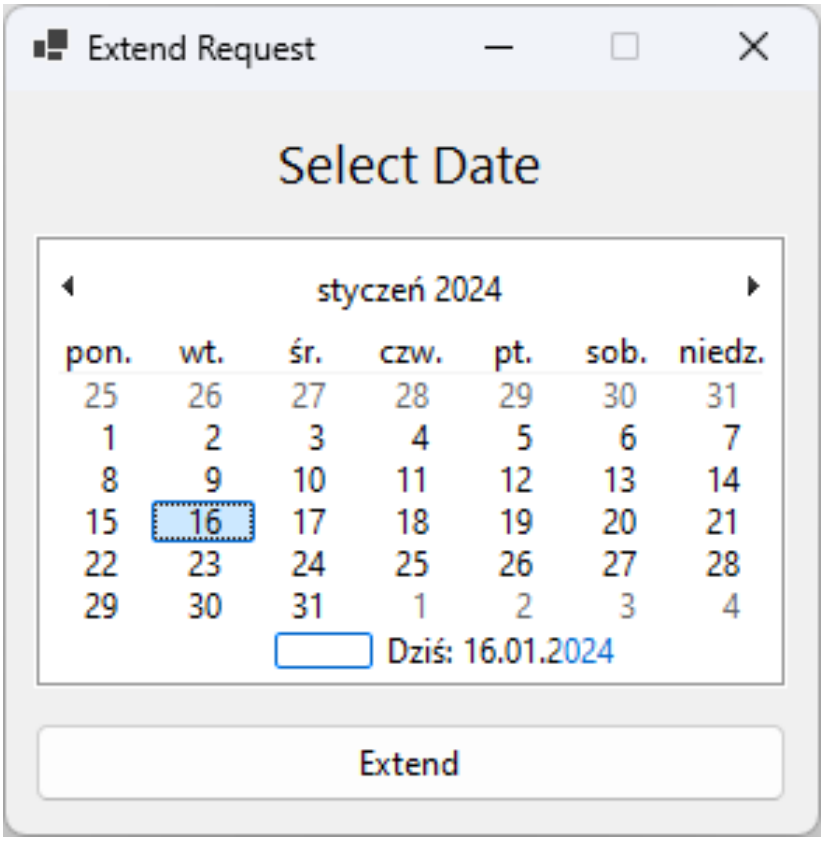
Rysunek 24: Okno przeglądania nowych próśb o wypożyczenie.



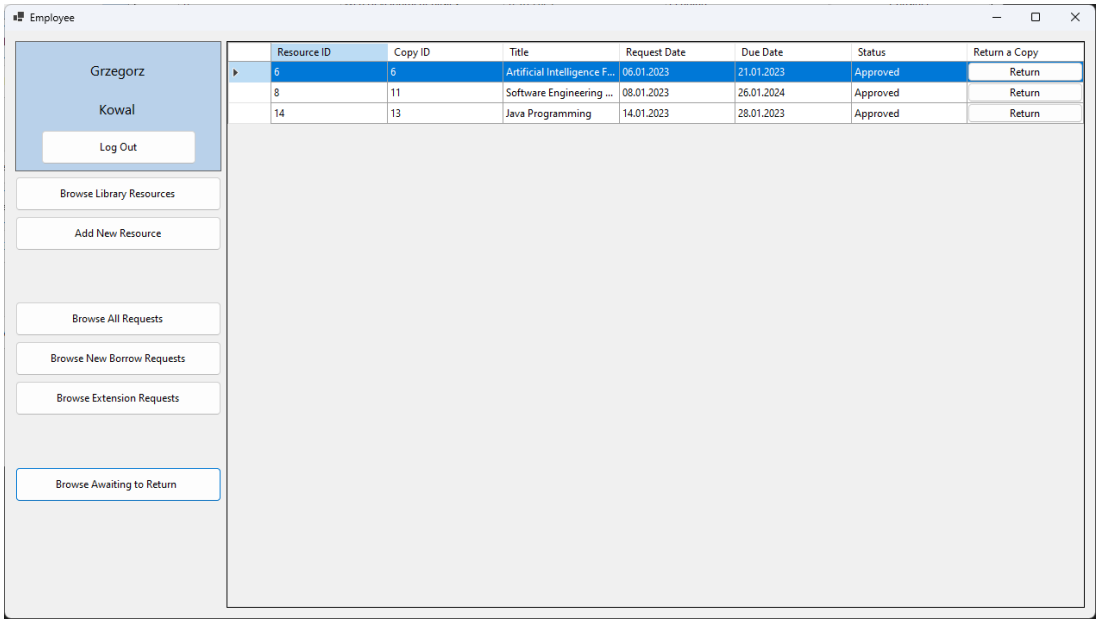
Rysunek 25: Okno akceptacji prośby o wypożyczenie.



Rysunek 26: Okno przeglądania próśb o przedłużenie wypożyczenia.



Rysunek 27: Okno akceptacji prośby o przedłużenie wypożyczenia.



Rysunek 28: Okno przeglądania wypożyczonych egzemplarzy (oczekujących na zwrot).

4.7 Testowanie aplikacji

4.7.1 Testy jednostkowe

```
    namespace LibraryDatabaseApiUnitTests;

public class Tests
{
    const string PROVIDER = ".NET Framework Data Provider for SQL Server";
    const string CONNECTION_STRING =
    "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=LibraryDataBase;Integrated Security=True";
    private DatabaseApi api = new DatabaseApi(PROVIDER, CONNECTION_STRING);

    [Fact]
    public void GetUserByLoginTest()
    {
        var users = api.GetUsers();
        Assert.NotNull(users);
        Assert.NotEmpty(users);

        var user = api.GetUserByLogin(users[0].Login);
        Assert.NotNull(user);

        Assert.Equal(users[0].UserID, user.UserID);
        Assert.Equal(users[0].FirstName, user.FirstName);
        Assert.Equal(users[0].LastName, user.LastName);
        Assert.Equal(users[0].Login, user.Login);
        Assert.Equal(users[0].Password, user.Password);
        Assert.Equal(users[0].UserType, user.UserType);
    }

    [Fact]
    public void GetUserByIdTest()
    {
        var users = api.GetUsers();
        Assert.NotNull(users);
        Assert.NotEmpty(users);

        var user = api.GetUserById(users[0].UserID);
        Assert.NotNull(user);

        Assert.Equal(users[0].UserID, user.UserID);
        Assert.Equal(users[0].FirstName, user.FirstName);
        Assert.Equal(users[0].LastName, user.LastName);
        Assert.Equal(users[0].Login, user.Login);
        Assert.Equal(users[0].Password, user.Password);
        Assert.Equal(users[0].UserType, user.UserType);
    }

    [Fact]
    public void GetUsersTest()
    {
        var users = api.GetUsers();
        Assert.NotNull(users);
    }

    [Fact]
    public void GetResourceAmountsTest()
```

```
{
    var resources = api.GetResources();
    Assert.NotNull(resources);
    Assert.NotEmpty(resources);

    var tmp = api.GetResourceAmounts(resources[0].ResourceID);
    Assert.NotNull(tmp);
    Assert.Equal(tmp.Value.amount, tmp.Value.borrowed + tmp.Value.available);
}

[Fact]
public void GetBorrowRequestsTest()
{
    var allReq = api.GetBorrowRequests();
    Assert.NotNull(allReq);
    Assert.NotEmpty(allReq);

    var returnedReq = api.GetBorrowRequests(Status.Returned);
    var approvedReq = api.GetBorrowRequests(Status.Approved);
    var pendingReq = api.GetBorrowRequests(Status.Pending);

    Assert.NotNull(returnedReq);
    Assert.NotNull(approvedReq);
    Assert.NotNull(pendingReq);

    Assert.Equal(allReq.FindAll(r => r.Status == Status.Returned).Count(), returnedReq.Count());
    Assert.Equal(allReq.FindAll(r => r.Status == Status.Approved).Count(), approvedReq.Count());
    Assert.Equal(allReq.FindAll(r => r.Status == Status.Pending).Count(), pendingReq.Count());

    Assert.True(pendingReq.TrueForAll(r => r.DueDate == null && r.CopyID == null));
    Assert.True(approvedReq.TrueForAll(r => r.DueDate != null && r.CopyID != null));
    Assert.True(returnedReq.TrueForAll(r => r.DueDate != null && r.CopyID != null));
}

[Fact]
public void GetResourceCopiesTest()
{
    var resAll = api.GetResourceCopies();
    Assert.NotNull(resAll);
}

[Fact]
public void GetResourcesTest()
{
    var resAll = api.GetResources();
    Assert.NotNull(resAll);
    Assert.NotEmpty(resAll);

    var fil = api.GetResources(
        resAll[0].Title, resAll[0].Author, resAll[0].YearPublished, resAll[0].ResourceType);
    Assert.NotNull(fil);
    Assert.NotEmpty(fil);

    foreach (var item in fil)
    {
        Assert.Equal(resAll[0].ResourceID, item.ResourceID);
        Assert.Equal(resAll[0].Title, item.Title);
```

```
        Assert.Equal(resAll[0].Author, item.Author);
        Assert.Equal(resAll[0].YearPublished, item.YearPublished);
        Assert.Equal(resAll[0].ResourceType, item.ResourceType);
    }
}

[Fact]
public void GetResourcesBorrowedByUserTest()
{
    var users = api.GetUsers();
    Assert.NotNull(users);
    Assert.NotEmpty(users);

    List<Resource>? borrowed = null;
    int userID = 0;
    foreach (var user in users)
    {
        borrowed = api.GetResourcesBorrowedByUser(user.UserID);
        Assert.NotNull(borrowed);
        userID = user.UserID;
        if (borrowed.Count > 0)
        {
            break;
        }
    }
    Assert.NotNull(borrowed);

    var reqAll = api.GetBorrowRequests();
    Assert.NotNull(reqAll);

    var resCopiesAll = api.GetResourceCopies();
    Assert.NotNull(resCopiesAll);

    var customResIds = reqAll.FindAll(
        r => r.UserID == userID && r.Status == Status.Approved).Select(r => r.ResourceID);


    Assert.True(borrowed.TrueForAll(b => customResIds.Contains(b.ResourceID)));
}

[Fact]
public void UpdateBorrowRequestStatusToReturnedTest()
{
    var borrowedReq = api.GetBorrowRequests();
    Assert.NotNull(borrowedReq);
    Assert.NotEmpty(borrowedReq);

    var approvedAll = borrowedReq.FindAll(b => b.Status == Status.Approved);
    Assert.NotNull(approvedAll);

    foreach (var item in approvedAll)
    {
        Assert.NotNull(item.CopyID);
        Assert.True(api.UpdateBorrowRequestStatusToReturned(item.CopyID.Value));
    }

    var updatedBorrowedReq = api.GetBorrowRequests();
```

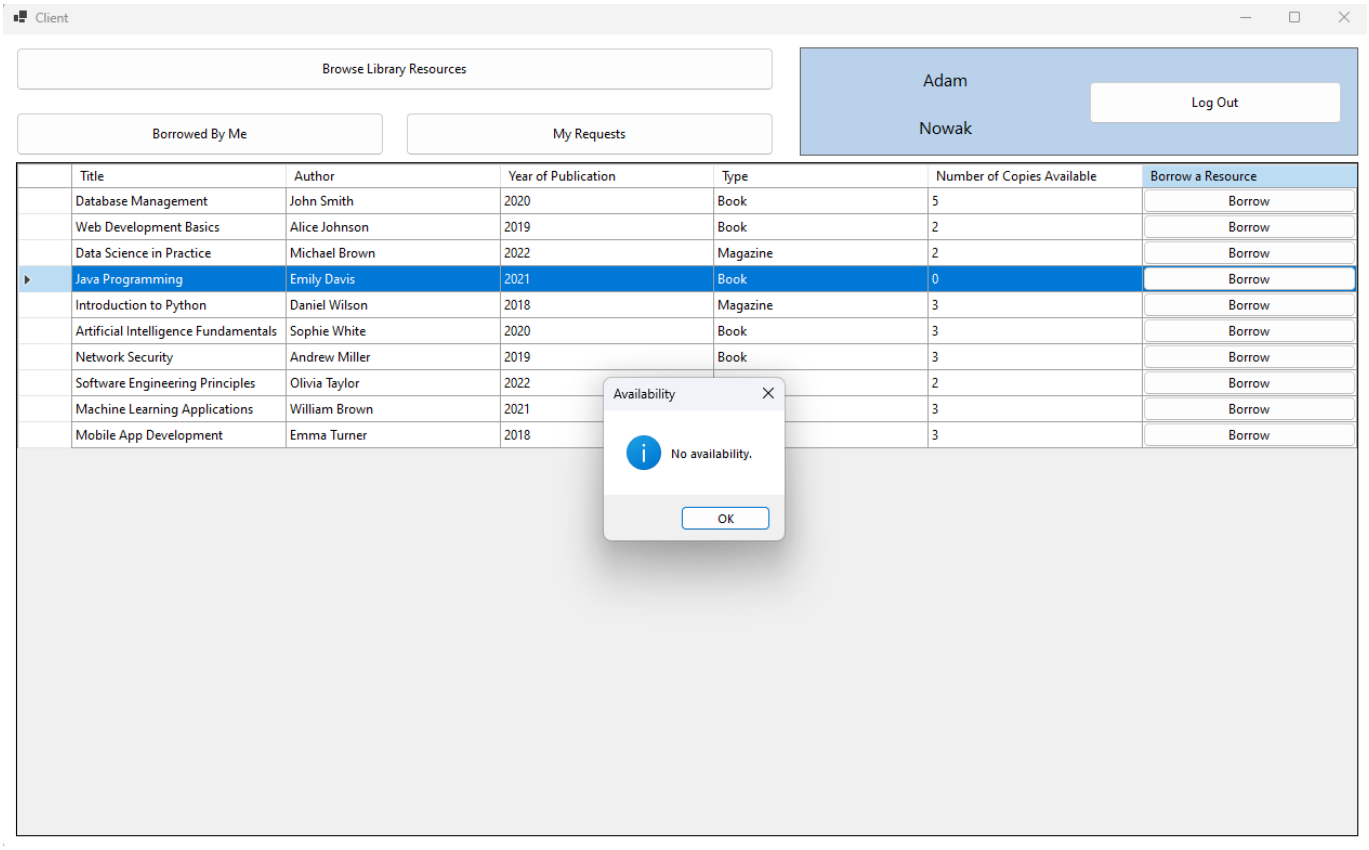
```
    Assert.NotNull(updatedBorrowedReq);
    Assert.True(updatedBorrowedReq.TrueForAll(
        ubr => ubr.Status == Status.Returned || ubr.Status == Status.Pending));

    foreach (var item in approvedAll)
    {
        Assert.True(api.UpdateBorrowRequest(
            item.RequestID, item.CopyID, item.DueDate, Status.Approved));
    }
}

[Fact]
public void UpdateBorrowRequestStatusToApprovedTest()
{
    var borrowedReq = api.GetBorrowRequests();
    Assert.NotNull(borrowedReq);
    Assert.NotEmpty(borrowedReq);

    var pendingAll = borrowedReq.FindAll(b => b.Status == Status.Pending);
    Assert.NotNull(pendingAll);

    var date = new DateOnly(2023, 1, 1);
    foreach (var item in pendingAll)
    {
        Assert.True(api.UpdateBorrowRequestStatusToApproved(item.RequestID, 1, date));
    }

    var updatedBorrowedReq = api.GetBorrowRequests();
    Assert.NotNull(updatedBorrowedReq);
    Assert.True(updatedBorrowedReq.TrueForAll(
        ubr => ubr.Status == Status.Returned || ubr.Status == Status.Approved));

    foreach (var item in pendingAll)
    {
        Assert.True(api.UpdateBorrowRequest(
            item.RequestID, item.CopyID, item.DueDate, item.Status));
    }
}

[Fact]
public void UpdateBorrowRequestTest()
{
    var borrowedReq = api.GetBorrowRequests();
    Assert.NotNull(borrowedReq);
    Assert.NotEmpty(borrowedReq);

    var pendingAll = borrowedReq.FindAll(b => b.Status == Status.Pending);
    Assert.NotNull(pendingAll);
    var approvedAll = borrowedReq.FindAll(b => b.Status == Status.Approved);
    Assert.NotNull(approvedAll);


    var date = new DateOnly(2023, 1, 1);
    foreach (var item in borrowedReq)
    {
        Assert.True(api.UpdateBorrowRequest(item.RequestID, 1, date, Status.Returned));
    }
```

```
    var updatedBorrowedReq = api.GetBorrowRequests();
    Assert.NotNull(updatedBorrowedReq);
    Assert.True(updatedBorrowedReq.TrueForAll(ubr => ubr.Status == Status.Returned));
    Assert.True(updatedBorrowedReq.TrueForAll(ubr => ubr.CopyID == 1));
    Assert.True(updatedBorrowedReq.TrueForAll(ubr => ubr.DueDate == date));

    foreach (var item in borrowedReq)
    {
        Assert.True(api.UpdateBorrowRequest(
            item.RequestID, item.CopyID, item.DueDate, item.Status));
    }
}

[Fact]
public void PostDeleteUserTest()
{
    Assert.True(api.PostNewUser(
        "testname", "testlastname", "testlogin", "123", UserType.Employee));

    var user = api.GetUserByLogin("testlogin");
    Assert.NotNull(user);

    var users = api.GetUsers();
    Assert.NotNull(users);

    Assert.True(api.DeleteUser(user.UserID));

    var newUsers = api.GetUsers();
    Assert.NotNull(newUsers);

    Assert.Equal(users.Count(), newUsers.Count() + 1);
}

[Fact]
public void PostDeleteBorrowRequestTest()
{
    var users = api.GetUsers();
    var res = api.GetResources();
    Assert.NotNull(users);
    Assert.NotEmpty(users);
    Assert.NotNull(res);
    Assert.NotEmpty(res);

    var req = api.GetBorrowRequests();
    Assert.NotNull(req);
    Assert.NotEmpty(req);

    Assert.True(api.PostNewBorrowRequest(
        users[0].UserID, res[0].ResourceID, new DateOnly(1900,1,1), null, null, Status.Pending));

    var newReq = api.GetBorrowRequests();
    Assert.NotNull(newReq);

    var selected = newReq.FindAll(e =>
    e.UserID == users[0].UserID &&
    res[0].ResourceID == e.ResourceID &&
```

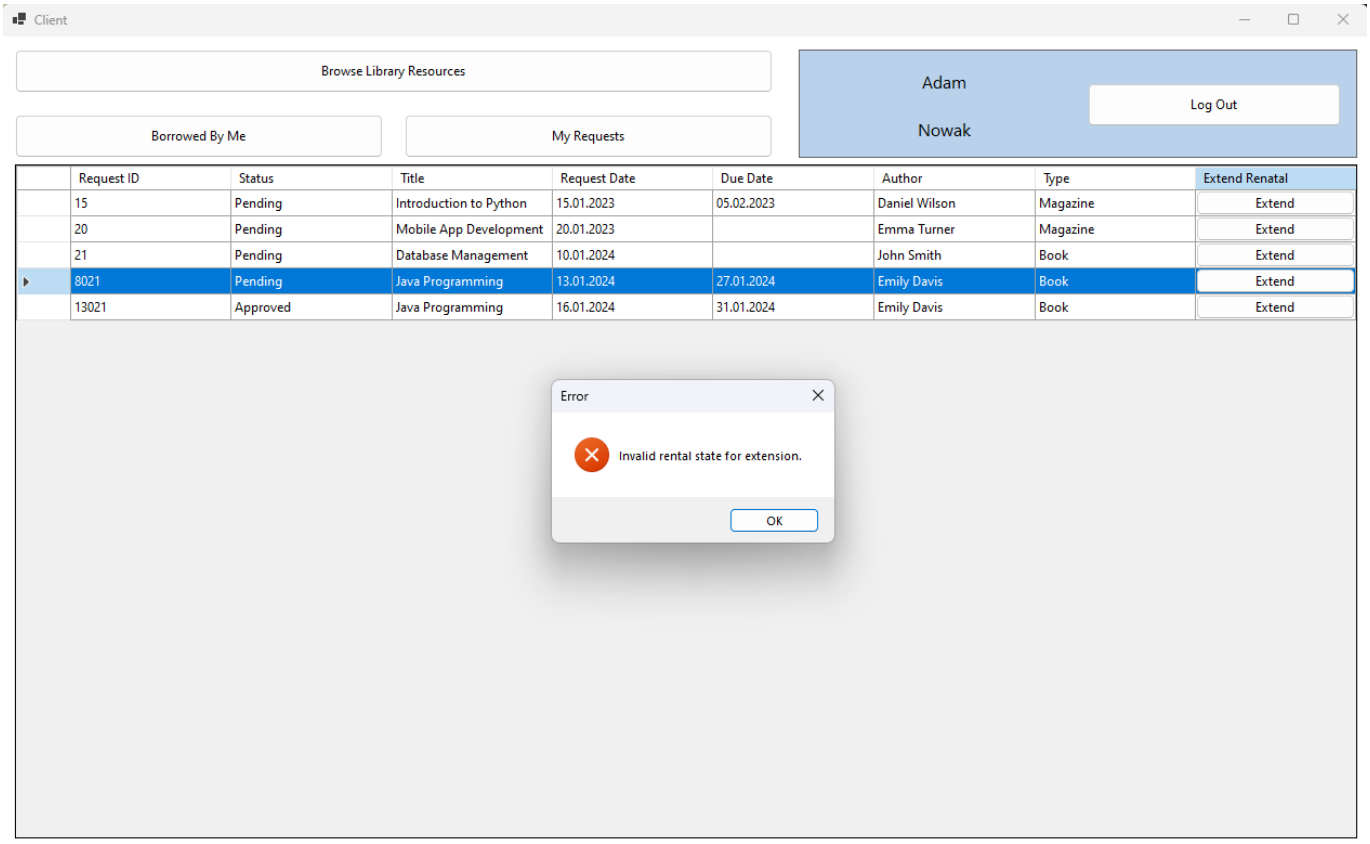
```
    e.RequestDate == new DateOnly(1900, 1, 1) &&
    e.DueDate == null &&
    e.CopyID == null &&
    e.Status == Status.Pending).Single();

    Assert.Equal(req.Count() + 1, newReq.Count());

    Assert.True(api.DeleteBorrowRequest(selected.RequestID));

    var newNewReq = api.GetBorrowRequests();
    Assert.NotNull(newNewReq);
    Assert.Equal(req.Count(), newNewReq.Count());
}

[Fact]
public void PostDeleteResourceCopyTest()
{
    var res = api.GetResources();
    Assert.NotNull(res);
    Assert.NotEmpty(res);

    var resCop = api.GetResourceCopies();
    Assert.NotNull(resCop);
    Assert.NotEmpty(resCop);

    Assert.True(api.PostNewResourceCopy(res[0].ResourceID));

    var newResCop = api.GetResourceCopies();
    Assert.NotNull(newResCop);

    Assert.Equal(resCop.Count() + 1, newResCop.Count());

    var selected = newResCop.FindAll(s => s.ResourceID == res[0].ResourceID).MaxBy(s => s.CopyID);
    Assert.NotNull(selected);

    Assert.True(api.DeleteResourceCopy(selected.CopyID));

    var newNewResCop = api.GetResourceCopies();
    Assert.NotNull(newNewResCop);
    Assert.Equal(resCop.Count(), newNewResCop.Count());
}

[Fact]
public void PostDeleteResourceTest()
{
    var res = api.GetResources();
    Assert.NotNull(res);
    Assert.NotEmpty(res);

    Assert.True(api.PostNewResource("Testtitle", "testauth", 1000, ResourceType.Magazine));

    var newRes = api.GetResources();
    Assert.NotNull(newRes);

    Assert.Equal(res.Count() + 1, newRes.Count());

    var added = api.GetResources("Testtitle", "testauth", 1000, ResourceType.Magazine);
```

```
        Assert.NotNull(added);

        Assert.True(api.DeleteResource(added[0].ResourceID));

        var newNewRes = api.GetResources();
        Assert.NotNull(newNewRes);
        Assert.Equal(res.Count(), newNewRes.Count());
    }
}
```

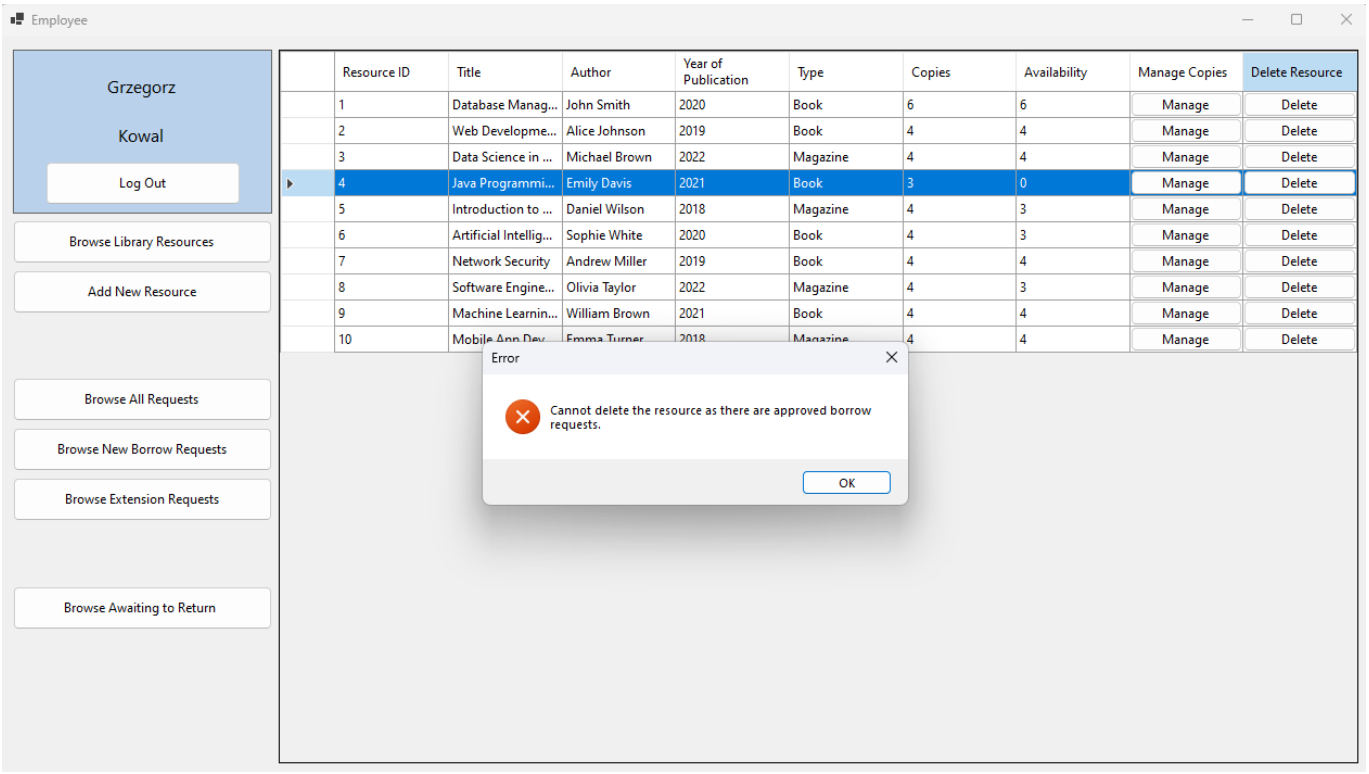
4.7.2 Testy GUI



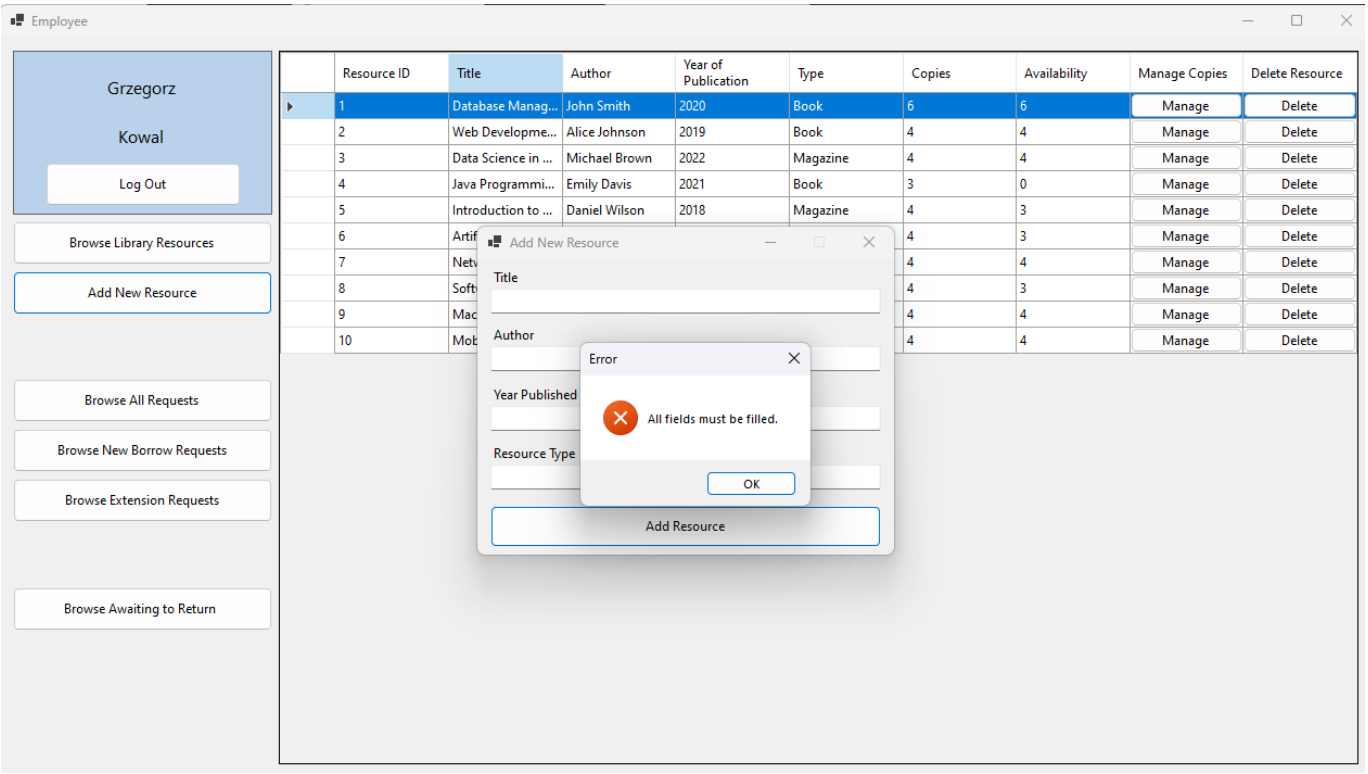
Rysunek 29: Test 1: Próba wypożyczenia zasobu, dla którego nie ma dostępnych kopii



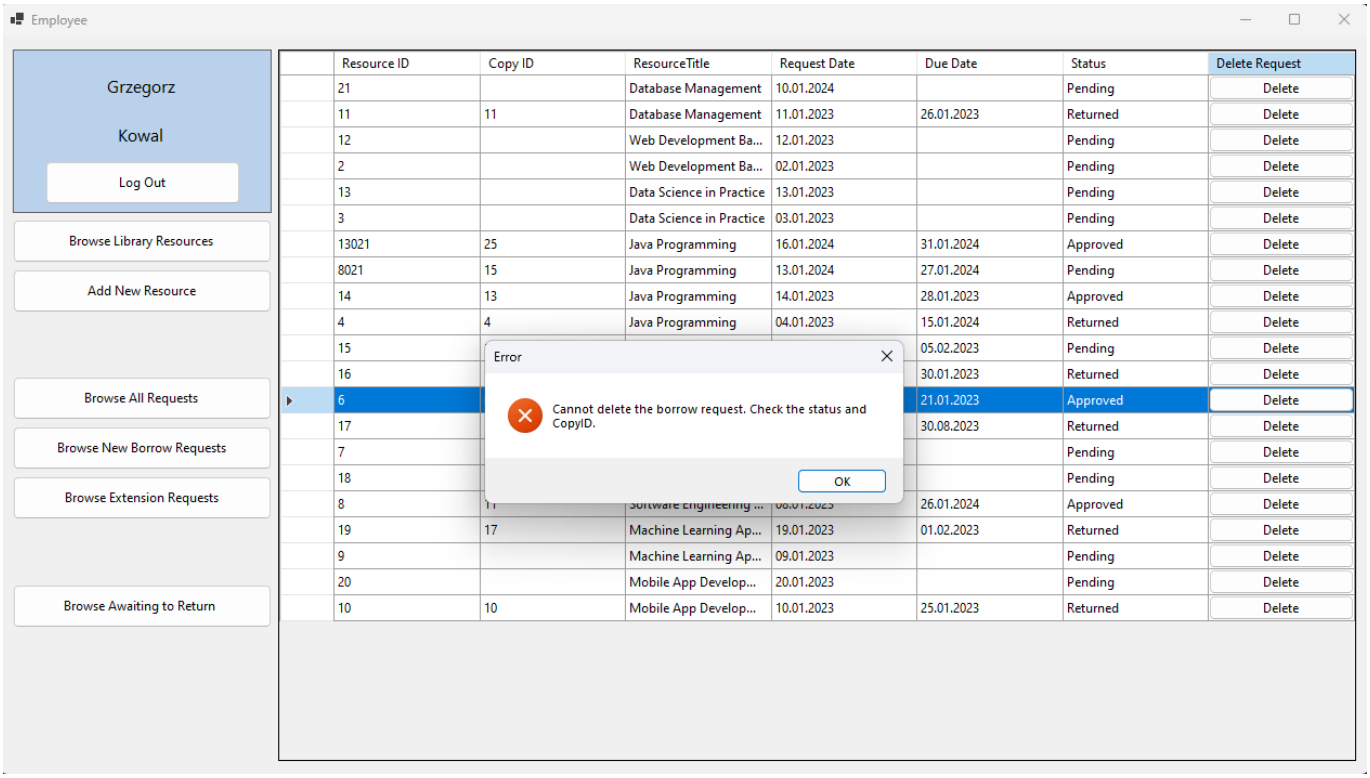
Rysunek 30: Test 2: Próba przedłużenia wypożyczenia egzemplarza, który nie jest wypożyczony przez użytkownika



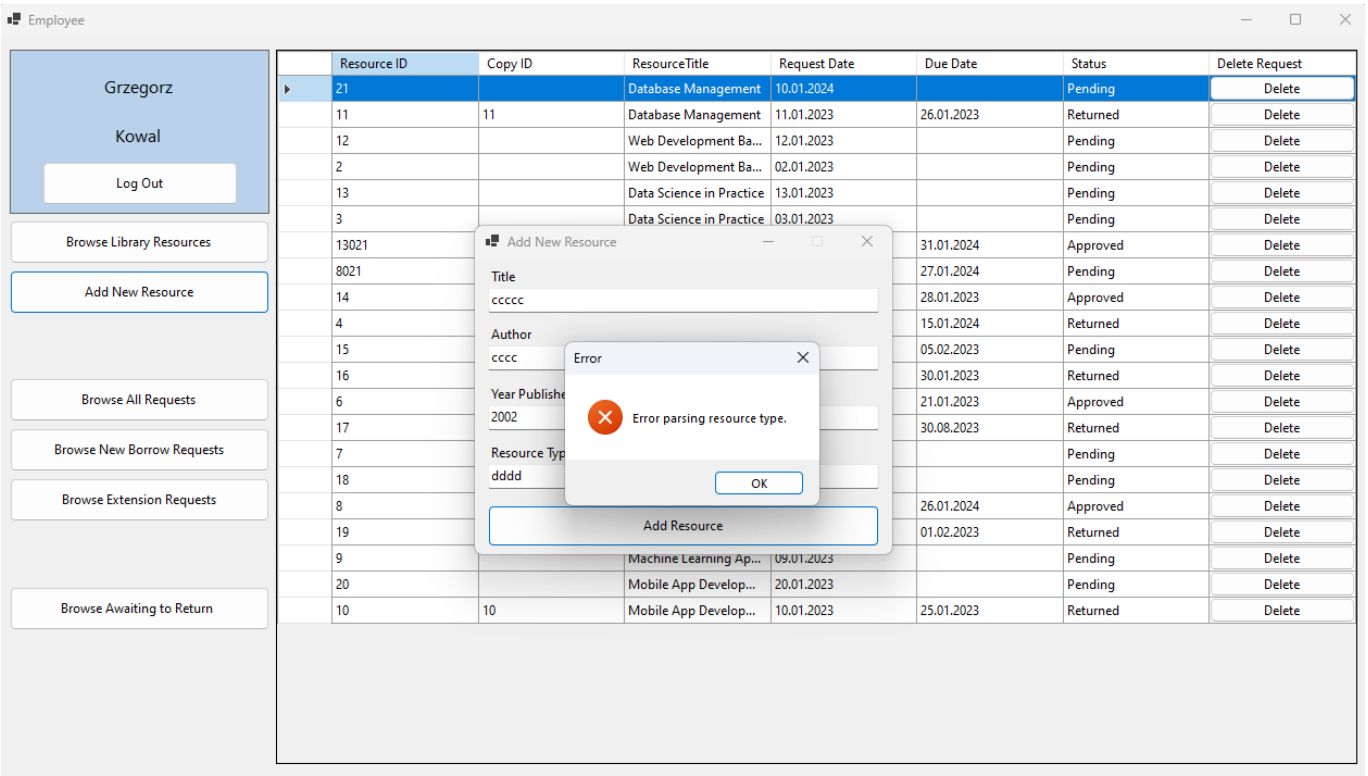
Rysunek 31: Test 3: Próba usunięcia zasobu, którego egzemplarz jest aktualnie wypożyczony klientowi



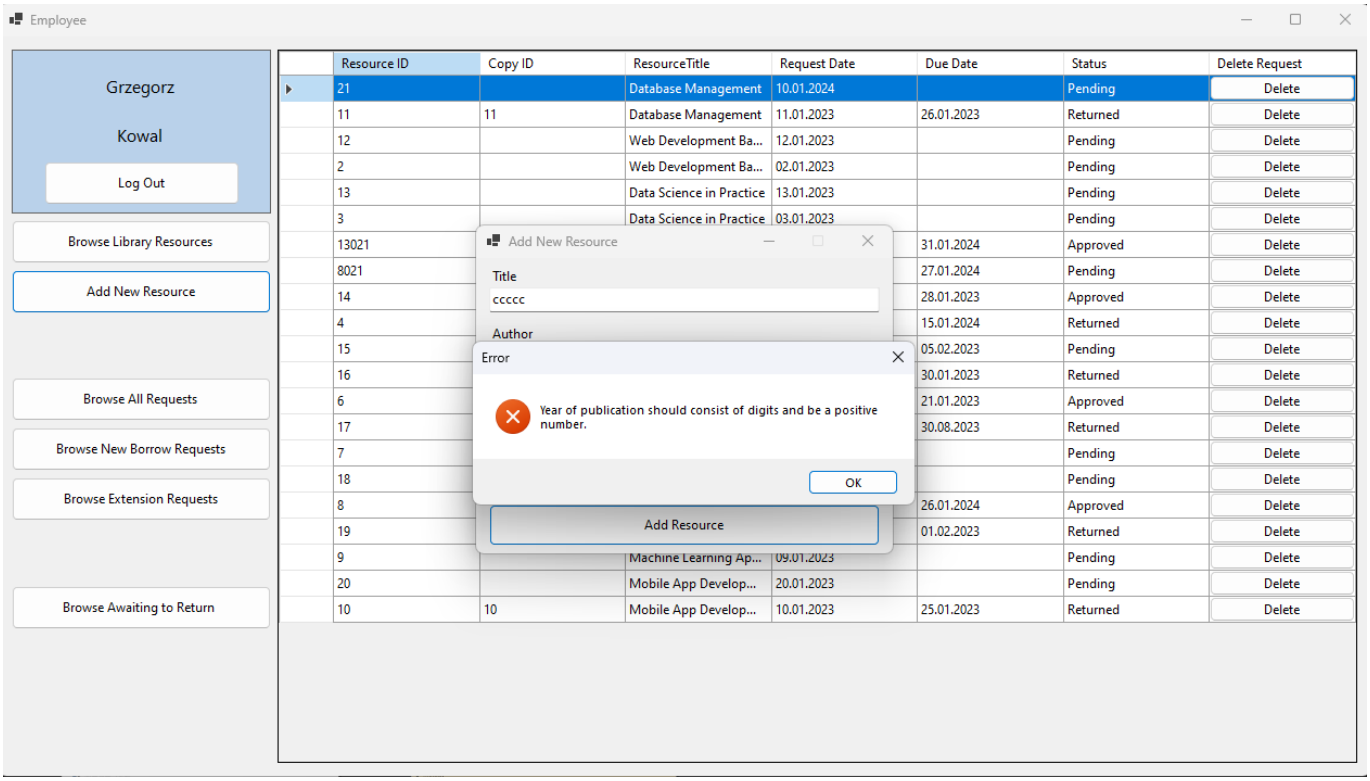
Rysunek 32: Test 4: Próba dodania zasobu bez uzupełnienia pól formularza



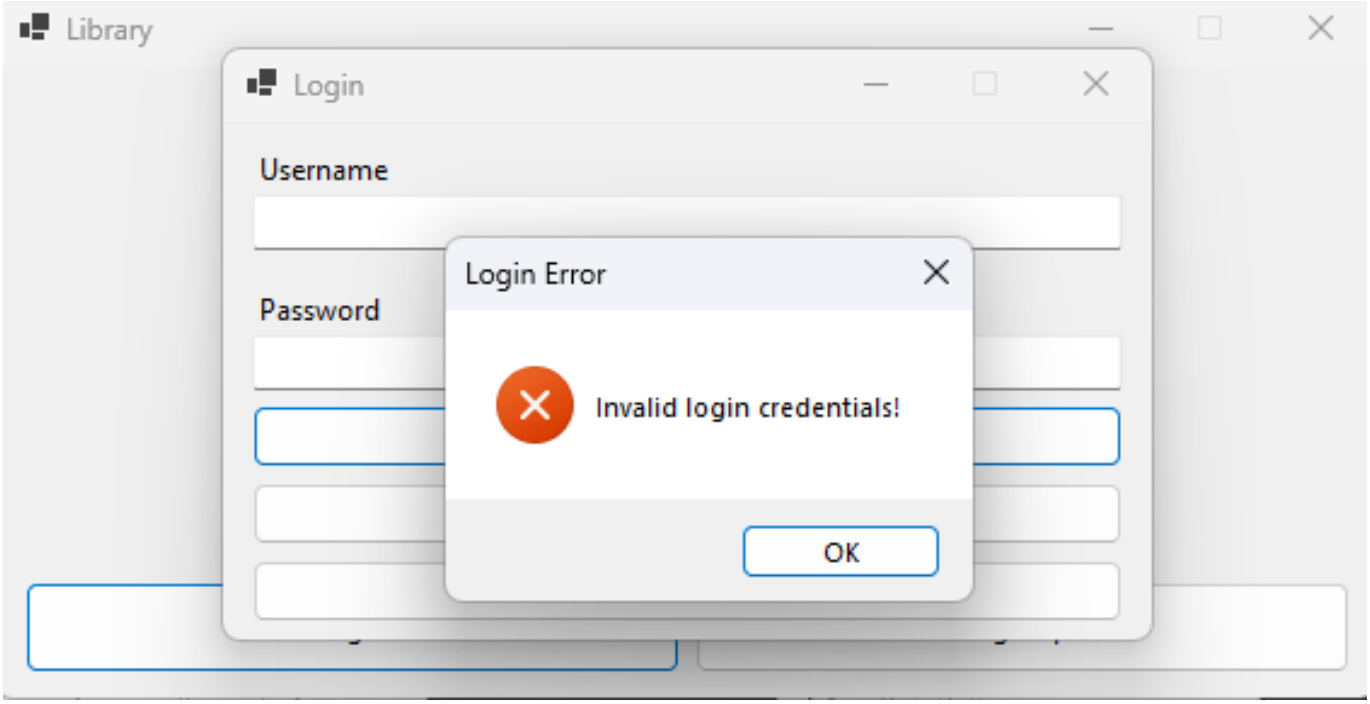
Rysunek 33: Test 5: Próba usunięcie prośby o wypożyczenie dla niezwróconej kopii.



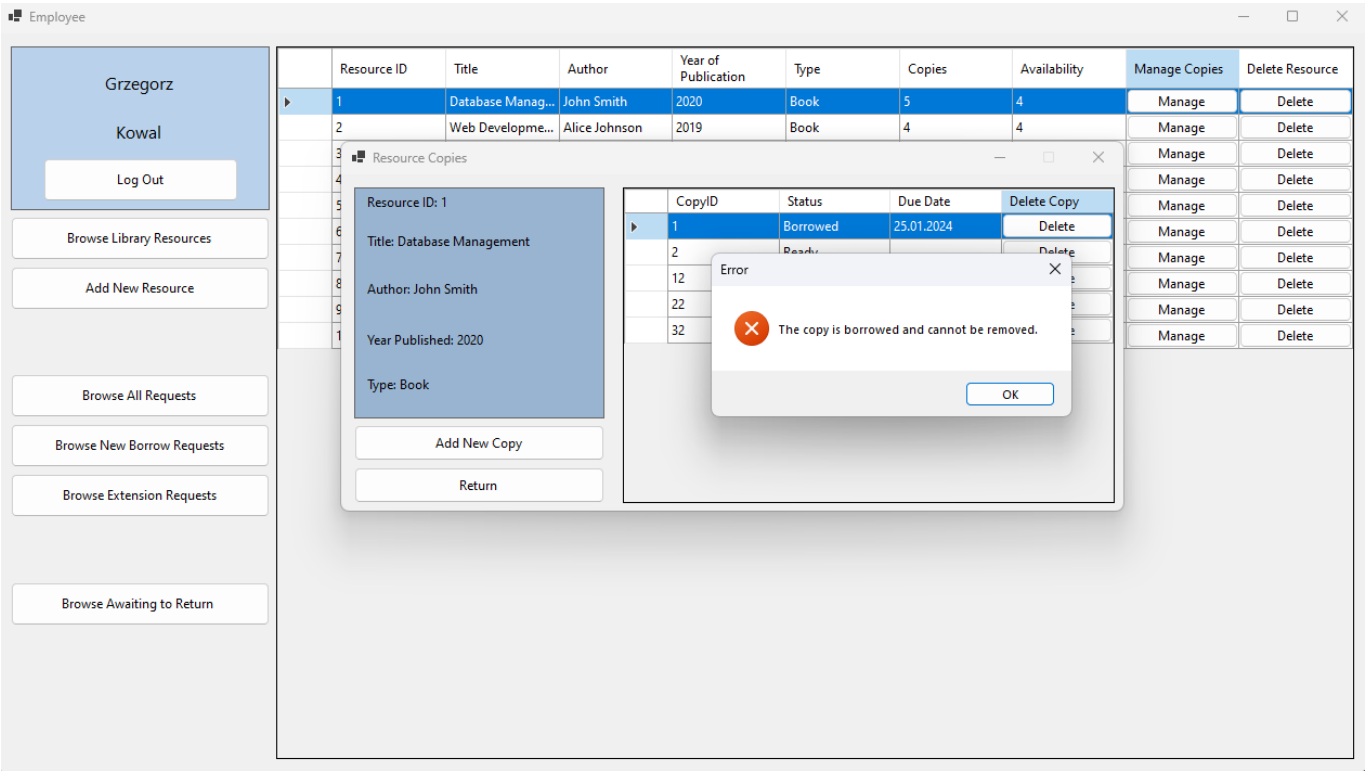
Rysunek 34: Test 6: Próba dodanie zasobu o nieznanym typie.



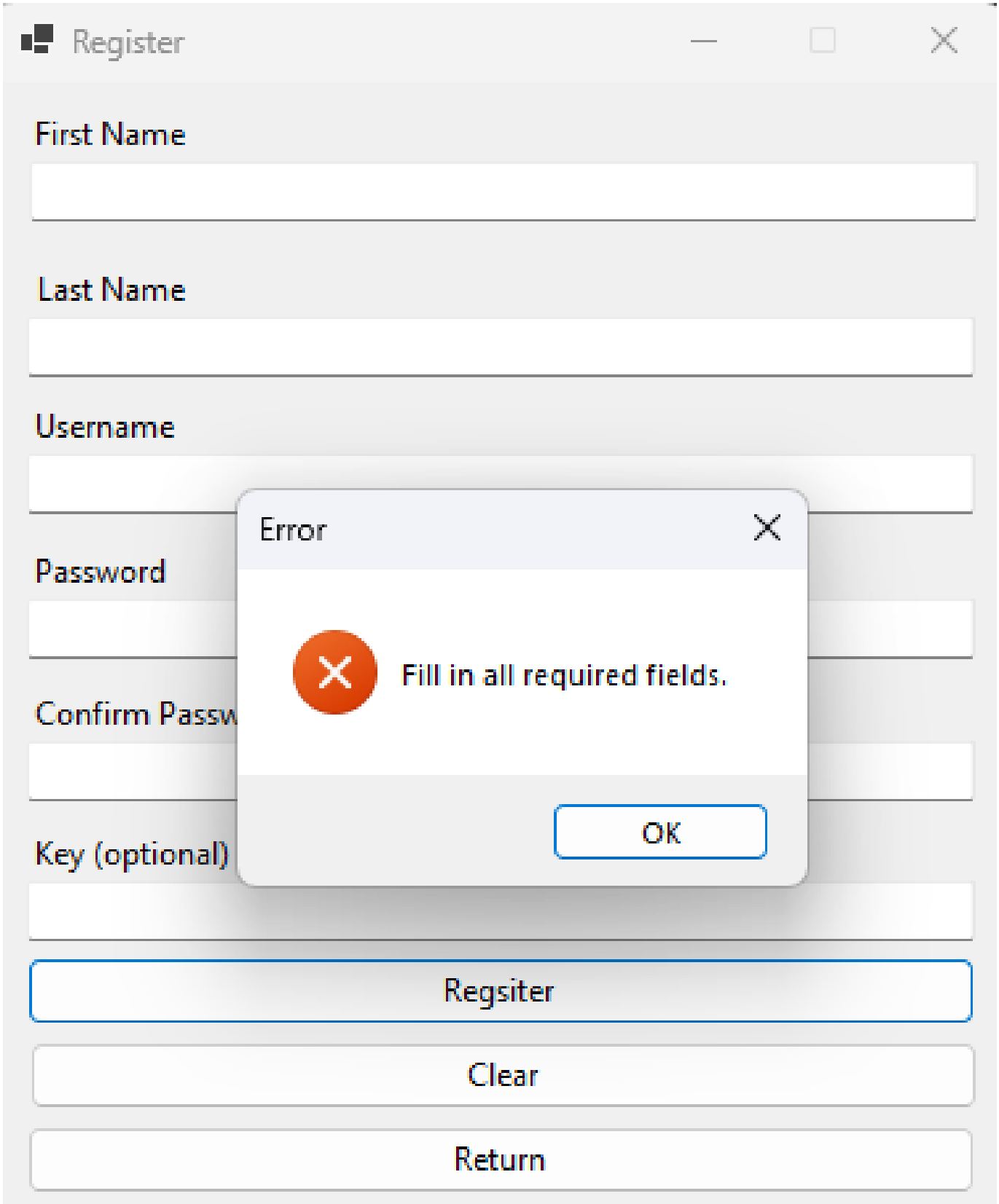
Rysunek 35: Test 7: Próba dodanie zasobu o niepoprawnej dacie wydania.



Rysunek 36: Test 8: Próba zalogowania bez podania danych



Rysunek 37: Test 9: Próba usunięcia wypozyczonej kopii zasobu.



Rysunek 38: Test 10: Próba rejestracji konta bez uzupełnionego formularza.