
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

1	Opis wycinka rzeczywistości	5
1.1	Opis rzeczywistości	5
1.2	Zasoby ludzkie	5
1.3	Opisy sytuacji	5
2	Model bazy danych	6
2.1	Wymagania funkcjonalne i нефункционалне	6
2.2	Diagram przypadków użycia	7
2.3	Opis scenariuszy	8
2.4	Diagram ERD	21
3	Implementacja bazy danych	21
3.1	Sformułowanie wymagań dotyczących dostępu do bazy i jej zawartości	21
3.2	Opracowanie diagramu modelu bazy na podstawie diagramu ERD	22
3.3	Model fizyczny bazy danych	22
3.3.1	Tabele	22
3.3.2	Widoki	23
3.3.3	Przykładowe Dane	25
3.3.4	Wdrożenie i przetestowanie bazy	28
4	Implementacja aplikacji	36
4.1	Zmiany w diagramie przypadków użycia	36
4.2	Makieta interfejsu graficznego	37
4.3	Diagram klas	43
4.4	Wdrożenie aplikacji	45
4.4.1	Okna startowe	45
4.4.2	Okna klienta	47
4.4.3	Okna pracownika	48
4.5	Testowanie aplikacji	53

4.5.1	Testy jednostkowe	53
4.5.2	Testy GUI	59

Spis tabel

Spis rysunków

1	Diagram przypadków użycia	7
2	Diagram ERD	21
3	Diagram wymagań dostępu do bazy i jej zawartości	21
4	Diagram modelu bazy danych	22
5	Sprawdzenie integralności semantycznej- Kod	28
6	Sprawdzenia integralności semantycznej - Wyniki	29
7	Sprawdzenie integralności encji: - Kod	29
8	Sprawdzenie integralności encji: - Wyniki	29
9	Sprawdzenie integralności semantycznej- Kod	30
10	Sprawdzenia integralności semantycznej - Wyniki	30
11	Sprawdzenie integralności encji: - Kod	31
12	Sprawdzenie integralności encji: - Wyniki	31
13	Sprawdzenie integralności semantycznej- Kod	31
14	Sprawdzenia integralności semantycznej - Wyniki	32
15	Sprawdzenie integralności encji: - Kod	33
16	Sprawdzenie integralności encji: - Wyniki	33
17	Sprawdzenie integralności referencji: - Kod	33
18	Sprawdzenie integralności referencji: - Wyniki	33
19	Sprawdzenie integralności semantycznej- Kod	34
20	Sprawdzenia integralności semantycznej - Wyniki	34
21	Sprawdzenie integralności encji: - Kod	35
22	Sprawdzenie integralności encji: - Wyniki	35
23	Sprawdzenie integralności referencji: - Kod	35
24	Sprawdzenie integralności referencji: - Wyniki	36

25	Nowy diagram przypadków użycia.	36
26	Makieta widoku powitalnego.	37
27	Makieta widoku logowania.	37
28	Makieta widoku rejestracji.	38
29	Makieta widoku klienta.	39
30	Makieta widoku pracownika.	40
31	Makieta widoku dodawania zasobu.	41
32	Makieta widoku zarządzania kopiami zasobu.	41
33	Makieta widoku akceptacji prośby o wypożyczenie.	42
34	Makieta widoku przedłużenia prośby o wypożyczenie.	42
35	Diagram klas dla Front-End'u.	43
36	Diagram klas dla Back-End'u.	44
37	Okno powitalne.	45
38	Okno logowania.	45
39	Okno rejestracji.	46
40	Okno przeglądania zbioru biblioteki.	47
41	Okno przeglądania pożyczonych pozycji.	47
42	Okno przeglądania próśb o wypożyczenie.	48
43	Okno przeglądania zasobów biblioteki.	48
44	Okno zarządzania egzemplarzami zasobu.	49
45	Okno dodawania nowego zasobu.	49
46	Okno przeglądania wszystkich próśb o wypożyczenie.	50
47	Okno przeglądania nowych próśb o wypożyczenie.	50
48	Okno akceptacji prośby o wypożyczenie.	51
49	Okno przeglądania próśb o przedłużenie wypożyczenia.	51
50	Okno akceptacji prośby o przedłużenie wypożyczenia.	52
51	Okno przeglądania wypożyczonych egzemplarzy (oczekujących na zwrot).	52
52	Test 1: Próba wypożyczenia zasobu, dla którego nie ma dostępnych kopii.	59
53	Test 2: Próba przedłużenia wypożyczenia egzemplarza, który nie jest wypożyczony przez użytkownika.	60

54	Test 3: Próba usunięcia zasobu, którego egzemplarz jest aktualnie wypożyczony klientowi	60
55	Test 4: Próba dodania zasobu bez uzupełnienia pól formularza	61
56	Test 5: Próba usunięcie prośby o wypożyczenie dla niezwróconej kopii.	61
57	Test 6: Próba dodanie zasobu o nieznanym typie.	62
58	Test 7: Próba dodanie zasobu o niepoprawnej dacie wydania.	62
59	Test 8: Próba zalogowania bez podania danych	63
60	Test 9: Próba usunięcia wypożyczonej kopii zasobu.	63
61	Test 10: Próba rejestracji konta bez uzupełnionego formularza.	64

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, kaset),
- 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 нефункционалне

1. Wymagania funkcjonalne

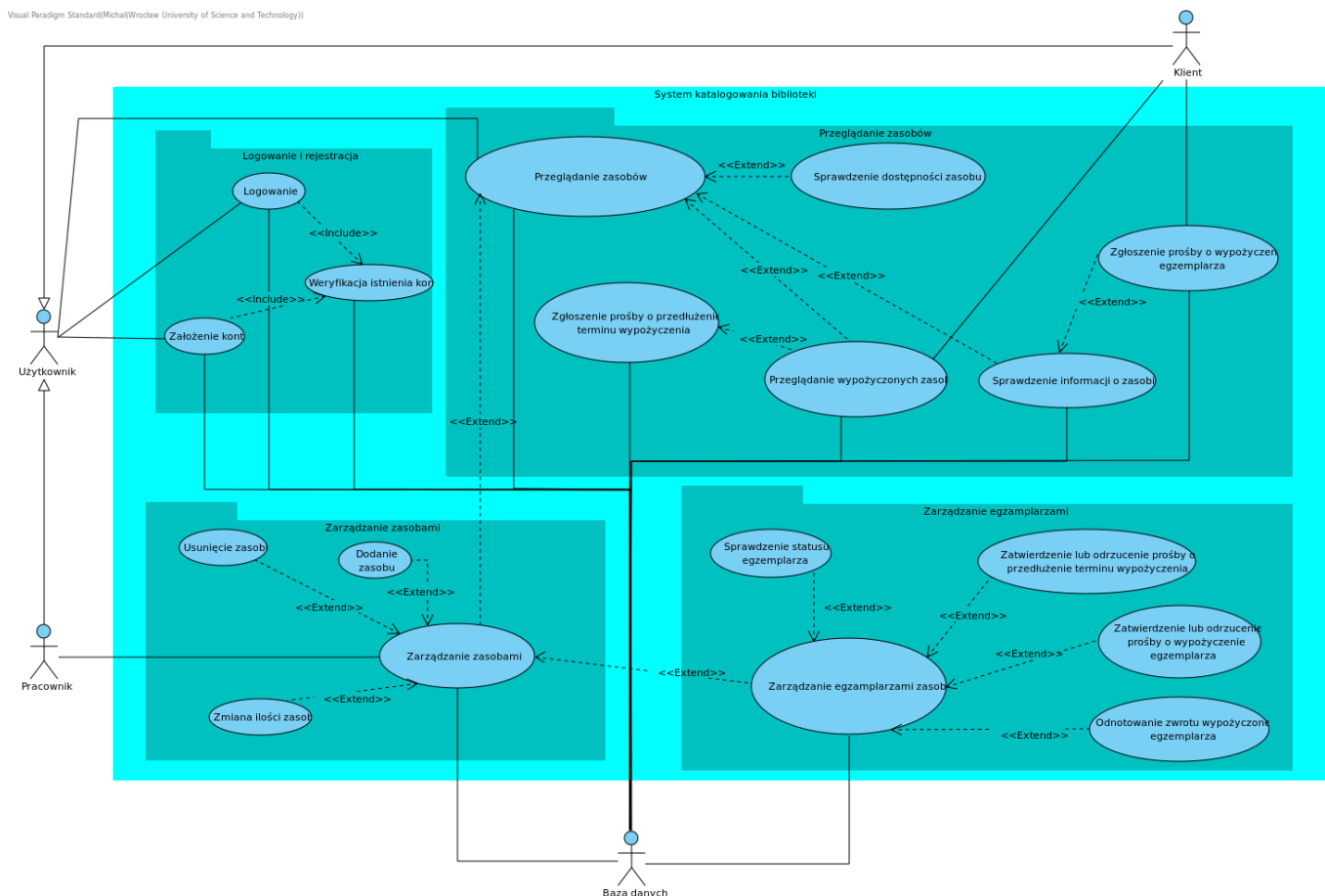
- (a) Użytkownik zakłada konto
- (b) Użytkownik loguje się
- (c) Użytkownik przegląda zasoby
- (d) Użytkownik sprawdza dostępność danego zasobu
- (e) Użytkownik sprawdza informację o zasobie
- (f) Pracownik dodaje zasoby
- (g) Pracownik usuwa zasoby
- (h) Pracownik zarządza zasobami
- (i) Pracownik zmienia 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 нефункционалне

- (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 zweryfikowaniu.
 - 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 hasle 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 wypadek:
 - 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 wpisuje 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 **Odnnotowanie 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 wypożyczenie 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 akcję "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) Zostają 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

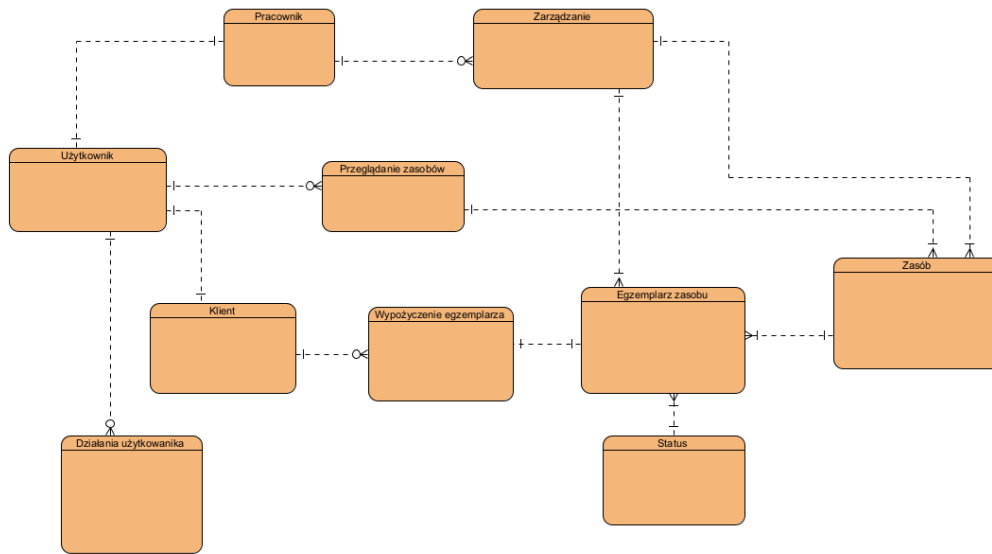
17. Sprawdzenie informacji o zasobie

- 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

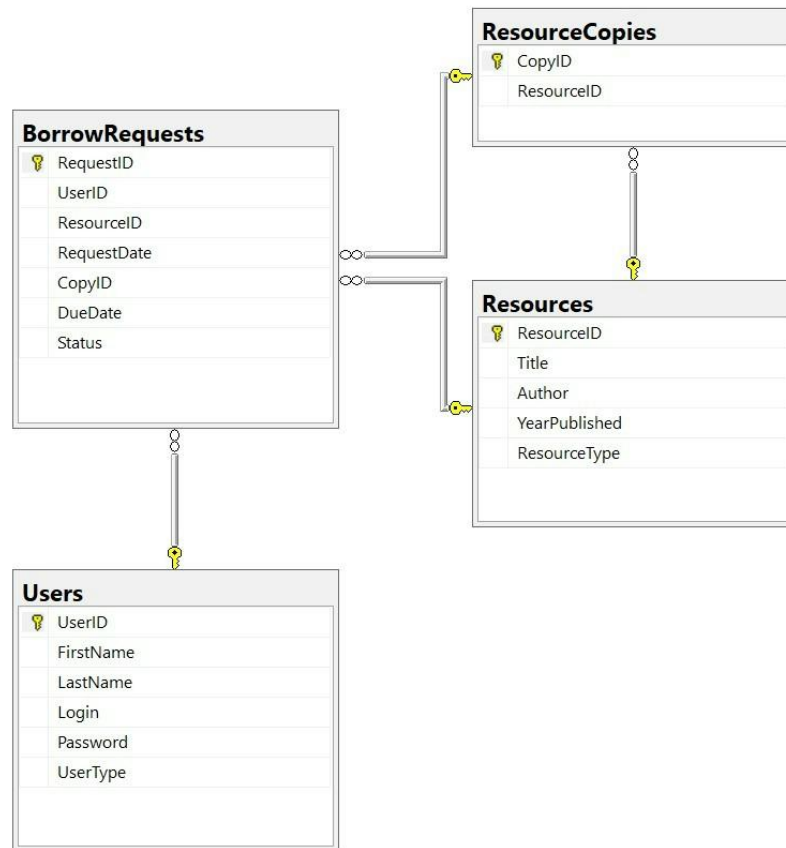
3 Implementacja bazy danych

3.1 Sformułowanie wymagań dotyczących dostępu do bazy i jej zawartości

TABELA\UŻYTKOWNIK	Administrator	Pracownik	Klient
Users	Wszystko	SELECT, UPDATE	SELECT
Resources		SELECT, INSERT, UPDATE, DELETE	SELECT
ResourceCopies		SELECT, INSERT, UPDATE, DELETE	SELECT
BorrowRequests		SELECT, INSERT, UPDATE	SELECT, INSERT, UPDATE

Rysunek 3: Diagram wymagań dostępu do bazy i jej zawartości

3.2 Opracowanie diagramu modelu bazy na podstawie diagramu ERD



Rysunek 4: Diagram modelu bazy danych

3.3 Model fizyczny bazy danych

3.3.1 Tabele

```
CREATE TABLE Users (  
  UserID INT PRIMARY KEY IDENTITY(1,1),  
  FirstName VARCHAR(255) NOT NULL,  
  LastName VARCHAR(255) NOT NULL,  
  Login VARCHAR(255) UNIQUE NOT NULL,  
  Password VARCHAR(255) NOT NULL,  
  UserType VARCHAR(50) NOT NULL -- Client / Employee  
);
```

GO

```
CREATE TABLE Resources (  
  ResourceID INT PRIMARY KEY IDENTITY(1,1),  
  Title VARCHAR(255) NOT NULL,  
  Author VARCHAR(255) NOT NULL,
```

```

        YearPublished INT NOT NULL,
        ResourceType VARCHAR(50) NOT NULL, -- Book / Magazine / ....
    );

GO

-- represents individual copies of resources
CREATE TABLE ResourceCopies (
    CopyID INT PRIMARY KEY IDENTITY(1,1),
    ResourceID INT NOT NULL,
    FOREIGN KEY (ResourceID) REFERENCES Resources(ResourceID),
);

GO

CREATE TABLE BorrowRequests (
    RequestID INT PRIMARY KEY IDENTITY(1,1),
    UserID INT NOT NULL,
    ResourceID INT NOT NULL,
    RequestDate DATE NOT NULL,
    CopyID INT, -- Assigned by Employee after acceptance
    DueDate DATE, -- Assigned by Employee after acceptance
    Status VARCHAR(50) NOT NULL, -- Pending / Approved / Returned
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (CopyID) REFERENCES ResourceCopies(CopyID),
    FOREIGN KEY (ResourceID) REFERENCES Resources(ResourceID),
);

GO

```

3.3.2 Widoki

```

CREATE VIEW UserWithBorrowedResources AS
SELECT
    U.UserID,
    R.ResourceID,
    R.Title,
    R.Author,
    R.YearPublished,
    R.ResourceType
FROM Users U
JOIN BorrowRequests BR ON U.UserID = BR.UserID
JOIN ResourceCopies RC ON BR.CopyID = RC.CopyID
JOIN Resources R ON BR.ResourceID = R.ResourceID
WHERE BR.CopyID IS NOT NULL AND BR.Status <> 'Returned';
GO

CREATE VIEW SummarisedResources AS
SELECT
    R.ResourceID,
    R.Title,
    COALESCE(RCTotal.TotalCopies, 0) AS TotalCopies,
    COALESCE(BRTotal.BorrowedCopies, 0) AS BorrowedCopies
FROM Resources R
LEFT JOIN (
    SELECT RC.ResourceID, COUNT(RC.CopyID) AS TotalCopies
    FROM ResourceCopies RC

```



```

        GROUP BY RC.ResourceID
    ) RCTotal ON R.ResourceID = RCTotal.ResourceID
LEFT JOIN (
    SELECT BR.ResourceID, COUNT(DISTINCT BR.RequestID) AS BorrowedCopies
    FROM BorrowRequests BR
    WHERE BR.Status IN ('Approved', 'Pending') AND BR.CopyID IS NOT NULL
    GROUP BY BR.ResourceID
) BRTotal ON R.ResourceID = BRTotal.ResourceID;
GO

```

```

CREATE VIEW UserDetailsWithBorrowedResources AS
SELECT
    U.UserID,
    U.FirstName,
    U.LastName,
    U.Login,
    U.UserType,
    BR.RequestID,
    BR.RequestDate,
    BR.CopyID,
    BR.DueDate,
    R.Title AS BorrowedResource
FROM Users U
JOIN BorrowRequests BR ON U.UserID = BR.UserID
JOIN ResourceCopies RC ON BR.CopyID = RC.CopyID
JOIN Resources R ON BR.ResourceID = R.ResourceID
WHERE BR.Status = 'Approved';
GO

```

```

CREATE VIEW DelayedBorrowersView AS
SELECT
    U.UserID,
    U.FirstName,
    U.LastName,
    BR.RequestID,
    R.Title AS BorrowedResource,
    BR.DueDate,
    DATEDIFF(DAY, BR.DueDate, GETDATE()) AS DaysLate
FROM Users U
JOIN BorrowRequests BR ON U.UserID = BR.UserID
JOIN ResourceCopies RC ON BR.CopyID = RC.CopyID
JOIN Resources R ON BR.ResourceID = R.ResourceID
WHERE BR.Status = 'Returned' AND BR.DueDate < GETDATE();
GO

```

```

CREATE VIEW ApprovedBorrowRequests AS
SELECT
    BR.RequestID,
    U.FirstName,
    U.LastName,
    R.Title AS BorrowedResource,
    BR.RequestDate,
    BR.DueDate,
    BR.Status
FROM BorrowRequests BR
JOIN Users U ON BR.UserID = U.UserID
JOIN Resources R ON BR.ResourceID = R.ResourceID

```

```

WHERE BR.Status = 'Approved';
GO

CREATE VIEW PendingBorrowRequests AS
SELECT
    BR.RequestID,
    U.FirstName,
    U.LastName,
    R.Title AS RequestedResource,
    BR.RequestDate,
    BR.DueDate
FROM BorrowRequests BR
JOIN Users U ON BR.UserID = U.UserID
JOIN Resources R ON BR.ResourceID = R.ResourceID
WHERE BR.Status = 'Pending';
GO

```

3.3.3 Przykładowe Dane

```

-- Users
INSERT INTO Users (FirstName, LastName, Login, Password, UserType) VALUES
('Adam', 'Nowak', 'anowak', 'pass123', 'Client'),
('Ewa', 'Kowalska', 'ekowalska', 'securepass', 'Client'),
('Mateusz', 'Wójcik', 'mwojcik', 'pass456', 'Client'),
('Karolina', 'Lis', 'klis', 'strongpass', 'Client'),
('Marcin', 'Kaczmarek', 'mkaczmarek', 'userpass', 'Client'),
('Alicja', 'Pawlak', 'apawlak', 'pass789', 'Client'),
('Michał', 'Duda', 'mduda', 'testpass', 'Client'),
('Katarzyna', 'Szymańska', 'kszymanska', 'mypassword', 'Client'),
('Piotr', 'Kowalczyk', 'pkowalczyk', 'pass1234', 'Client'),
('Anna', 'Jankowska', 'ajankowska', 'pass5678', 'Client'),
('Tomasz', 'Wiśniewski', 'twisniewski', 'securepass', 'Client'),
('Magdalena', 'Zajac', 'mzajac', 'pass987', 'Client'),
('Grzegorz', 'Kowal', 'gkowal', 'testpass', 'Employee'),
('Agnieszka', 'Nowicka', 'anowicka', 'mypass', 'Employee'),
('Łukasz', 'Sikora', 'lsikora', 'pass654', 'Employee');

GO

-- Resources
INSERT INTO Resources (Title, Author, YearPublished, ResourceType) VALUES
('Database Management', 'John Smith', 2020, 'Book'),
('Web Development Basics', 'Alice Johnson', 2019, 'Book'),
('Data Science in Practice', 'Michael Brown', 2022, 'Magazine'),
('Java Programming', 'Emily Davis', 2021, 'Book'),
('Introduction to Python', 'Daniel Wilson', 2018, 'Magazine'),
('Artificial Intelligence Fundamentals', 'Sophie White', 2020, 'Book'),
('Network Security', 'Andrew Miller', 2019, 'Book'),
('Software Engineering Principles', 'Olivia Taylor', 2022, 'Magazine'),
('Machine Learning Applications', 'William Brown', 2021, 'Book'),
('Mobile App Development', 'Emma Turner', 2018, 'Magazine');

GO

-- ResourceCopies
INSERT INTO ResourceCopies (ResourceID) VALUES
(1),

```

[illegible]

GO

```
INSERT INTO BorrowRequests (UserID, ResourceID, RequestDate, CopyID, DueDate, Status) VALUES
```

```
(6, 6, '2023-01-06', 6, '2023-01-21', 'Approved'),
```

```
(8, 7, '2023-01-07', NULL, NULL, 'Pending'),
(10, 8, '2023-01-08', 11, '2024-01-26', 'Approved'),
(7, 9, '2023-01-09', NULL, NULL, 'Pending'),
(9, 10, '2023-01-10', 10, '2023-01-25', 'Approved'),
(11, 1, '2023-01-11', 11, '2023-01-26', 'Returned'),
(7, 2, '2023-01-12', NULL, NULL, 'Pending'),
(9, 3, '2023-01-13', NULL, NULL, 'Pending'),
(12, 4, '2023-01-14', 13, '2023-01-28', 'Approved'),
(1, 5, '2023-01-15', 14, '2023-01-29', 'Approved'),
(4, 6, '2023-01-16', 15, '2023-01-30', 'Returned'),
(3, 7, '2023-01-17', 1, '2023-08-30', 'Approved'),
(5, 8, '2023-01-18', NULL, NULL, 'Pending'),
(4, 9, '2023-01-19', 17, '2023-02-01', 'Returned'),
(1, 10, '2023-01-20', NULL, NULL, 'Pending');
```

3.3.4 Wdrożenie i przetestowanie bazy

Tabela Users

```
-- Sprawdzenie typów, rozmiarów oraz formatów danych dla Users
SELECT
    CASE
        WHEN TRY_CAST(UserID AS INT) IS NULL THEN 'UserID is not numeric'
        WHEN TRY_CAST(FirstName AS varchar) IS NULL THEN 'FirstName is not varchar'
        WHEN TRY_CAST(LastName AS varchar) IS NULL THEN 'LastName is not varchar'
        WHEN TRY_CAST(Login AS varchar) IS NULL THEN 'Login is not varchar'
        WHEN TRY_CAST>Password AS varchar) IS NULL THEN 'Password is not varchar'
        WHEN TRY_CAST(UserType AS varchar) IS NULL THEN 'UserID is not varchar'
        WHEN LEN(FirstName) > 255 THEN 'FirstName exceeds 255 characters'
        WHEN LEN(LastName) > 255 THEN 'LastName exceeds 255 characters'
        WHEN LEN(Login) > 255 THEN 'Login exceeds 255 characters'
        WHEN LEN>Password) > 255 THEN 'Password exceeds 255 characters'
        WHEN LEN(UserType) > 50 THEN 'UserType exceeds 50 characters'
        WHEN UserID IS NULL THEN 'UserID is NULL'
        WHEN FirstName IS NULL THEN 'FirstName is NULL'
        WHEN LastName IS NULL THEN 'LastName is NULL'
        WHEN Login IS NULL THEN 'Login is NULL'
        WHEN Password IS NULL THEN 'Password is NULL'
        WHEN UserType IS NULL THEN 'UserType is NULL'
        WHEN UserType NOT IN ('Client', 'Employee') THEN 'UserType is incorrect'
        ELSE 'Data types, formats and sizes are correct'
    END AS DataValidation
FROM Users
```

Rysunek 5: Sprawdzenie integralności semantycznej- Kod

	DataValidation
1	Data types, fromats and sizes are correct
2	Data types, fromats and sizes are correct
3	Data types, fromats and sizes are correct
4	Data types, fromats and sizes are correct
5	Data types, fromats and sizes are correct
6	Data types, fromats and sizes are correct
7	Data types, fromats and sizes are correct
8	Data types, fromats and sizes are correct
9	Data types, fromats and sizes are correct
10	Data types, fromats and sizes are correct
11	Data types, fromats and sizes are correct
12	Data types, fromats and sizes are correct
13	Data types, fromats and sizes are correct
14	Data types, fromats and sizes are correct
15	Data types, fromats and sizes are correct

Rysunek 6: Sprawdzenia integralności semantycznej - Wyniki

```
-- Sprawdzenie niepustości i unikalności wartości klucza głównego (UserID)
SELECT
  CASE
    WHEN COUNT(*) = COUNT(DISTINCT UserID) AND COUNT(UserID) = COUNT(NULLIF(UserID, NULL))
    THEN 'Primary key value (UserID) is not empty, and is unique'
    ELSE 'Primary key value (UserID) is empty, or is not unique'
  END AS PrimaryKeyIntegrity
FROM Users;
```

Rysunek 7: Sprawdzenie integralności encji: - Kod

	PrimaryKeyIntegrity
1	Primary key value (UserID) is not empty and is u...

Rysunek 8: Sprawdzenie integralności encji: - Wyniki

Tabela Resources

```
-- Sprawdzenie typów, rozmiarów oraz formatów danych dla Resources
SELECT
CASE
    WHEN TRY_CAST(ResourceID AS INT) IS NULL THEN 'ResourceID is not numeric'
    WHEN TRY_CAST(Title AS VARCHAR) IS NULL THEN 'Title is not varchar'
    WHEN TRY_CAST(Author AS VARCHAR) IS NULL THEN 'Author is not varchar'
    WHEN TRY_CAST(YearPublished AS INT) IS NULL THEN 'YearPublished is not numeric'
    WHEN TRY_CAST(ResourceType AS VARCHAR) IS NULL THEN 'ResourceType is not varchar'
    WHEN LEN(Title) > 255 THEN 'Title exceeds 255 characters'
    WHEN LEN(Author) > 255 THEN 'Author exceeds 255 characters'
    WHEN LEN(ResourceType) > 50 THEN 'ResourceType exceeds 50 characters'
    WHEN ResourceID IS NULL THEN 'ResourceID is NULL'
    WHEN Title IS NULL THEN 'Title is NULL'
    WHEN YearPublished IS NULL THEN 'YearPublished is NULL'
    WHEN ResourceType IS NULL THEN 'ResourceType is NULL'
    WHEN ResourceType NOT IN ('Book', 'Magazine', 'Article', 'Letter') THEN 'ResourceType is incorrect'
    ELSE 'Data types, fromats and sizes are correct'
END AS DataValidation
FROM Resources

-- Sprawdzenie, czy dla danego tytułu istnieją różne kombinacje rodzaju zasobu, autora i roku publikacji
SELECT
CASE
    WHEN COUNT(*) <> (
        SELECT COUNT(DISTINCT ResourceType + '|' + Author + '|' + CAST(YearPublished AS VARCHAR(10)))
        FROM Resources R2
        WHERE R2.Title = R1.Title
    )
    THEN 'Duplicate combinations of ResourceType, Author, and YearPublished found for the same title'
    ELSE 'Combinations of ResourceType, Author, and YearPublished are unique for each title'
END AS CombinationUniqueness
FROM Resources R1
GROUP BY Title;
```

Rysunek 9: Sprawdzenie integralności semantycznej- Kod

	DataValidation
1	Data types, fromats and sizes are correct
2	Data types, fromats and sizes are correct
3	Data types, fromats and sizes are correct
4	Data types, fromats and sizes are correct
5	Data types, fromats and sizes are correct
6	Data types, fromats and sizes are correct
7	Data types, fromats and sizes are correct
8	Data types, fromats and sizes are correct
9	Data types, fromats and sizes are correct
10	Data types, fromats and sizes are correct

	CombinationUniqueness
1	Combinations of Resource Type, Author, and YearPu...
2	Combinations of Resource Type, Author, and YearPu...
3	Combinations of Resource Type, Author, and YearPu...
4	Combinations of Resource Type, Author, and YearPu...
5	Combinations of Resource Type, Author, and YearPu...
6	Combinations of Resource Type, Author, and YearPu...
7	Combinations of Resource Type, Author, and YearPu...
8	Combinations of Resource Type, Author, and YearPu...
9	Combinations of Resource Type, Author, and YearPu...
10	Combinations of Resource Type, Author, and YearPu...

Rysunek 10: Sprawdzenia integralności semantycznej - Wyniki

```

-- Sprawdzenie niepustości i unikalności wartości klucza głównego (ResourceID)
SELECT
CASE
    WHEN COUNT(*) = COUNT(DISTINCT ResourceID) AND COUNT(ResourceID) = COUNT(NULLIF(ResourceID, NULL))
    THEN 'Primary key value (ResourceID) is not empty, and is unique'
    ELSE 'Primary key value (ResourceID) is empty, or is not unique'
END AS PrimaryKeyIntegrity
FROM Resources;

```

Rysunek 11: Sprawdzenie integralności encji: - Kod

	PrimaryKeyIntegrity
1	Primary key value (ResourceID) is not empty, an...

Rysunek 12: Sprawdzenie integralności encji: - Wyniki

Tabela ResourcesCopies

```

-- Sprawdzenie typów, rozmiarów oraz formatów danych dla ResourcesCopies
SELECT
CASE
    WHEN TRY_CAST(ResourceID AS INT) IS NULL THEN 'ResourceID is not numeric'
    WHEN TRY_CAST(CopyID AS INT) IS NULL THEN 'CopyID is not numeric'
    WHEN ResourceID IS NULL THEN 'ResourceID is NULL'
    WHEN CopyID IS NULL THEN 'CopyID is NULL'
    ELSE 'Data types, fromats and sizes are correct'
END AS DataValidation
FROM ResourceCopies

```

Rysunek 13: Sprawdzenie integralności semantycznej- Kod

	DataValidation
1	Data types, fromats and sizes are correct
2	Data types, fromats and sizes are correct
3	Data types, fromats and sizes are correct
4	Data types, fromats and sizes are correct
5	Data types, fromats and sizes are correct
6	Data types, fromats and sizes are correct
7	Data types, fromats and sizes are correct
8	Data types, fromats and sizes are correct
9	Data types, fromats and sizes are correct
10	Data types, fromats and sizes are correct
11	Data types, fromats and sizes are correct
12	Data types, fromats and sizes are correct
13	Data types, fromats and sizes are correct
14	Data types, fromats and sizes are correct
15	Data types, fromats and sizes are correct
16	Data types, fromats and sizes are correct
17	Data types, fromats and sizes are correct
18	Data types, fromats and sizes are correct
19	Data types, fromats and sizes are correct
20	Data types, fromats and sizes are correct
21	Data types, fromats and sizes are correct
22	Data types, fromats and sizes are correct
23	Data types, fromats and sizes are correct
24	Data types, fromats and sizes are correct
25	Data types, fromats and sizes are correct
26	Data types, fromats and sizes are correct
27	Data types, fromats and sizes are correct
28	Data types, fromats and sizes are correct
29	Data types, fromats and sizes are correct
30	Data types, fromats and sizes are correct
31	Data types, fromats and sizes are correct
32	Data types, fromats and sizes are correct
33	Data types, fromats and sizes are correct
34	Data types, fromats and sizes are correct
35	Data types, fromats and sizes are correct
36	Data types, fromats and sizes are correct
37	Data types, fromats and sizes are correct
38	Data types, fromats and sizes are correct
39	Data types, fromats and sizes are correct
40	Data types, fromats and sizes are correct
41	Data types, fromats and sizes are correct

Rysunek 14: Sprawdzenia integralności semantycznej - Wyniki

```
-- Sprawdzenie niepustości i unikalności wartości klucza głównego (CopyID)
SELECT
  CASE
    WHEN COUNT(*) = COUNT(DISTINCT CopyID) AND COUNT(CopyID) = COUNT(NULLIF(CopyID, NULL))
    THEN 'Primary key value (CopyID) is not empty, and is unique'
    ELSE 'Primary key value (CopyID) is empty, or is not unique'
  END AS PrimaryKeyIntegrity
FROM ResourceCopies;
```

Rysunek 15: Sprawdzenie integralności encji: - Kod

	PrimaryKeyIntegrity
1	Primary key value (CopyID) is not empty, and is ...

Rysunek 16: Sprawdzenie integralności encji: - Wyniki

```
-- Sprawdzenie integralności referencji w tabeli ResourceCopies
SELECT
  CASE
    WHEN COUNT(*) = COUNT(NULLIF(Resources.ResourceID, NULL))
    THEN 'Foreign key references are valid'
    ELSE 'Foreign key references are not valid'
  END AS ForeignKeyIntegrity
FROM ResourceCopies
LEFT JOIN Resources ON ResourceCopies.ResourceID = Resources.ResourceID;
```

Rysunek 17: Sprawdzenie integralności referencji: - Kod

	ForeignKeyIntegrity
1	Foreign key references are valid

Rysunek 18: Sprawdzenie integralności referencji: - Wyniki

Tabela BorrowRequests

```
-- Sprawdzenie typów, rozmiarów oraz formatów danych dla BorrowRequests
SELECT
CASE
WHEN TRY_CAST(RequestID AS INT) IS NULL THEN 'RequestID is not numeric'
WHEN TRY_CAST(UserID AS INT) IS NULL THEN 'UserID is not numeric'
WHEN TRY_CAST(ResourceID AS INT) IS NULL THEN 'ResourceID is not numeric'
WHEN TRY_CAST(RequestDate AS DATE) IS NULL THEN 'RequestDate is not a valid date'
WHEN TRY_CAST(CopyID AS INT) IS NULL AND CopyID IS NOT NULL THEN 'CopyID is not numeric'
WHEN TRY_CAST(DueDate AS DATE) IS NULL AND DueDate IS NOT NULL THEN 'DueDate is not a valid date'
WHEN TRY_CAST(Status AS VARCHAR) IS NULL THEN 'Status is not varchar'
WHEN LEN(Status) > 50 THEN 'Status exceeds 50 characters'
WHEN RequestID IS NULL THEN 'RequestID is NULL'
WHEN UserID IS NULL THEN 'UserID is NULL'
WHEN ResourceID IS NULL THEN 'ResourceID is NULL'
WHEN RequestDate IS NULL THEN 'RequestDate is NULL'
WHEN Status NOT IN ('Pending', 'Approved', 'Returned') THEN 'Status is incorrect'
ELSE 'Data types, fromats and sizes are correct'
END AS DataValidation
FROM BorrowRequests;
```

Rysunek 19: Sprawdzenie integralności semantycznej- Kod

	DataValidation
1	Data types, fromats and sizes are correct
2	Data types, fromats and sizes are correct
3	Data types, fromats and sizes are correct
4	Data types, fromats and sizes are correct
5	Data types, fromats and sizes are correct
6	Data types, fromats and sizes are correct
7	Data types, fromats and sizes are correct
8	Data types, fromats and sizes are correct
9	Data types, fromats and sizes are correct
10	Data types, fromats and sizes are correct
11	Data types, fromats and sizes are correct
12	Data types, fromats and sizes are correct
13	Data types, fromats and sizes are correct
14	Data types, fromats and sizes are correct
15	Data types, fromats and sizes are correct
16	Data types, fromats and sizes are correct
17	Data types, fromats and sizes are correct
18	Data types, fromats and sizes are correct
19	Data types, fromats and sizes are correct
20	Data types, fromats and sizes are correct

Rysunek 20: Sprawdzenia integralności semantycznej - Wyniki

```
-- Sprawdzenie niepustości i unikalności wartości klucza głównego (RequestID) w BorrowRequests
SELECT
CASE
WHEN COUNT(*) = COUNT(DISTINCT RequestID) AND COUNT(RequestID) = COUNT(NULLIF(RequestID, NULL))
THEN 'Primary key value (RequestID) is not empty, and is unique'
ELSE 'Primary key value (RequestID) is empty, or is not unique'
END AS PrimaryKeyIntegrity
FROM BorrowRequests;
```

Rysunek 21: Sprawdzenie integralności encji: - Kod

	PrimaryKeyIntegrity
1	Primary key value (RequestID) is not empty, and ...

Rysunek 22: Sprawdzenie integralności encji: - Wyniki

```
-- Sprawdzenie, czy UserID istnieje w tabeli Users
SELECT
CASE
WHEN COUNT(*) <> COUNT(NULLIF(Users.UserID, NULL))
THEN 'UserID references are not valid'
ELSE 'UserID references are valid'
END AS UserIDReferences
FROM BorrowRequests
LEFT JOIN Users ON BorrowRequests.UserID = Users.UserID;

-- Sprawdzenie, czy CopyID istnieje w tabeli ResourceCopies
SELECT
CASE
WHEN COUNT(*) <> COUNT(NULLIF(ResourceCopies.CopyID, NULL))
THEN 'CopyID references are not valid'
ELSE 'CopyID references are valid'
END AS CopyIDReferences
FROM BorrowRequests
LEFT JOIN ResourceCopies ON BorrowRequests.CopyID = ResourceCopies.CopyID;

-- Sprawdzenie, czy ResourceID istnieje w tabeli Resources
SELECT
CASE
WHEN COUNT(*) <> COUNT(NULLIF(Resources.ResourceID, NULL))
THEN 'ResourceID references are not valid'
ELSE 'ResourceID references are valid'
END AS ResourceIDReferences
FROM BorrowRequests
LEFT JOIN Resources ON BorrowRequests.ResourceID = Resources.ResourceID;
```

Rysunek 23: Sprawdzenie integralności referencji: - Kod

	UserIDReferences
1	UserID references are valid
	CopyIDReferences
1	CopyID references are not valid
	ResourceIDReferences
1	ResourceID references are valid

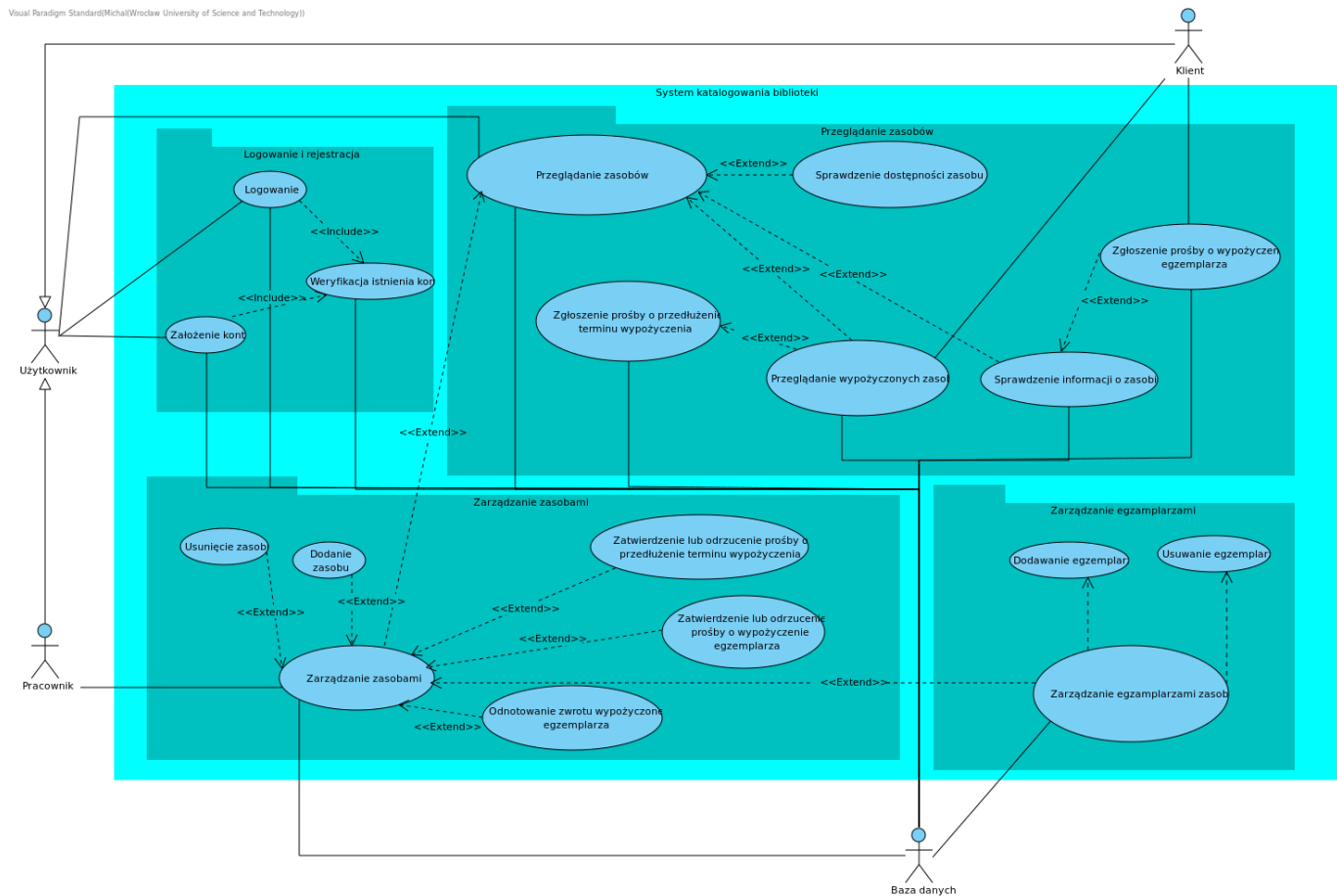
Rysunek 24: Sprawdzenie integralności referencji: - Wyniki

4 Implementacja aplikacji

4.1 Zmiany w diagramie przypadków użycia

Podczas projektowania interfejsu aplikacji dokonaliśmy kilku zmian w diagramie przypadków użycia. Zmiany te miały na celu polepszenie komfortu użytkownika.

Zmienione zostały sekcje zarządzania prośbami o wypożyczenie zasobów.



Rysunek 25: Nowy diagram przypadków użycia.

4.2 Makieta interfejsu graficznego



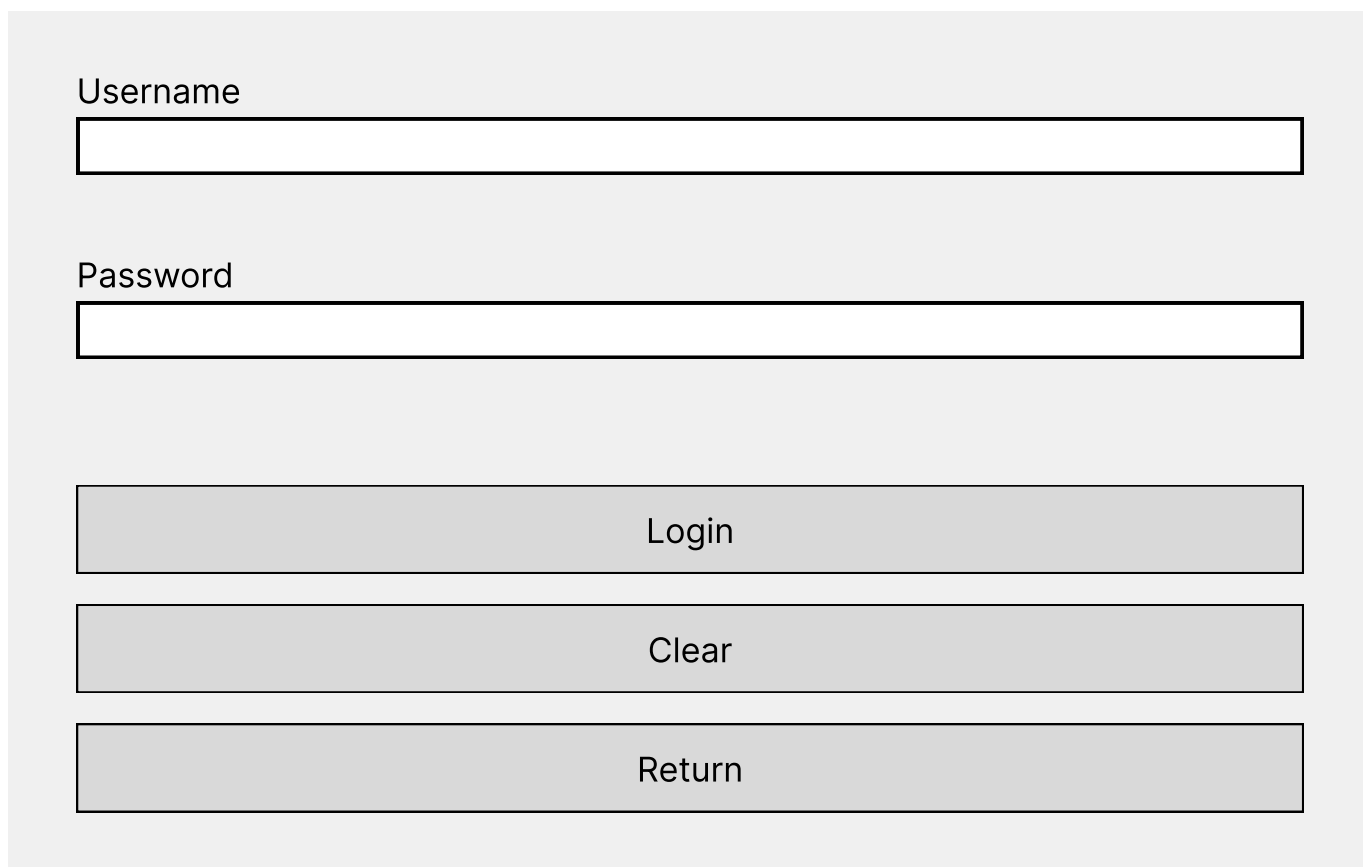
A mockup of a library welcome screen. It features a large, bold, black title "Library" centered at the top. Below the title, there are two rectangular buttons with black borders and light gray backgrounds. The left button is labeled "Add New Copy" and the right button is labeled "Sign Up".

Library

Add New Copy

Sign Up

Rysunek 26: Makieta widoku powitalnego.



A mockup of a library login screen. It features a light gray background. At the top, the label "Username" is followed by a white rectangular input field with a black border. Below this, the label "Password" is followed by another white rectangular input field with a black border. At the bottom, there are three stacked rectangular buttons with black borders and light gray backgrounds. The top button is labeled "Login", the middle button is labeled "Clear", and the bottom button is labeled "Return".

Username

Password

Login

Clear

Return

Rysunek 27: Makieta widoku logowania.

First Name

Last Name

Username

Password

Confirm Password

Key (optional)

Register

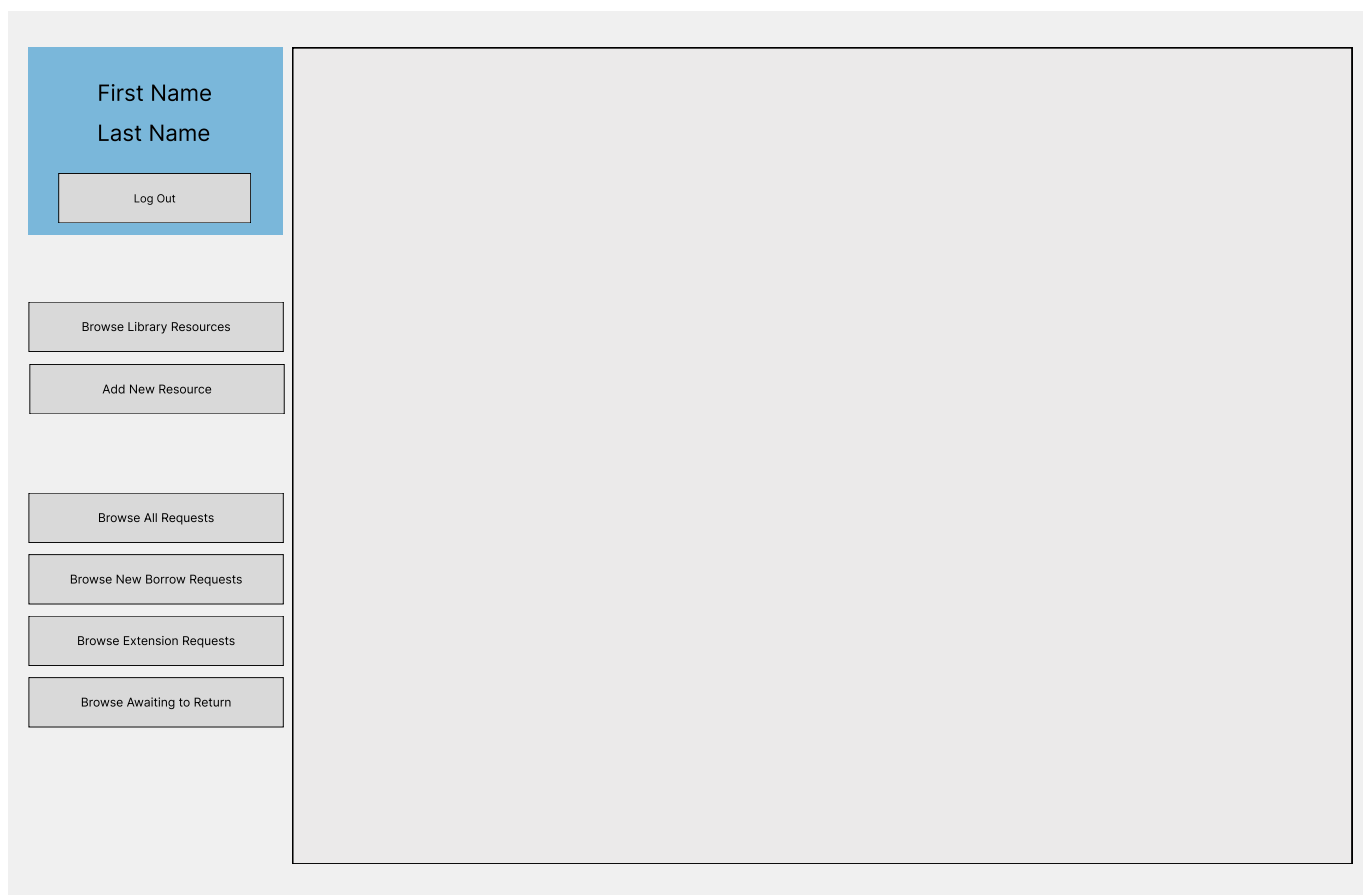
Clear

Return

Rysunek 28: Makieta widoku rejestracji.

Browse Library Resources		First Name Last Name	Log Out
Borrowed By Me	My Requests		

Rysunek 29: Makieta widoku klienta.



Rysunek 30: Makieta widoku pracownika.

Title

Author

Year Published

Resource Type

Add Resource Button

Rysunek 31: Makieta widoku dodawania zasobu.

ResourceID:

Title:

Author:

Year Published:

Type:

Add New Copy

Return

Rysunek 32: Makieta widoku zarządzania kopiami zasobu.

Copy ID:

◀

January 2024

▶

Accept

Rysunek 33: Makieta widoku akceptacji prośby o wypożyczenie.

Select Date

◀

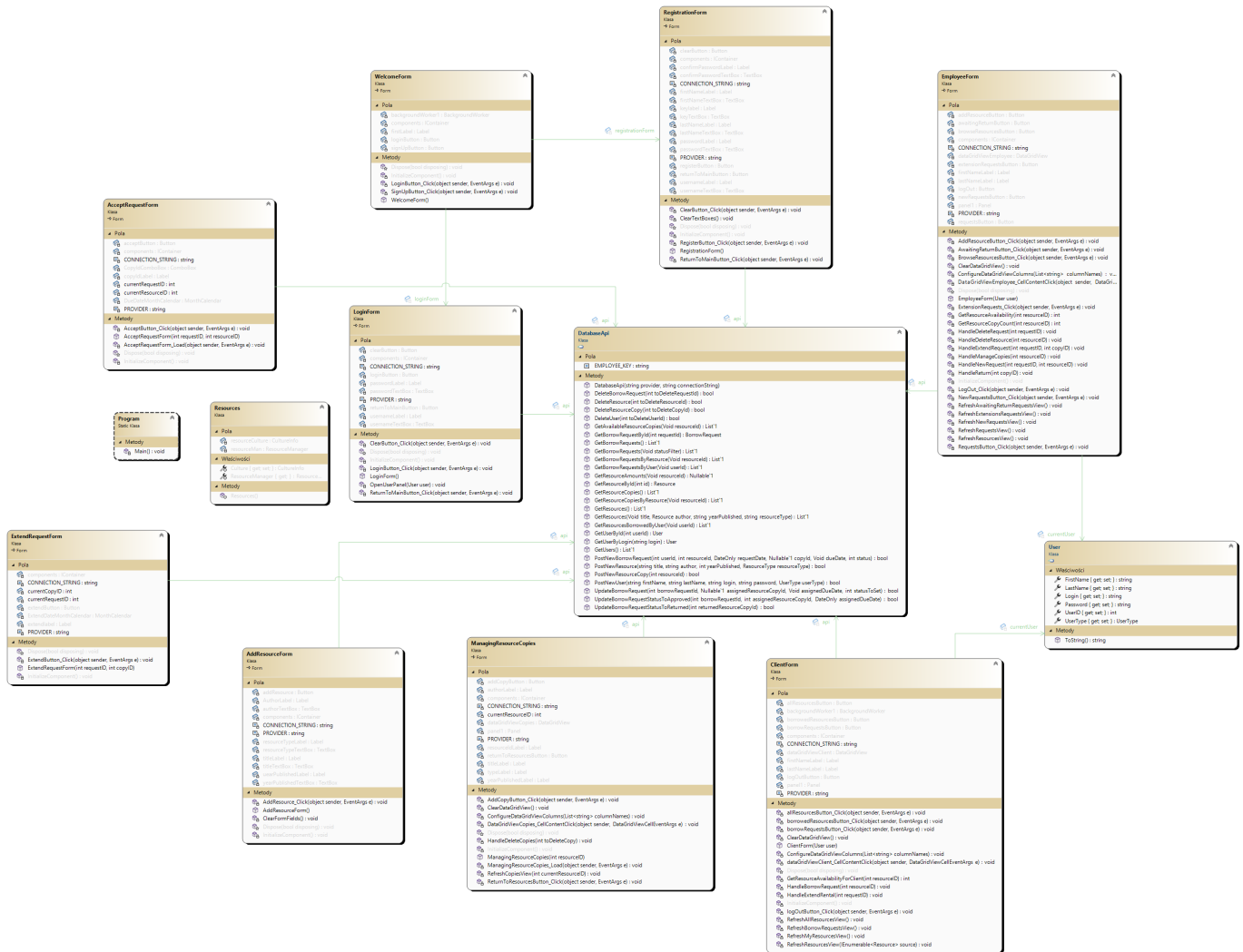
January 2024

▶

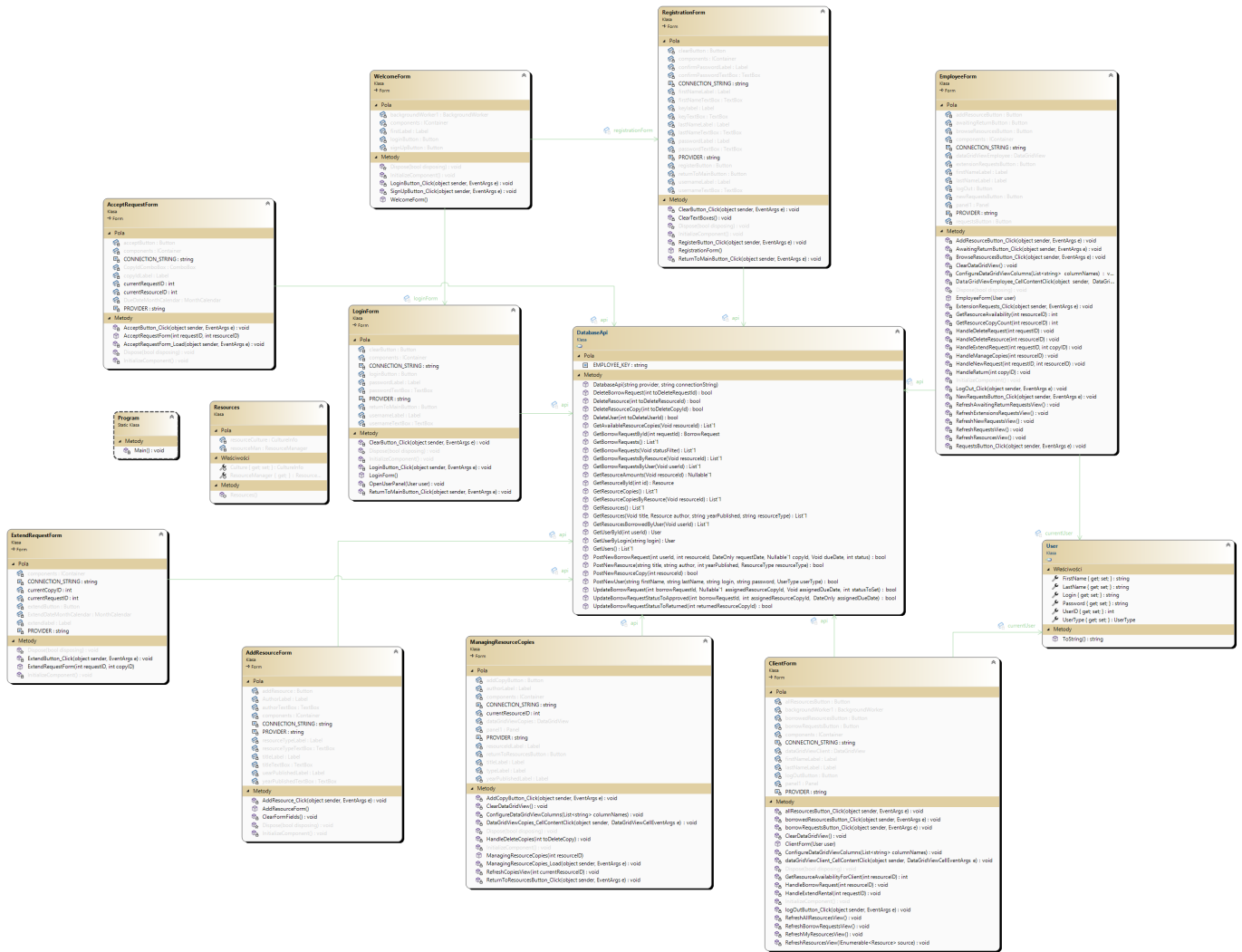
Extend

Rysunek 34: Makieta widoku przedłużenia prośby o wypożyczenie.

4.3 Diagram klas



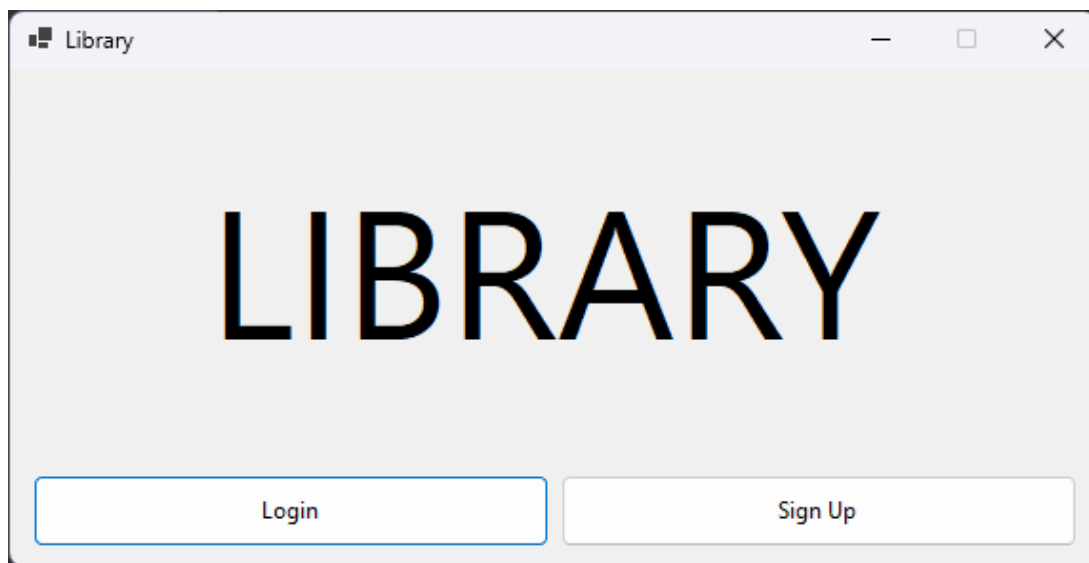
Rysunek 35: Diagram klas dla Front-End'u.



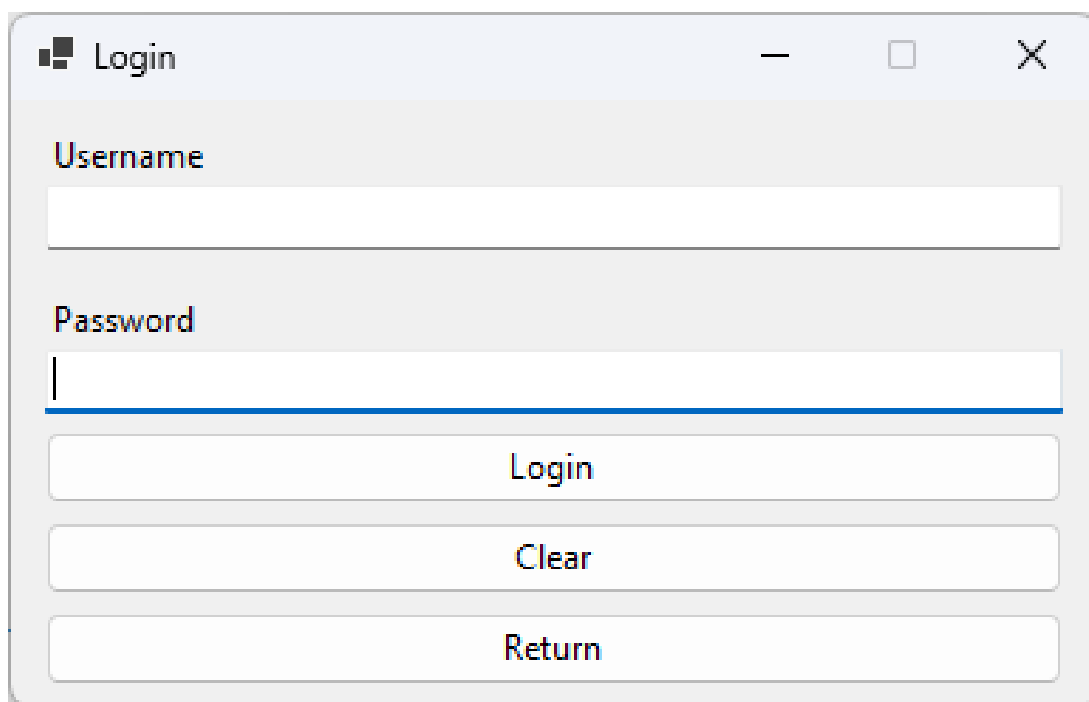
Rysunek 36: Diagram klas dla Back-End'u.

4.4 Wdrożenie aplikacji

4.4.1 Okna startowe



Rysunek 37: Okno powitalne.



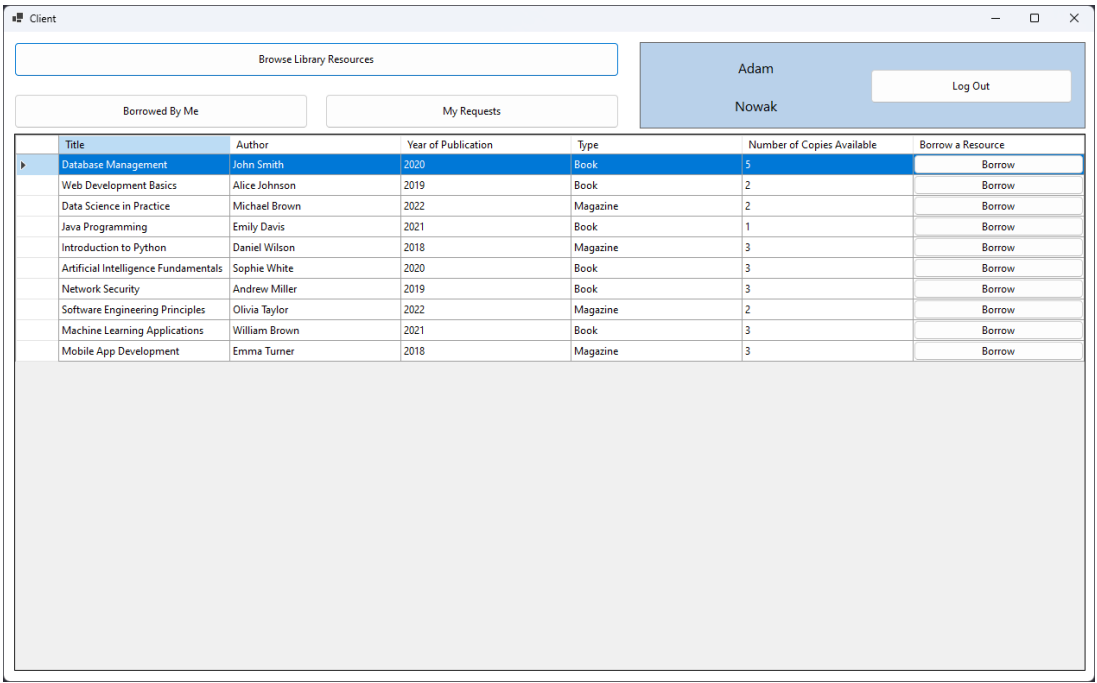
Rysunek 38: Okno logowania.

The image shows a graphical user interface for a registration window titled "Register". The window has a standard title bar with a minimize button, a maximize button (disabled), and a close button. The main area contains six text input fields, each preceded by a label: "First Name", "Last Name", "Username", "Password", "Confirm Password", and "Key (optional)". Below the input fields are three buttons: "Regsiter" (with a blue border), "Clear", and "Return".

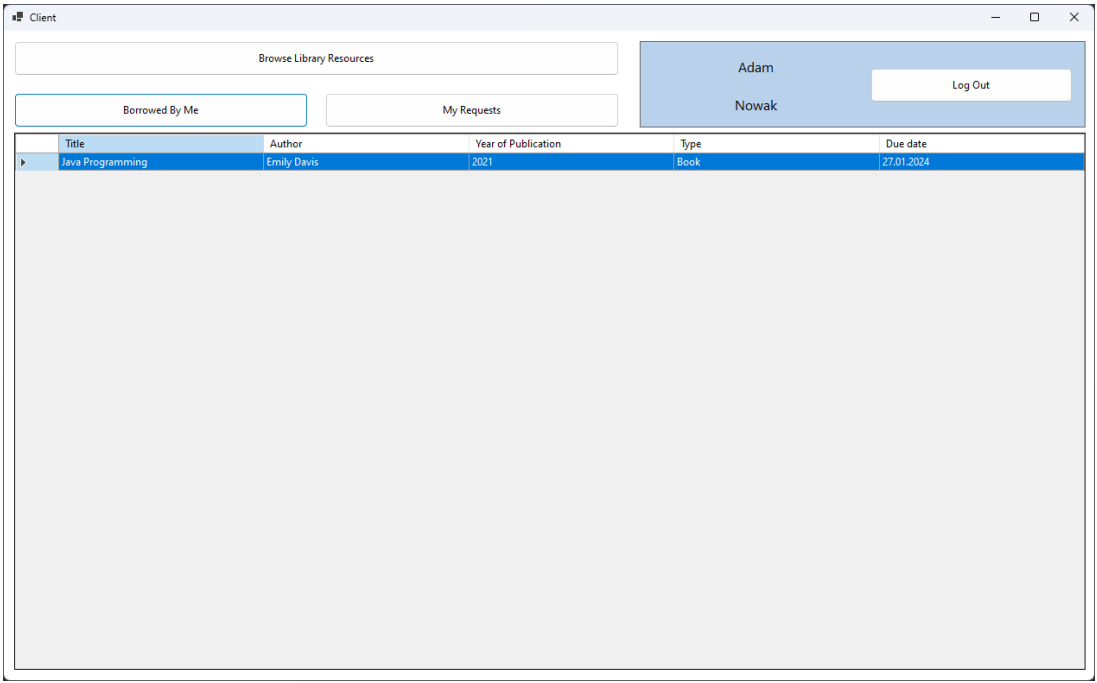
Field Label	Field Type
First Name	Text Input
Last Name	Text Input
Username	Text Input
Password	Text Input
Confirm Password	Text Input
Key (optional)	Text Input
Regsiter	Submit Button
Clear	Reset Button
Return	Cancel Button

Rysunek 39: Okno rejestracji.

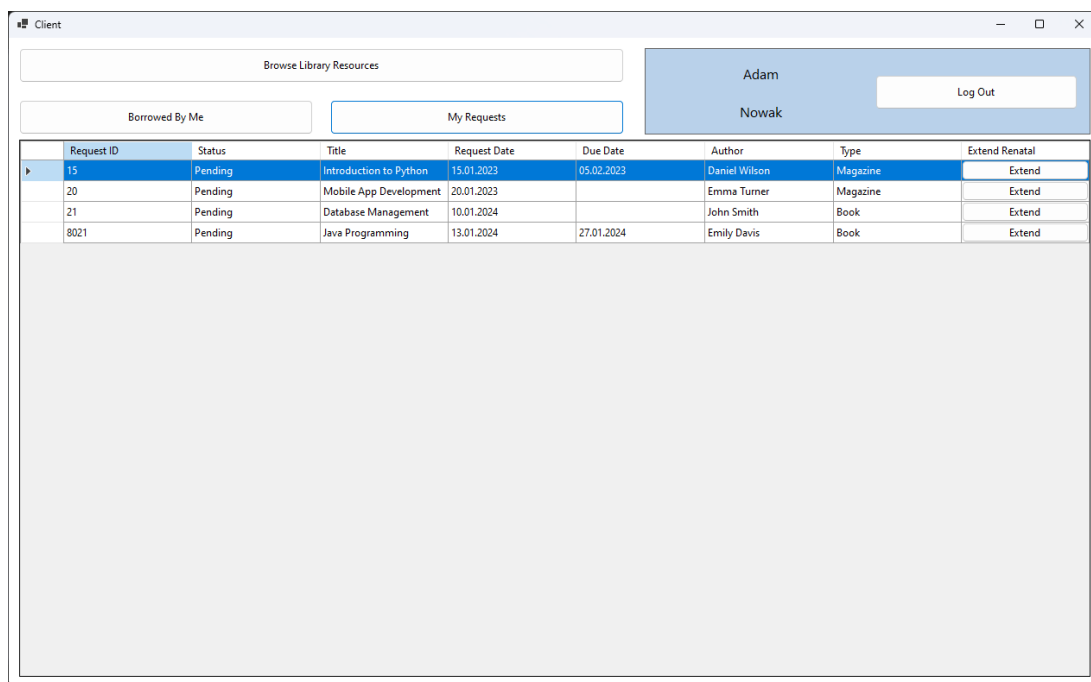
4.4.2 Okna klienta



Rysunek 40: Okno przeglądania zbioru biblioteki.

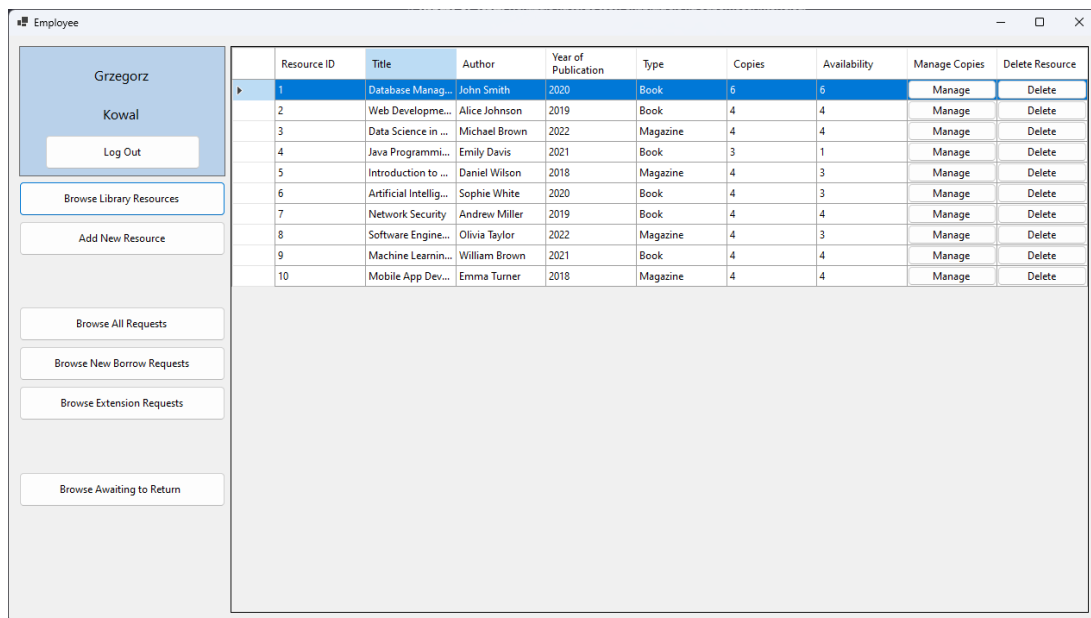


Rysunek 41: Okno przeglądania pożyczonych pozycji.



Rysunek 42: Okno przeglądania próśb o wypożyczenie.

4.4.3 Okna pracownika



Rysunek 43: Okno przeglądania zasobów biblioteki.

Resource Copies

Resource ID: 1

Title: Database Management

Author: John Smith

Year Published: 2020

Type: Book

Add New Copy

Return

CopyID	Status	Due Date	Delete Copy
1	Ready		Delete
2	Ready		Delete
12	Ready		Delete
22	Ready		Delete
32	Ready		Delete
11042	Ready		Delete

Rysunek 44: Okno zarządzania egzemplarzami zasobu.

Add New Resource

Title

Author

Year Published

Resource Type

Add Resource

Rysunek 45: Okno dodawania nowego zasobu.

Employee							
<div>Grzegorz</div> <div>Kowal</div> <div>Log Out</div> <div>Browse Library Resources</div> <div>Add New Resource</div> <div>Browse All Requests</div> <div>Browse New Borrow Requests</div> <div>Browse Extension Requests</div> <div>Browse Awaiting to Return</div>							
Resource ID	Copy ID	ResourceTitle	Request Date	Due Date	Status	Delete Request	
21		Database Management	10.01.2024		Pending	Delete	
11	11	Database Management	11.01.2023	26.01.2023	Returned	Delete	
12		Web Development Ba...	12.01.2023		Pending	Delete	
2		Web Development Ba...	02.01.2023		Pending	Delete	
13		Data Science in Practice	13.01.2023		Pending	Delete	
3		Data Science in Practice	03.01.2023		Pending	Delete	
8021	15	Java Programming	13.01.2024	27.01.2024	Pending	Delete	
14	13	Java Programming	14.01.2023	28.01.2023	Approved	Delete	
4	4	Java Programming	04.01.2023	15.01.2024	Returned	Delete	
15	14	Introduction to Python	15.01.2023	05.02.2023	Pending	Delete	
16	15	Artificial Intelligence F...	16.01.2023	30.01.2023	Returned	Delete	
6	6	Artificial Intelligence F...	06.01.2023	21.01.2023	Approved	Delete	
17	1	Network Security	17.01.2023	30.08.2023	Returned	Delete	
7		Network Security	07.01.2023		Pending	Delete	
18		Software Engineering ...	18.01.2023		Pending	Delete	
8	11	Software Engineering ...	08.01.2023	26.01.2024	Approved	Delete	
19	17	Machine Learning Ap...	19.01.2023	01.02.2023	Returned	Delete	
9		Machine Learning Ap...	09.01.2023		Pending	Delete	
20		Mobile App Develop...	20.01.2023		Pending	Delete	
10	10	Mobile App Develop...	10.01.2023	25.01.2023	Returned	Delete	

Rysunek 46: Okno przeglądania wszystkich próśb o wypożyczenie.

Employee					
<div>Grzegorz</div> <div>Kowal</div> <div>Log Out</div> <div>Browse Library Resources</div> <div>Add New Resource</div> <div>Browse All Requests</div> <div>Browse New Borrow Requests</div> <div>Browse Extension Requests</div> <div>Browse Awaiting to Return</div>					
Resource ID	Title	Request Date	Status	Consider Borrow Request	
2	Web Development Basics	02.01.2023	Pending	Consider	
3	Data Science in Practice	03.01.2023	Pending	Consider	
7	Network Security	07.01.2023	Pending	Consider	
9	Machine Learning Applications	09.01.2023	Pending	Consider	
12	Web Development Basics	12.01.2023	Pending	Consider	
13	Data Science in Practice	13.01.2023	Pending	Consider	
18	Software Engineering Principles	18.01.2023	Pending	Consider	
20	Mobile App Development	20.01.2023	Pending	Consider	
21	Database Management	10.01.2024	Pending	Consider	

Rysunek 47: Okno przeglądania nowych próśb o wypożyczenie.

Accept New Request

Copy ID: 3

styczeń 2024

pon.	wt.	śr.	czw.	pt.	sob.	niedz.
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Dziś: 16.01.2024

Accept

Rysunek 48: Okno akceptacji prośby o wypożyczenie.

Employee

Grzegorz Kowal

Log Out

Browse Library Resources

Add New Resource

Browse All Requests

Browse New Borrow Requests

Browse Extension Requests

Browse Awaiting to Return

Resource ID	Copy ID	Title	Request Date	Due Date	Status	Consider Extension
15	14	Introduction to Python	15.01.2023	05.02.2023	Pending	Consider
8021	15	Java Programming	13.01.2024	27.01.2024	Pending	Consider

Rysunek 49: Okno przeglądania próśb o przedłużenie wypożyczenia.

Extend Request

Select Date

styczeń 2024

pon.	wt.	śr.	czw.	pt.	sob.	niedz.
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Dziś: 16.01.2024

Extend

Rysunek 50: Okno akceptacji prośby o przedłużenie wypożyczenia.

Employee

Grzegorz Kowal

Log Out

Browse Library Resources

Add New Resource

Browse All Requests

Browse New Borrow Requests

Browse Extension Requests

Browse Awaiting to Return

Resource ID	Copy ID	Title	Request Date	Due Date	Status	Return a Copy
6	6	Artificial Intelligence F...	06.01.2023	21.01.2023	Approved	Return
8	11	Software Engineering ...	08.01.2023	26.01.2024	Approved	Return
14	13	Java Programming	14.01.2023	28.01.2023	Approved	Return

Rysunek 51: Okno przeglądania wypożyczonych egzemplarzy (oczekujących na zwrot).

4.5 Testowanie aplikacji

4.5.1 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);

```

```

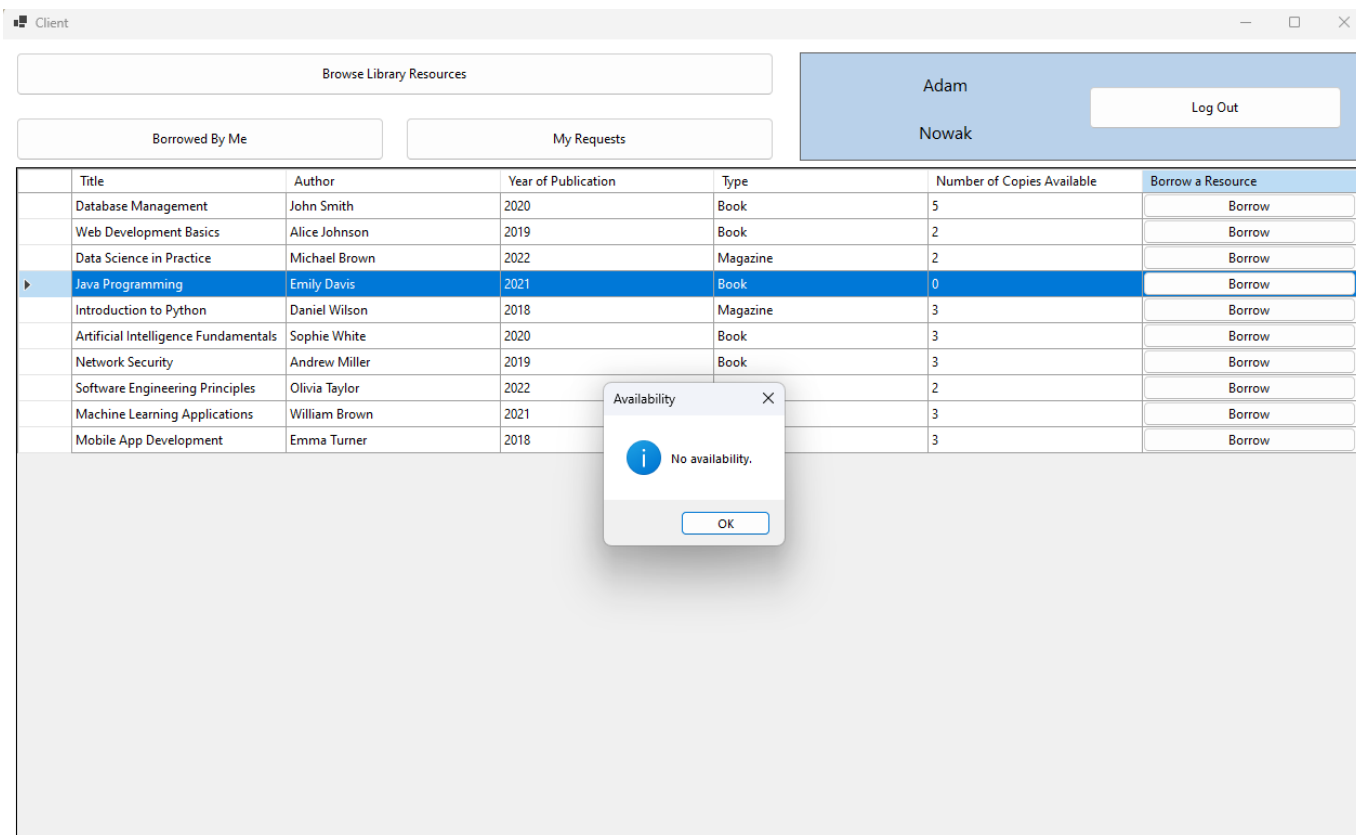
    Assert.NotNull(added);

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

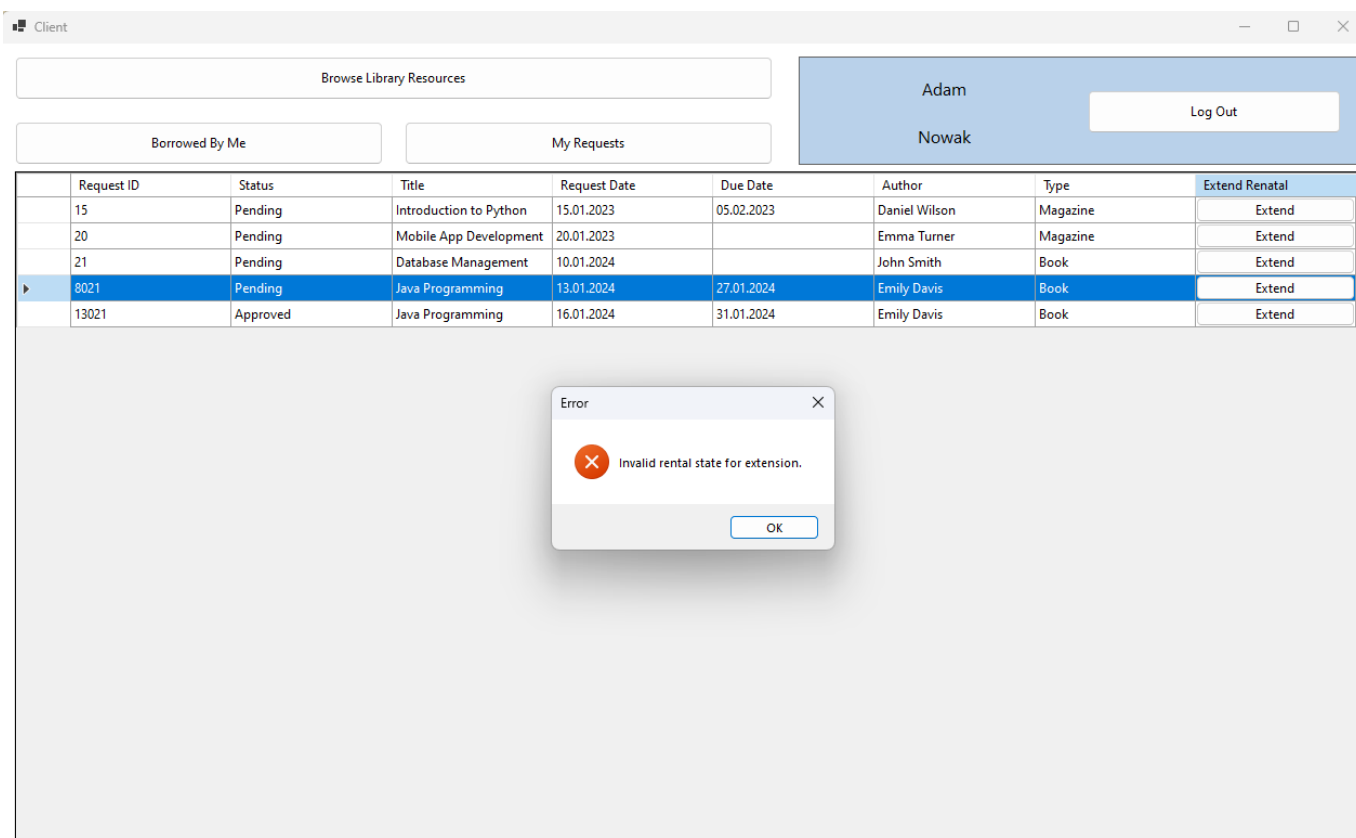
    var newNewRes = api.GetResources();
    Assert.NotNull(newNewRes);
    Assert.Equal(res.Count(), newNewRes.Count());
}
}

```

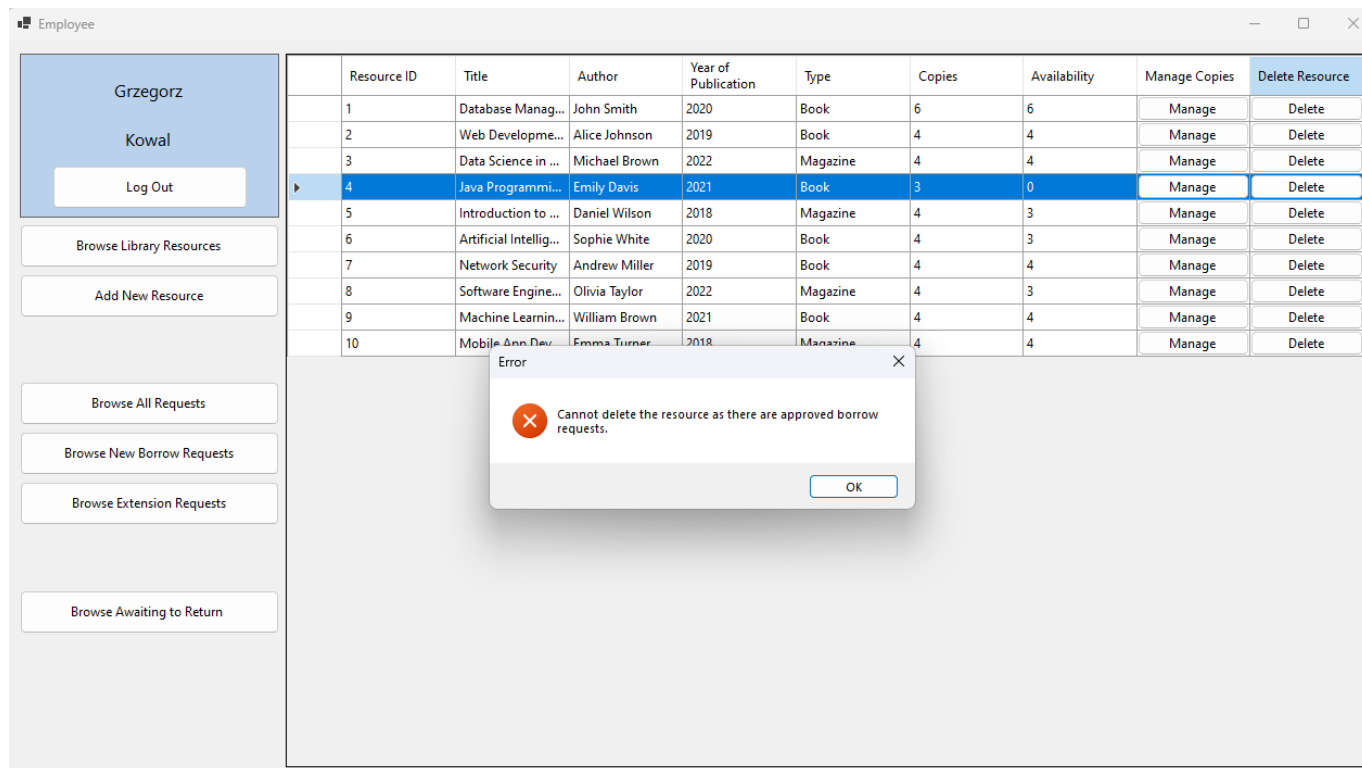
4.5.2 Testy GUI



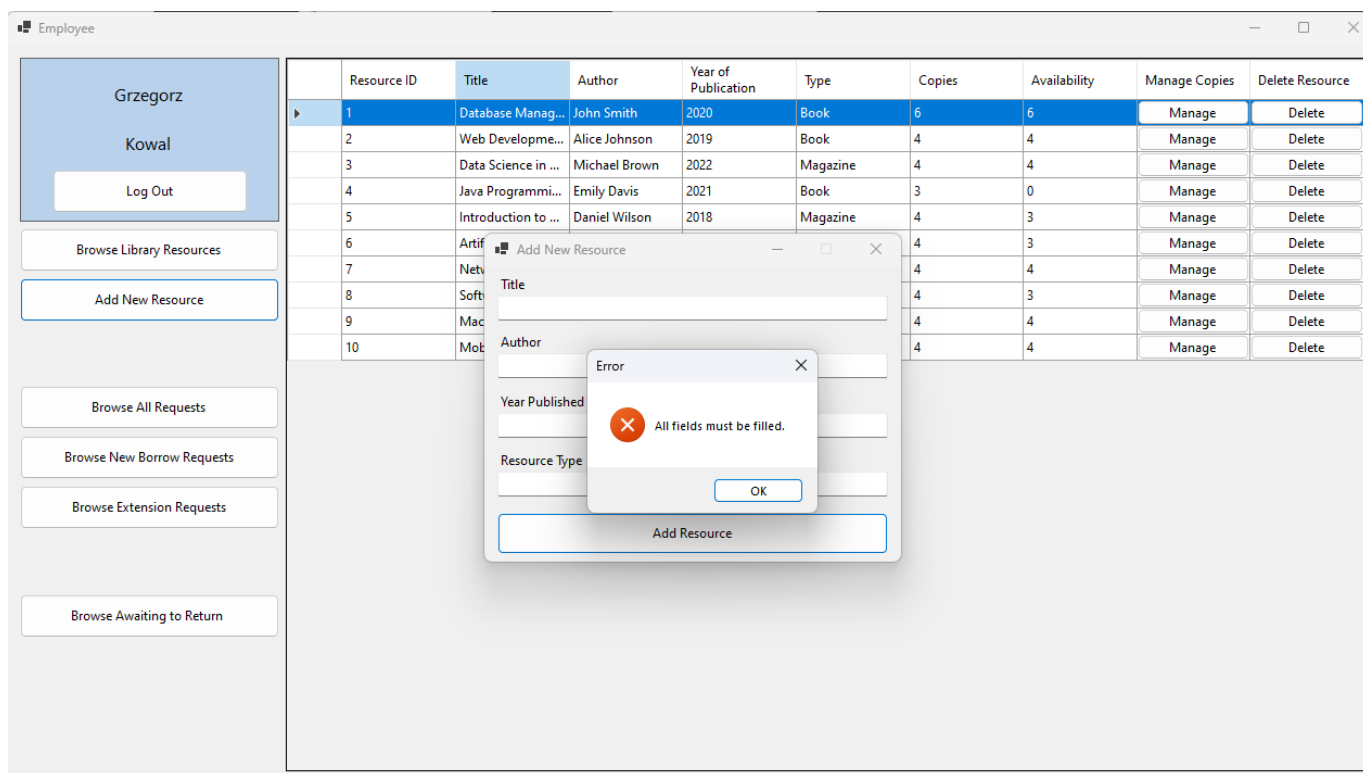
Rysunek 52: Test 1: Próba wypożyczenia zasobu, dla którego nie ma dostępnych kopii



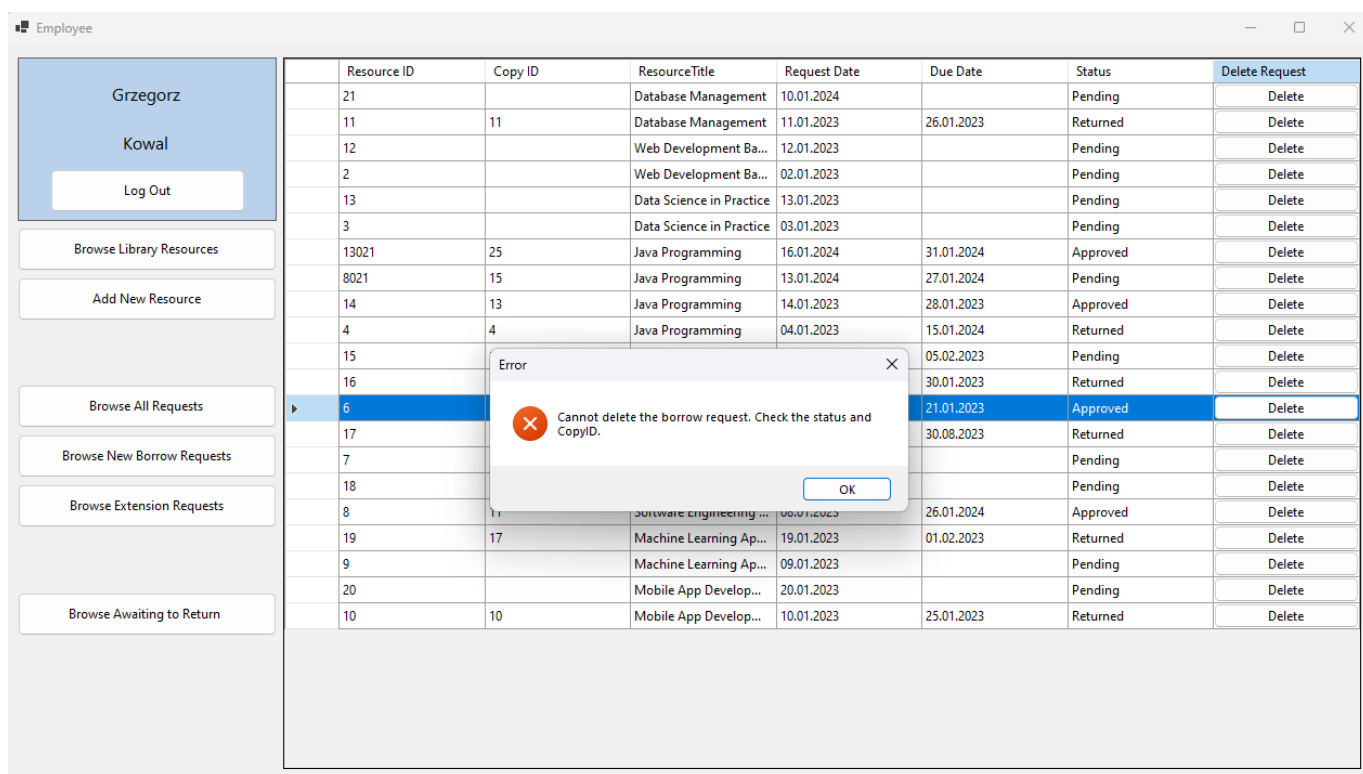
Rysunek 53: Test 2: Próba przedłużenia wypożyczenia egzemplarza, który nie jest wypożyczony przez użytkownika



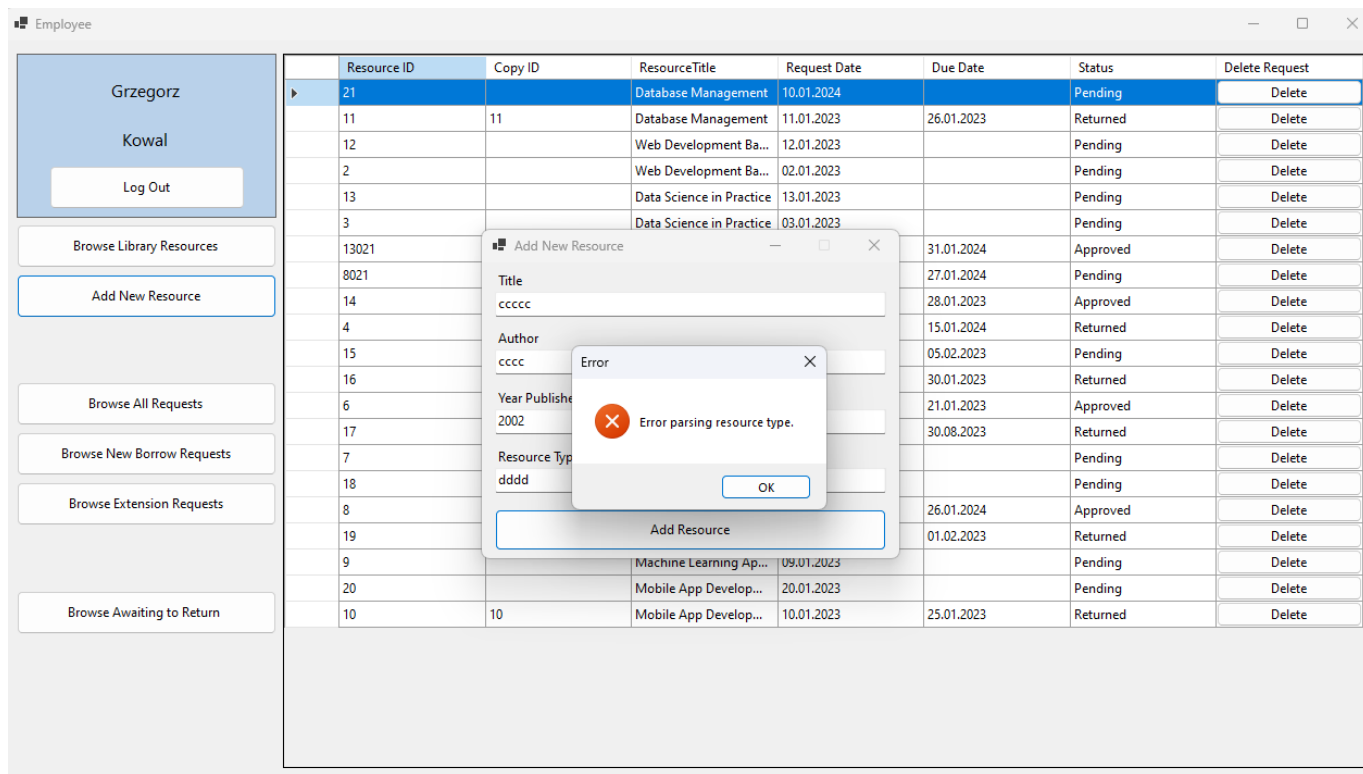
Rysunek 54: Test 3: Próba usunięcia zasobu, którego egzemplarz jest aktualnie wypożyczony klientowi



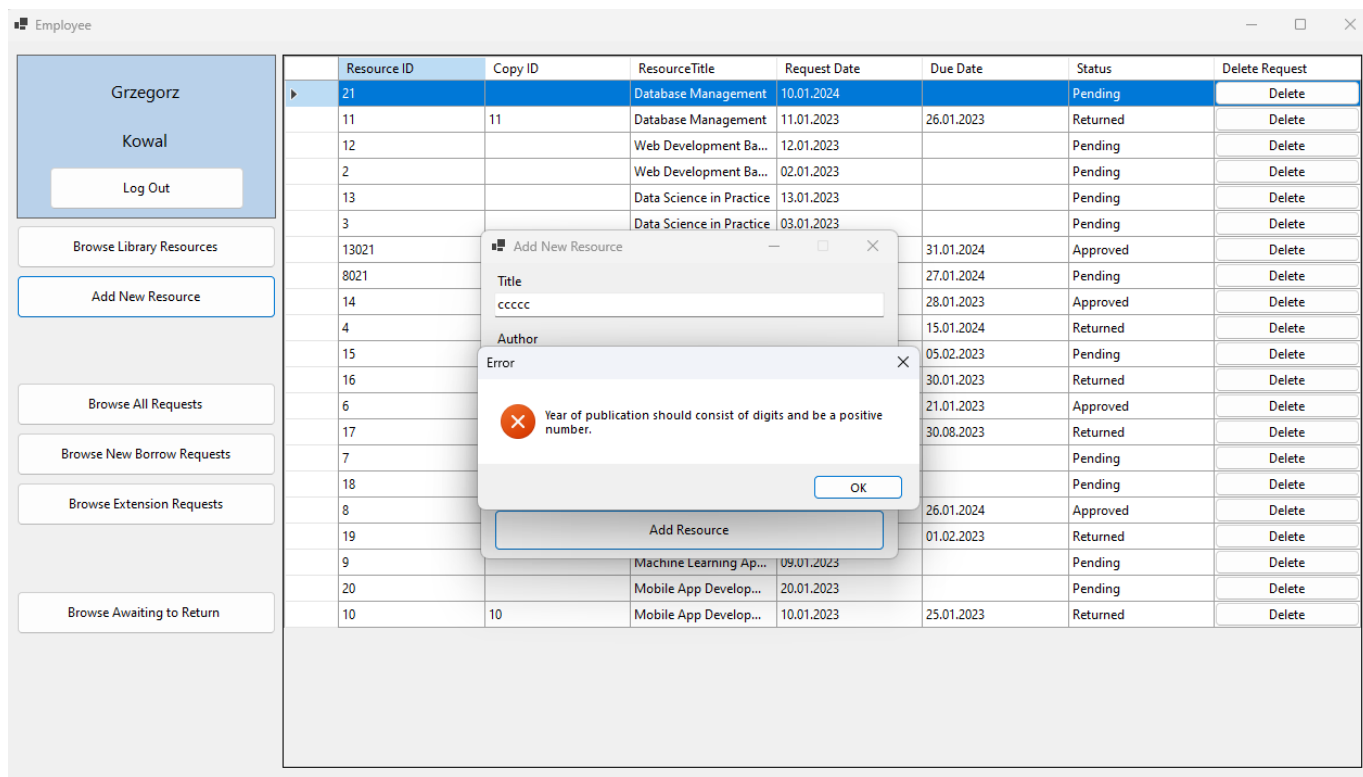
Rysunek 55: Test 4: Próba dodania zasobu bez uzupełnienia pól formularza



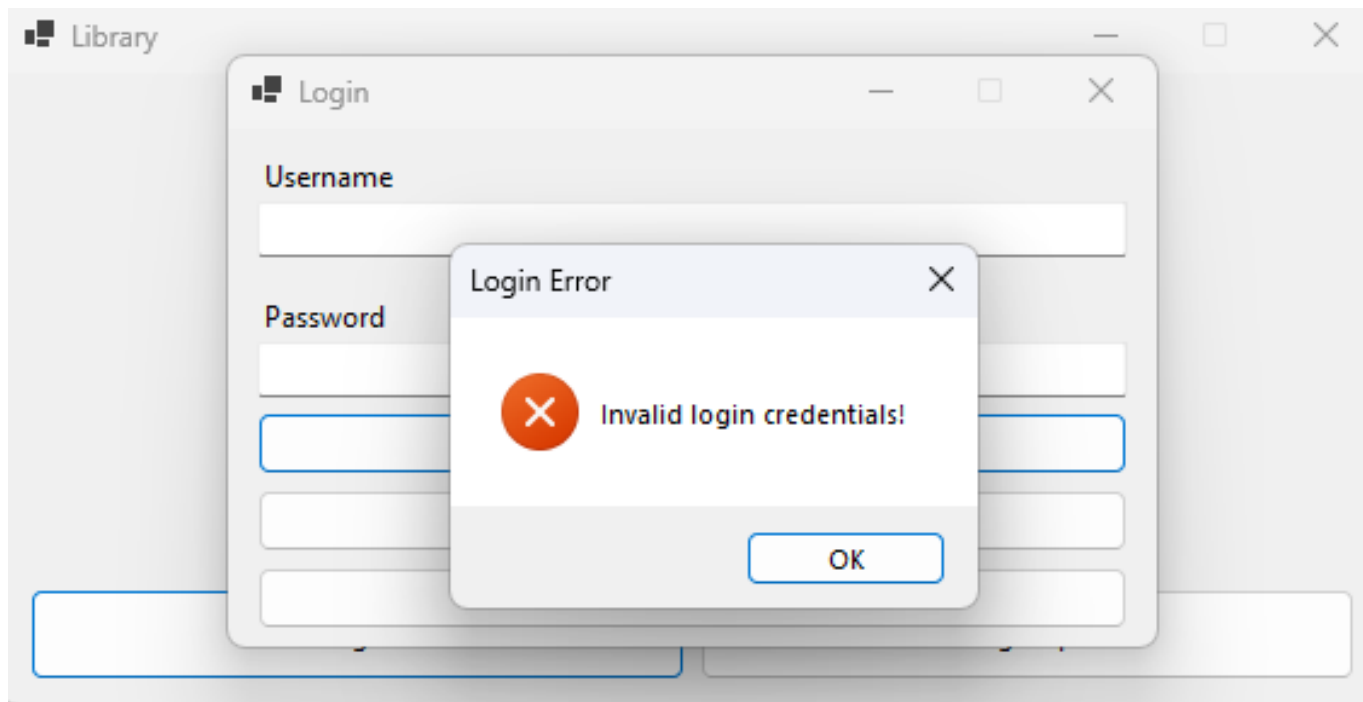
Rysunek 56: Test 5: Próba usunięcie prośby o wypożyczenie dla niezwróconej kopii.



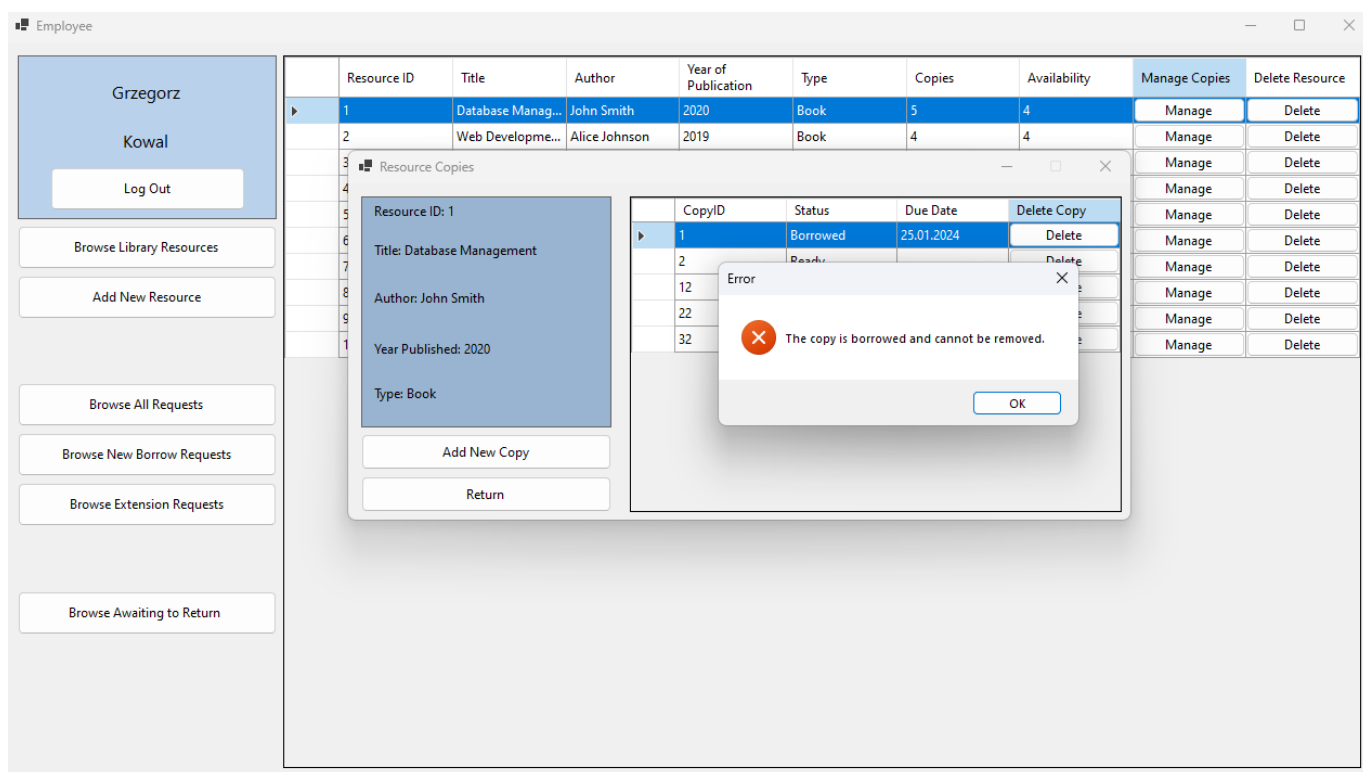
Rysunek 57: Test 6: Próba dodanie zasobu o nieznanym typie.



Rysunek 58: Test 7: Próba dodanie zasobu o niepoprawnej dacie wydania.



Rysunek 59: Test 8: Próba zalogowania bez podania danych



Rysunek 60: Test 9: Próba usunięcia wypożyczonej kopii zasobu.

The image shows a web application window titled "Register". It contains a registration form with the following fields: "First Name", "Last Name", "Username", "Password", "Confirm Password", and "Key (optional)". Each field is represented by a text input box. Below the form are three buttons: "Register", "Clear", and "Return". An error dialog box is displayed in the center of the window, titled "Error". It features a red circular icon with a white 'X' and the text "Fill in all required fields." Below the message is an "OK" button.

Rysunek 61: Test 10: Próba rejestracji konta bez uzupełnionego formularza.