**AKADEMIA NAUK STOSOWANYCH W ELBLĄGU**Instytut Informatyki Stosowanej im. K. Brzeskiego

**PRACA DYPLOMOWA**

**inżynierska**

**Tomasz Rycko**

PROJEKT I IMPLEMENTACJA SYSTEMU DO OBSŁUGI CZYTELNI ELEKTRONICZNEJ

**Promotor:   
dr inż. Stanisław Witkowski**

****

Elbląg, 2024



**Akademia nauk stosowanych   
w Elblągu**

**Instytut Informatyki Stosowanej  
im. Krzysztofa Brzeskiego**

Imię i nazwisko dyplomanta: **Tomasz Rycko**

Nr albumu: **19744**

Rodzaj studiów: **stacjonarne**

Kierunek studiów: **informatyka**

Specjalność: **projektowanie baz danych i oprogramowanie użytkowe**

**PRACA DYPLOMOWA  
INŻYNIERSKA**

Temat pracy: **Projekt i implementacja systemu do obsługi czytelni elektronicznej**

Zakres pracy:

1. Współczesne czytelnie książek i czasopism elektronicznych oraz wersji papierowych.
2. Dostępne narzędzia informatyczne dla implementacji obsługi czytelni.
3. Wymagania sprzętowe i sieciowe dla czytelni.
4. Baza danych dla użytkowników i obsługi czytelni.
5. Baza danych komunikatów i alarmów.
6. Implementacja strony.
7. Testy produktu serwisowego.
8. Dokumentacja techniczna.

|  |  |  |
| --- | --- | --- |
| Promotor pracy |  | Dyrektor Instytutu |
| dr inż. Stanisław Witkowski |  |  |
| .................................................... podpis promotora pracy |  | ...................................................... podpis Dyrektora Instytutu |

Elbląg, 2024 r.

**O Ś W I A D C Z E N I E**

…………………………………….

Imię i Nazwisko Studenta

……………………………

nr albumu

Oświadczam, że moja praca pt.: …………………………………………………….….

………………………………………………………………………………………………

* + - * 1. została przygotowana przeze mnie samodzielnie,\*
        2. nie narusza praw autorskich w rozumieniu ustawy o prawie autorskim i prawach pokrewnych oraz dóbr osobistych chronionych prawem,
        3. nie zawiera danych i informacji, które uzyskałem w sposób niedozwolony,
        4. nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie, ani innej osobie.

Ponadto oświadczam, że treść pracy zamieszczonej przeze mnie w Archiwum Prac Dyplomowych jest identyczna z jej wersją drukowaną.

………………………, dnia ……………… …………………….……………….

podpis studenta

\* Uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminarium dyplomowego).

Spis treści

[1. Wstęp 7](#_Toc168332856)

[1.1. Wprowadzenie w problematykę organizacji pracy bibliotek 7](#_Toc168332857)

[1.2. Systemy biblioteczne i ich zalety 8](#_Toc168332858)

[1.3. Cel pracy 9](#_Toc168332859)

[2. Rozwiązania istniejące na rynku 11](#_Toc168332860)

[2.1. SowaSQL 11](#_Toc168332861)

[2.2. Alma 12](#_Toc168332862)

[2.3. MAK+ 13](#_Toc168332863)

[2.4. Podsumowanie funkcjonalności istniejących systemów bibliotecznych 14](#_Toc168332864)

[3. Projekt 16](#_Toc168332865)

[3.1. Wymagania funkcjonalne 16](#_Toc168332866)

[3.2. Wymagania niefunkcjonalne 18](#_Toc168332867)

[3.3. Przypadki użycia 18](#_Toc168332868)

[3.4. Wymagania sprzętowe i sieciowe 20](#_Toc168332869)

[3.5. Zakres działań programistycznych 21](#_Toc168332870)

[3.6. Baza danych 22](#_Toc168332871)

[4. Użyte technologie 25](#_Toc168332872)

[4.1. Środowisko programistyczne 25](#_Toc168332873)

[4.2. Technologie interfejsu użytkownika (frontend) 25](#_Toc168332874)

[4.2.1. i18next 26](#_Toc168332877)

[4.2.2. React 26](#_Toc168332878)

[4.2.3. Material UI 26](#_Toc168332879)

[4.2.4. i18next 26](#_Toc168332880)

[4.3. Technologie serwerowe (backend) 27](#_Toc168332881)

[4.3.1. Java 27](#_Toc168332882)

[4.3.2. Spring 27](#_Toc168332883)

[4.3.3. Hibernate 27](#_Toc168332884)

[4.4. Baza danych 28](#_Toc168332885)

[4.5. Kontrola wersji 28](#_Toc168332886)

[4.6. Inne narzędzia programistyczne 28](#_Toc168332887)

[4.6.1. Postman 28](#_Toc168332888)

[4.6.2. Chrome DevTools 29](#_Toc168332889)

[5. Implementacja 30](#_Toc168332890)

[5.1. Aplikacja serwerowa 30](#_Toc168332891)

[5.1.1. Struktura projektu Spring 30](#_Toc168332892)

[5.1.2. Architektura warstwowa aplikacji 31](#_Toc168332893)

[5.1.3. Repozytoria 33](#_Toc168332894)

[5.1.4. Serwisy 35](#_Toc168332895)

[5.1.5. Kontrolery 36](#_Toc168332896)

[5.1.6. Modele 37](#_Toc168332897)

[5.1.7. Bezpieczeństwo aplikacji 39](#_Toc168332898)

[5.2. Aplikacja przeglądarkowa 42](#_Toc168332899)

[5.2.1. Struktura projektu React 42](#_Toc168332900)

[5.2.2. Konteksty React 44](#_Toc168332901)

[5.2.3. Autoryzacja 44](#_Toc168332902)

[5.2.4. Tłumaczenie 45](#_Toc168332903)

[5.2.5. Interfejs aplikacji 46](#_Toc168332904)

[6. Testowanie 49](#_Toc168332905)

[6.1. Testowanie aplikacji serwerowej 49](#_Toc168332906)

[6.2. Testowanie aplikacji przeglądarkowej 50](#_Toc168332907)

[7. Podsumowanie 52](#_Toc168332908)

[8. Bibliografia 54](#_Toc168332909)

[9. Spis rysunków 57](#_Toc168332910)

[10. Spis tabel 58](#_Toc168332911)

# 1. Wstęp

## 1.1. Wprowadzenie w problematykę organizacji pracy bibliotek

Biblioteki działają w różnej formie od czasów starożytnych. Były rzadkością dostępną jedynie dla nielicznych osób. Korzystali z nich tylko ludzie zamożni, których stać było na wykształcenie. W dzisiejszych czasach jest jednak całkowicie inaczej. Umiejętność czytania jest dzisiaj pospolitą kompetencją, którą posiadają niemal wszystkie osoby na świecie. Nic więc dziwnego, że coraz więcej osób decyduje się na korzystanie z bibliotek.

Obraz zawierający tekst, zrzut ekranu, diagram, mapa

Opis wygenerowany automatycznieWedług raportu Głównego Urzędu Statystycznego w 2022 roku w Polsce działało 7638 bibliotek publicznych. Placówki te posiadały księgozbiór liczący 124,4 mln pozycji. Zdecydowana większość bibliotek publicznych i ich filii posiadała od 10 do 50 tys. materiałów do czytania. Przeciętnie na każdą z takich bibliotek przypadało 4945 mieszkańców. Rysunek 1.1 przedstawia mapę z liczbą bibliotek w każdym województwie oraz liczbę czytelników przypadającą na każdą z nich. Czytelników, którzy zdecydowali się na wypożyczenie, było 5,1 mln. Średnio na jedną osobę przypada 19 wypożyczonych woluminów [1].

Rysunek 1.1. Biblioteki publiczne w Polsce oraz liczba czytelników na 1 bibliotekę [1]  
*źródło: Główny Urząd Statystyczny, Biblioteki publiczne w 2022 r., Warszawa, 2023.*

Na podstawie powyższych danych można wywnioskować, jakie wyzwania stały przed bibliotekami chcącymi rozszerzać i udoskonalać swoje zbiory. Biblioteki musiały również stawić czoła problemom związanym z udostępnianiem zbiorów coraz większej ilości osób. Dawniej biblioteki zmagały się z wieloma problemami związanymi z zarządzaniem i monitorowaniem ruchu zasobów. Coraz większa i stale rosnąca liczba woluminów wymagała skatalogowania. Problemem stawał się żmudny proces katalogowania zasobów oraz zajmowana przez nie przestrzeń. Podstawowe czynności związane z pracą takiej placówki były wykonywane ręcznie i niezależnie od siebie. Chociaż pojawiały się próby ulepszenia niektórych aspektów tej pracy, nie zmieniły one integralności usług bibliotecznych. Rosnąca popularność księgozbiorów ujawniła wady dotychczasowej organizacji ich pracy. Było to spowodowane między innymi zwiększającą się liczbą czytelników oraz ich rosnącymi potrzebami. Brak automatyzacji podstawowych procesów, takich jak wypożyczanie i rezerwowanie książek, czy przypomnienia o terminach zwrotu, prowadził do długotrwałych operacji. Procesy te były nieefektywnie wykonywane, co z kolei powodowało powstawanie kolejek czytelników. Czytelnicy musieli czekać na obsłużenie przez pracowników biblioteki.

Czytelnik zainteresowany konkretnym tytułem nie wiedział, czy jest on dostępny w bibliotece, dopóki osobiście się do niej nie udał w celu jego wypożyczenia. Na miejscu często okazywało się, że biblioteka nie posiada go w swoich zasobach lub został on wypożyczony przez kogoś innego. Wśród wielu środowisk korzystających z bibliotek pojawiły się oczekiwania zmiany tego stanu rzeczy.

W celu poprawy jakości świadczonych usług, dyrekcje bibliotek zaczęły szukać sposobów na naprawę borykających ich problemów. Pomocne okazały się najnowsze rozwiązania technologiczne, szczególnie te z początku XXI wieku.

## 1.2. Systemy biblioteczne i ich zalety

W dzisiejszych czasach informatyka i jej rozwiązania często stają się odpowiedzią na wiele problemów w różnych aspektach życia. Wraz z jej rozwojem biblioteki zaczęły być unowocześnianie i komputeryzowane. Komputeryzacja bibliotek pozwoliła na znaczne polepszenie jakości obsługi czytelników. Usprawniła i ułatwiła zarządzanie rozrastającymi się katalogami oraz automatyzację rutynowych procesów bibliotecznych. Dała bibliotekom możliwość wprowadzenia nowych usług opartych o ich katalog, takich jak: zdalne przeglądanie (wyszukiwanie) w nim, zdalna rezerwacja książek, przedłużanie wypożyczenia oraz szybkie wypożyczenia międzybiblioteczne. Dodatkowo pozwoliło to na rozpoczęcie świadczenia nowych usług, takich jak udostępnianie baz danych i archiwów czy rozpowszechnianie pełnych tekstów za pomocą Internetu [2].

Trudno jest dziś wyobrazić sobie bibliotekę, która nie udostępnia swojego katalogu zasobów w Internecie i nie pozwala na ich zdalne wypożyczenie lub zapisanie się do listy oczekujących. Takie rozwiązanie jest bardzo wygodne dla czytelnika, pozwala bowiem na sprawdzenie dostępności poszukiwanej literatury z domu, pracy czy podczas podróży. Dzięki temu nie musi on osobiście udawać się do biblioteki. Może również skorzystać z innej biblioteki, która posiada tę konkretną pozycję i ma ją dostępną do wypożyczenia.

W 2022 roku w Polsce 6633 z 7638 bibliotek publicznych oferowało usługi elektroniczne. Katalog on-line oferowało 92,2% z tych bibliotek. Zdalne złożenie zamówienia na zasoby biblioteczne oferowało 64,4% placówek. Możliwość zdalnej rezerwacji pozycji bibliotecznych była dostępna w 67,5% bibliotek. Zdalne przedłużenie wypożyczenia było możliwe w 58,9% z nich [3]. Są to liczby, które stale rosną.

Wprowadzenie systemów komputerowych w działalność bibliotek nie byłoby możliwe bez odpowiedniego oprogramowania. Obecnie funkcjonują one w sposób, który znamy dzięki zintegrowanym systemom bibliotecznym. Umożliwiły one wykonanie podstawowych operacji przeprowadzanych w bibliotekach w sposób szybszy i wygodniejszy. Było to korzystne zarówno dla pracowników placówki, jak i dla jej klientów, czyli czytelników. Dodatkowo, operacje te stały się o wiele bardziej precyzyjne. Co najważniejsze, czynności te zostały ze sobą zintegrowane i zcentralizowane. Wcześniej procesy te były wykonywane niezależnie od siebie, co mocno utrudniało wydajne zarządzanie biblioteką. Obecne zintegrowane systemy biblioteczne najczęściej są podzielone na różne, ściśle powiązane ze sobą moduły, które współpracują w celu umożliwienia wykonania transakcji. Wśród modułów wyróżnia się:

* nabywanie, gromadzenie - wykorzystywany do zamawiania, nabywania nowych materiałów bibliotecznych;
* katalogowanie - do generowania i zarządzania zasobami biblioteki;
* OPAC (z ang. *Online Public Access Catalog*) - używany do przeszukiwania katalogu biblioteki przez użytkowników;
* obieg - stosowany do zarządzania i monitorowania ruchu woluminów;
* serie - śledzenie zasobów czasopism, magazynów i gazet [4].

## 1.3. Cel pracy

Celem niniejszej pracy jest stworzenie projektu systemu bibliotecznego oraz jego implementacja w zakresie obsługi czytelni elektronicznej. Aplikacja będzie zapewniać funkcje przechowywania informacji o materiałach bibliotecznych oraz wypożyczania ich przez zainteresowanych czytelników. Materiały cyfrowe będą możliwe do pobrania bezpośrednio z systemu informatycznego. Ponadto, dostępna będzie funkcja dodawania nowych rekordów do bazy danych, co ułatwi wprowadzanie nowych pozycji literaturowych. Użytkownik będzie mógł przeglądać swoje wypożyczenia, a w razie potrzeby przedłużyć je. Osoba zarządzająca biblioteką będzie mogła akceptować prośby wypożyczeń lub je odrzucać. Będzie również sprawdzać książki dodane do bazy danych przez użytkowników przed ich udostępnieniem do wypożyczenia. Aplikacja będzie posiadać prosty system powiadomień po zalogowaniu się do aplikacji. W aplikacji wprowadzony będzie podział na użytkowników (czytelników) oraz administratora zarządzającego bazą danych. Niniejszy system będzie wymagał komputera podłączonego do Internetu.

# 2. Rozwiązania istniejące na rynku

Obecnie na rynku dostępnych jest wiele rozwiązań zintegrowanych systemów bibliotecznych. Wszystkie oferują zbliżone możliwości z niewielkimi różnicami funkcjonalnymi. Wynika to między innymi z faktu, że były projektowane na potrzeby różnych bibliotek. Każda z tych placówek miała inne wymagania i organizację pracy. Miało to główny wpływ na to jak te systemy wyglądają oraz jak działają [5].

W niniejszym rozdziale autor pracy omówił trzy systemy biblioteczne używane w różnych bibliotekach w Polsce. Z powodu braku dostępu do modułów administratorskich, zostały przeanalizowane funkcje dostępne dla czytelnika. Oprócz funkcjonalności skupiających się na automatyzacji procesów bibliotecznych (przeglądanie katalogu, wypożyczanie, zarządzanie trwającymi wypożyczeniami, rezerwowanie), autor przeanalizował również funkcje związane z dostępnością. Obejmowały one takie aspekty, jak kontrast czy rozmiar czcionki, oraz funkcje związane z wygodą użytkownika, np. zmianę języka wyświetlania. Na podstawie analizy działania wybranych systemów bibliotecznych wskazano w pracy ich najważniejsze funkcje. Następnie uwzględniono w projekcie tworzonej aplikacji do zrealizowania celu pracy.

## 2.1. SowaSQL

SowaSQL to zintegrowany system biblioteczny stworzony oraz dystrybuowany przez firmę Sokrates-software. Jest on używany w wielu rodzajach bibliotek. W 2012 roku był wykorzystywany w 1326 bibliotekach publicznych w Polsce [6].

Analiza funkcji tego systemu została przeprowadzona na podstawie strony biblioteki Akademii Nauk Stosowanych w Elblągu, która udostępnia swój katalog on-line. Placówka ta używa różnych wersji tego systemu od 2007 roku [7].

Można w nim wyróżnić funkcje dla użytkownika zalogowanego jak i niezalogowanego w systemie. Użytkownik niezalogowany może przeglądać katalog biblioteki wykorzystując do tego wyszukiwarkę prostą lub zaawansowaną. Rekordy wyszukiwać można podając tytuł, autora, temat, wydawcę, nazwę imprezy, numer ISBN, instytucję czy tytuł serii. Kryteria wyszukiwania można ze sobą łączyć. Za pomocą wyszukiwarki zaawansowanej można też wykluczać z wyników wyszukiwania rekordy o kryteriach niepożądanych. Wyszukane rekordy można dodatkowo filtrować. SowaSQL oferuje rozbudowane możliwości filtrowania wyników wyszukiwania. Dostępne jest również sortowanie m.in. wg trafności czy roku wydania. Oferowana jest historia zadanych zapytań wyszukiwania oraz liczba wyników, które dane zapytanie zwróciło. Zapytania te są zapisywana w pamięci przeglądarki, można je jednak również zapisać na koncie czytelnika. System dostarcza informacji, ile egzemplarzy danego materiału bibliotecznego jest dostępnych do wypożyczenia oraz w jakich placówkach biblioteki można wypożyczyć dany materiał. Interesujące czytelnika rekordy można zapisywać w koszyku informatycznym. Koszyk ten przechowuje rekordy w pamięci przeglądarki internetowej, umożliwiając szybkie ich ponowne znalezienie. Rekordy te są dostępne dla każdego użytkownika komputera (chyba, że zostaną one z niego usunięte). Po zalogowaniu się na konto użytkownika istnieje również opcja przeniesienia tych danych z pamięci przeglądarki do bazy. Wówczas rekordy te nie będą dostępne dla innych użytkowników komputera po wylogowaniu się z systemu. Na tym kończą się podstawowe funkcje użytkownika niezalogowanego.

Po zalogowaniu czytelnik dostaje możliwość wypożyczenia książki (która musi zostać oznaczona jako gotowa do odbioru przez osobę pracującą w bibliotece). W przypadku, gdy dana pozycja nie jest aktualnie dostępna do wypożyczenia, może zapisać się do listy oczekujących. Z poziomu konta użytkownika można przeglądać profil użytkownika oraz historię wypożyczeń czytelnika. Można także zarządzać wypożyczeniami, prolongować te trwające lub anulować te oczekujące. System wysyła także przypomnienia w formie e-mail o zbliżającym się terminie zwrotu książki. System oferuje również zmianę języka wyświetlania. Do wyboru są język polski lub język angielski. Aspekt dostępności także został uwzględniony. System oferuje 4 warianty kontrastu oraz 3 warianty rozmiaru czcionki.

## 2.2. Alma

System Alma jest oprogramowaniem stworzonym przez izraelską firmę Ex Libris. Zadebiutował on w 2011 roku. Działa w oparciu o model SaaS (ang. Software as a Service), dzięki czemu nie wymaga lokalnej instalacji. W 2018 Biblioteka Narodowa zakupiła ten system w ramach przetargu i od tego czasu wdraża ten system w bibliotekach w całej Polsce [8]. W latach 2022-2023 wdrożono ten system w aż 131 bibliotekach [9].

Na podstawie katalogu on-line Biblioteki Elbląskiej im. C. Norwida przeanalizowano funkcje dostępne dla czytelnika. Placówka ta używa systemu Alma od 2022 roku [10].

W przypadku tego oprogramowania, jego funkcje także różnią się w zależności od tego czy jest użytkowane przez użytkownika niezalogowanego czy zalogowanego. Czytelnik niezalogowany może przeglądać katalog placówki, uwzględniając wszystkie filie biblioteki, oraz połączony katalog wszystkich bibliotek włączonych do systemu. Wyszukiwarka umożliwia standardowe wyszukiwanie po haśle oraz wyszukiwanie zaawansowane. Możliwe jest również wyszukiwanie głosowe. Wynik wyszukiwania można zawęzić za pomocą rozbudowanego filtrowania oraz sortowania. Użytkownik ma możliwość zdefiniowania ile rekordów ma być pokazywane na jednej stronie. Umożliwia to mechanizm paginacji. Rekordy można zapisywać do ulubionych, w celu szybszego dostępu w późniejszym czasie. Dostępne są informacje o wszystkich wersjach danego materiału (różne wydania, formaty) oraz placówki, w których jest on dostępny do wypożyczenia, łącznie z ilością egzemplarzy. Użytkownik może również sprawdzić dostępność woluminu w innych bibliotekach z całej Polski korzystających z tego systemu.

Zalogowany użytkownik ma dostęp do dodatkowych funkcji w porównaniu do tych, które są dostępne dla czytelnika niezalogowanego. Możliwa jest opcja zamówienia książki, tj. dostarczenie książki do biblioteki wskazanej przez użytkownika. Dostępny jest także profil czytelnika, w ramach którego można przeglądać poprzednie i aktywne wypożyczenia oraz prolongować je. System pokazuje informacje o dacie wypożyczenia czy terminie zwrotu. Użytkownik może również przeglądać swoje zamówienia oraz obowiązujące go kary i opłaty. Dodatkowo można zapisać konkretne wyszukiwania oraz przeglądać historię wykonanych wcześniej zapytań. System nie oferuje jednak zmiany kontrastu ani wyboru rozmiaru czcionki. Możliwa jest zmiana języka wyświetlania. Użytkownik może wybrać między j. polskim lub j. angielskim.

## 2.3. MAK+

MAK+ jest system bibliotecznym stworzonym oraz rozwijanym przez Instytut Książki. Według jego producenta aktualnie jest używany w 2713 bibliotekach w Polsce [11]. Katalogowanie zbiorów biblioteki za pomocą tego systemu ma miejsce jednocześnie w dwóch miejscach: na stronie internetowej biblioteki oraz na portalu szukamksiazki.pl.

Funkcje tego systemu zostały przeanalizowane na podstawie katalogu on-line Miejskiej Biblioteki Publicznej w Braniewie na portalu szukamksiazki.pl.

Tak jak w dwóch poprzednich rozwiązaniach, system MAK+ również oferuje podział na użytkownika niezalogowanego i zalogowanego. Niezalogowany użytkownik może przeglądać katalog za pomocą wyszukiwarki, która posiada tryb prosty oraz zaawansowany. Nie ma możliwości przeglądania całego katalogu bez podania przynajmniej jednego kryterium wyszukiwania. W trybie zaawansowanym użytkownik może wybrać format szukanego materiału (książki, audiobooki, filmy, artykuły). Wyniki zwrócone przez wyszukiwarkę można zawęzić używając filtrów oferowanych przez system. Dostępna jest także opcja sortowania wyników, czy zdefiniowanie ilości wyświetlanych rekordów na jednej. System informuje użytkownika o formacie materiału bibliotecznego, ilości egzemplarzy dostępnych do wypożyczenia oraz w której placówce można dany materiał wypożyczyć.

Po zalogowaniu system oferuje czytelnikowi więcej możliwości. W przypadku egzemplarza książki, który jest już wypożyczony, widoczna jest liczba osób w kolejce do jej wypożyczenia. Użytkownik może zarezerwować egzemplarz książki lub dopisać się do kolejki oczekujących. Może też dodać rekord do swojej listy zapisanej na koncie użytkownika. Z poziomu widoku tego konta czytelnik może przeglądać aktualnie wypożyczone przez niego materiały biblioteczne, zarezerwowane woluminy czy też przeanalizować historię wypożyczeń. Dodatkowo może przeglądać wszystkie biblioteki do których jest zapisany w ramach systemu MAK+ i wybrać jedną z nich jako bibliotekę główną. Ponadto system oferuje zmianę kontrastu interfejsu oraz trzy rozmiary czcionki. Pozwala to na dostosowanie go pod potrzeby użytkowników potrzebujących tego rodzaju udogodnień. Tak jak dwa wcześniej omówione oprogramowania biblioteczne, system MAK+ również oferuje interfejs w języku polskim lub angielskim.

## 2.4. Podsumowanie funkcjonalności istniejących systemów bibliotecznych

W Tabeli 2.1 znajduje się zestawienie funkcjonalności, które omówione systemy biblioteczne oferują czytelnikom.

Tabela 0.1. Zestawienie funkcjonalności oferowanych czytelnikowi przez wybrane systemy biblioteczne  
*źródło: opracowanie własne*

|  |  |  |  |
| --- | --- | --- | --- |
| **System biblioteczny**  **Funkcjonalność** | **SowaSQL** | **Alma** | **MAK+** |
| Przeglądanie katalogu bez podawania kryteriów wyszukiwania | Tak | Tak | Nie |
| Wyszukiwanie proste w katalogu | Tak | Tak | Tak |
| Wyszukiwanie zaawansowane w katalogu | Tak | Tak | Tak |
| Wyszukiwanie głosowe w katalogu | Nie | Tak | Nie |
| Filtrowanie wyników wyszukiwania | Tak | Tak | Tak |
| Sortowanie wyników wyszukiwania | Tak | Tak | Tak |
| Definiowanie liczby wyświetlanych rekordów na stronie | Nie | Tak | Tak |
| Koszyk/lista ulubionych | Tak | Tak | Tak |
| Zarządzanie kontem czytelniczym (historia wypożyczeń, prolongowanie wypożyczeń, przegląd trwających wypożyczeń) | Tak | Tak | Tak |
| Wypożyczanie/rezerwowanie materiałów bibliotecznych | Tak | Tak | Tak |
| Zapisanie wyszukiwań | Tak | Tak | Nie |
| Historia wyszukiwania | Tak | Tak | Nie |
| System powiadomień (przypominanie o zwrocie, powiadomienie o przetrzymanych materiałach bibliotecznych) | Tak | Tak | Tak |
| Zmiana kontrastu interfejsu | Tak | Nie | Tak |
| Zmiana języka wyświetlania | Tak | Tak | Tak |
| Przeglądanie katalogu innych bibliotek włączonych do systemu | Nie | Tak | Tak |

Po dokonaniu analizy wybranych systemów bibliotecznych autor pracy stwierdza, że są one w swojej funkcjonalności bardzo zbliżone. Najczęściej różnią się one sposobem ich implementacji lub stopniem ich rozbudowania. Jedna z najbardziej znaczących różnic, jaką można zaobserwować to funkcja filtrowania wyników wyszukiwania, którą posiadają wszystkie analizowane systemy biblioteczne. Jednak w systemie Alma jest ona najbardziej rozbudowana i daje największe możliwości funkcjonalne.

# 3. Projekt

Projektowanie systemu to jeden z najważniejszych etapów tworzenia oprogramowania. Ułatwia ono późniejszą implementację, poprzez dokładne określenie wszystkich funkcji oraz specyfikacji systemu. Na fazę tę składa się stworzenie różnego rodzaju modeli, schematów czy wykresów, które wskazywać będą sposób działania aplikacji w różnych scenariuszach. Pozwala ona na zaplanowanie dalszej części pracy, wykluczając w większości przypadków możliwość wystąpienia niespodziewanych czynników.

Podczas fazy projektowania aplikacji określono jej wymagania funkcjonalne i niefunkcjonalne, a także sporządzono diagram przypadków użycia. Ponadto, zdefiniowane zostały przybliżone minimalne i zalecane wymagania sprzętowe i sieciowe do poprawnej obsługi systemu. Dodatkowo wykonany został projekt bazy danych, która będzie obsługiwała tworzoną aplikację, a także sporządzono zakres działań programistycznych potrzebnych do wykonania podczas całego etapu tworzenia programu.

## 3.1. Wymagania funkcjonalne

Określenie wymagań funkcjonalnych to bardzo ważny etap podczas powstawania różnego rodzaju oprogramowania. Definiują one w jaki sposób będzie działał tworzony system i jak będzie się on zachowywał. Przedstawia cechy i funkcje, które powinna posiadać aplikacja, tak aby potrzeby i oczekiwania użytkownika zostały spełnione w jak największym stopniu. W uproszczeniu można przyjąć, że są to cechy produktu, które są wykrywane przez jego użytkownika. Przykładowo, może to być przycisk na stronie pełniący konkretną, określoną funkcję. System taki nie będzie mógł działać, jeśli nie będą spełnione wszystkie określone wcześniej wymagania funkcjonalne [12].

Na podstawie analizy istniejących rozwiązań bibliotecznych wybrano najważniejsze cechy, które powinna posiadać tworzona aplikacja. Cechy te zostały podzielone ze względu na stan zalogowania użytkownika oraz jego rolę w systemie.

Dla użytkownika niezalogowanego system umożliwia:

* zarejestrowanie się i stworzenie konta podając imię, nazwisko, unikalny adres e-mail oraz hasło spełniające wymogi bezpieczeństwa;
* zalogowanie się do systemu za pomocą adresu e-mail oraz hasła podanego przy rejestracji.

Dla użytkownika zalogowanego system umożliwia:

* przeglądanie materiałów bibliotecznych (książki, czasopisma, prace dyplomowe) w formie papierowej udostępnione do wypożyczenia;
* przeglądanie materiałów bibliotecznych w formie cyfrowej udostępnione do pobrania;
* przeglądanie materiałów bibliotecznych w formie papierowej aktualnie niedostępnych (wypożyczonych przez innych użytkowników);
* filtrowanie katalogu stosując kryteria: autor, tytuł, wydawca lub kategoria. Kryteria filtrowania powinny móc być zastosowane osobno lub kilka jednocześnie;
* zarezerwowanie materiałów bibliotecznych dostępnych do wypożyczenia;
* prolongowanie aktualnie trwającego wypożyczenia;
* pobranie materiału bibliotecznego w formie cyfrowej dostępnego do pobrania;
* dodanie za pomocą formularza nowych rekordów do katalogu (bazy danych) materiałów bibliotecznych wymagających zaakceptowania przez administratora;
* dodanie autora do bazy danych za pomocą formularza;
* dodanie wydawnictwa do bazy danych za pomocą formularza;
* otrzymywanie powiadomień o wydarzeniach związanych z wypożyczeniami lub materiałami bibliotecznymi (zaakceptowanie lub odrzucenie dodanego materiału bibliotecznego, zaakceptowanie lub odrzucenie wypożyczenia, przypomnienie o zbliżającym się terminie zwrotu materiału, przypomnienie o upływie terminie zwrotu materiału);
* oznaczanie powiadomień jako odczytane;
* przeglądanie trwających, oczekujących, zakończonych oraz odrzuconych wypożyczeń;
* wylogowanie się z systemu.

Dla użytkownika administratora system umożliwia:

* przeglądanie materiałów bibliotecznych oczekujących na zatwierdzenie;
* zaakceptowanie lub odrzucenie oczekującego materiału bibliotecznego;
* przeglądanie trwających lub zakończonych wypożyczeń materiałów bibliotecznych wszystkich użytkowników;
* filtrowanie wypożyczeń użytkowników według ich imienia i nazwiska;
* wysyłanie powiadomień przypominających o terminie zwrotu materiału bibliotecznego;
* oznaczanie wypożyczeń jako zakończone;
* przeglądanie wypożyczeń oczekujących na zaakceptowanie;
* zatwierdzanie lub odrzucanie oczekujących wypożyczeń;
* edytowanie informacji o materiałach bibliotecznych w bazie danych;
* usuwanie materiałów bibliotecznych z bazy danych;
* dodanie za pomocą formularza nowych rekordów do katalogu (bazy danych) materiałów bibliotecznych bez potrzeby zatwierdzania.

Projekt zakłada, że niektóre funkcje użytkownika zalogowanego i administratora są zbieżne. Dotyczą one przeglądania katalogu dostępnych materiałów bibliotecznych czy dodawania autorów i wydawnictw do bazy danych podczas uzupełniania formularza do dodawania nowego rekordu bibliotecznego.

## 3.2. Wymagania niefunkcjonalne

Obok wymagań funkcjonalnych należy zdefiniować wymagania niefunkcjonalne. Podobnie jak wymagania funkcjonalne pomagają stworzyć oprogramowanie, które sprosta oczekiwaniom użytkujących go osób i będzie działać zgodnie z jego przeznaczeniem. Nie mają wpływu na funkcjonalność projektowanej aplikacji, ale definiują jej atrybuty jakościowe. Dotyczą takich zagadnień jak wydajność, bezpieczeństwo czy skalowalność. Wymagania niefunkcjonalne są niezbędne podczas projektowania systemu. Jednakże, w przeciwieństwie do wymagań funkcjonalnych, system jest w stanie działać nawet w przypadku niespełnienia wszystkich postawionych mu wymagań tego rodzaju [13].

Dla tworzonej aplikacji zdefiniowano następujące wymagania niefunkcjonalne:

* system powinien działać asynchronicznie;
* system powinien udostępniać dostęp do zasobów dopiero po uwierzytelnieniu;
* system powinien mieć opcję zmiany języka wyświetlania;
* system powinien posiadać tryb ciemny oraz tryb jasny interfejsu;
* system powinien posiadać czytelny i prosty interfejs;
* system powinien być prosty w obsłudze;
* dane użytkowników powinny być bezpieczne;
* system powinien być responsywny i dostosowywać wygląd interfejsu do używanego urządzenia.

## 3.3. Przypadki użycia

Przypadki użycia są kolejnym sposobem na dokładne określenie wymagań, jakie ma spełniać projektowane oprogramowanie. Odwzorowują one jego funkcje w sposób, w jaki są postrzegane przez jego użytkowników. Opisują sekwencje zachodzące między systemem a jego użytkownikami [14]. Są one bardzo ważne w kwestii projektowania funkcji.

Przypadki użycia najczęściej przedstawia się za pomocą graficznego diagramu przypadków użycia. Pozwala to na czytelne przedstawienie specyfikacji projektowanego systemu. Diagram taki składa się z aktorów, przypadków użycia oraz powiązań między nimi. Aktorami określa się pewną rolę, którą użytkownik pełni w odniesieniu do systemu. Przypadki użycia zapisywane są w elipsie, za pomocą trybu rozkazującego. Są zbiorem różnych scenariuszy, które łączy wspólny cel aktora [15]. Każdy z przypadków użycia musi być powiązany przynajmniej z jednym aktorem.

Rysunek 3.1 przedstawia diagram przypadków użycia dla projektowanej aplikacji. Na diagramie znajdują się dwaj aktorzy: czytelnik oraz administrator. Pominięty został użytkownik niezalogowany ze względu na czytelność diagramu oraz dlatego, że może on jedynie zalogować i zarejestrować się do systemu.

Obraz zawierający tekst, mapa, diagram

Opis wygenerowany automatycznie

Rysunek 3.1. Diagram przypadków użycia projektowanej aplikacji  
*źródło: opracowanie własne*

Każdy z aktorów jest powiązany z danymi przypadkami użycia za pomocą asocjacji skierowanej. Wskazuje ona na to, że element inicjujący (aktor) zna element inicjowany (przypadek użycia). Dodatkowo między przypadkami użycia zachodzą związki rozszerzania oraz zawierania. Związek rozszerzania oznaczony jest podpisem *extends* i wskazuje on na opcjonalne rozszerzenie bazowego przypadku użycia. Przykładowo, przypadek użycia *Przeglądaj listę materiałów bibl.* może zostać rozszerzony o filtrowanie tej listy za pomocą odpowiedniego elementu interfejsu użytkownika. Związki zawierania, oznaczone podpisem *includes*, wskazują, że dany przypadek użycia zawsze rozszerza bazowy przypadek użycia. Na wykresie przypadek użycia *Zarezerwuj materiał biblioteczny* zawsze musi skończyć się potwierdzeniem lub odrzuceniem wypożyczenia przez administratora.

## 3.4. Wymagania sprzętowe i sieciowe

Podczas projektowania aplikacji należy pamiętać o tym, że istnieją różne konfiguracje sprzętowe. System powinien działać na jak największej liczbie możliwych konfiguracji. Mimo to niemożliwe jest, by działał na każdej z nich. Rozwój oprogramowania niekiedy wymaga również większej mocy obliczeniowej komputera, na którym będzie ono uruchamiane.

W celu uniknięcia problemów z działaniem oprogramowania, które nie są związane z tym jak zostało ono zaprojektowane i zaimplementowane, sporządza się wymagania sprzętowe. Niekiedy dochodzą do tego także wymagania jakie musi spełniać sieć, do której podłączony będzie komputer z zainstalowanym systemem. Najczęściej sporządzane są minimalne i zalecane wymagania sprzętowe. Dzięki temu producent oprogramowania może określić sprzęt, na którym jego produkt będzie działał najlepiej. Najbardziej pożądane jest, aby system wymagał jak najmniej zasobów do poprawnego działania. Jednak mimo optymalizacji kodu, nie zawsze efekt ten udaje się osiągnąć.

Na podstawie analizy udostępnionych wymagań sprzętowych istniejących systemów bibliotecznych, technologii zaplanowanych do zaimplementowania aplikacji i doświadczeń autora sporządzono minimalne oraz zalecane wymagania sprzętowe i sieciowe. Wymagania te znajdują się w Tabeli 3.1.

Tabela 3.1. Minimalne i zalecane wymagania komputera dla instalacji projektowanej aplikacji  
*źródło: opracowanie własne*

|  |  |  |
| --- | --- | --- |
| **Wymaganie** | **Minimalna specyfikacja** | **Zalecana specyfikacja** |
| Procesor | Intel Core i3 z 2015 roku, AMD Ryzen 3 z 2017 roku | Intel Core i5 z 2015 roku lub nowszy, AMD Ryzen 5 z 2017 roku lub nowszy |
| Pamięć operacyjna (RAM) | 8 GB | 16 GB lub więcej |
| Pamięć masowa | 500 GB | 1 TB lub więcej |
| Karta sieciowa | 10/100 Mbit/s | 10/100/1000 Mbit/s |
| System operacyjny | Windows 10 | Windows 10 |
| Sieć (prędkość pobierania / prędkość wysyłania) | 20 Mb/s / 5 Mb/s | 50 Mb/s /15 Mb/s |

Przedstawione parametry techniczne komputera i sieci powinny zapewnić, że obsłużenie bazy danych z około 5 tys. rekordów bibliotecznych powinno działać poprawnie. Nie można dokładnie określić wymagań, więc są to jedynie przybliżone wymagania sprzętowe i sieciowe. Zdaniem autora komputer, na którym instalowany będzie projektowany system powinien je spełniać, by zapewnić poprawne działanie aplikacji. Należy pamiętać, że producenci oprogramowania również podają przybliżone wymagania sprzętowe i sieciowe, które powinny spełniać komputery, na których instalowane jest tworzone przez nich oprogramowanie.

## 3.5. Zakres działań programistycznych

Mając wyniki analizy istniejących już systemów bibliotecznych oraz określone wymagania funkcjonalne i niefunkcjonalne, można określić zakres działań programistycznych. Definiuje on zestaw działań oraz czynności, które należy podjąć w celu zrealizowania wszystkich celów projektu. Poprawnie zdefiniowany zakres działań pozwoli na stworzenie systemu realizującego swoje zadania w sposób, który zadowoli użytkowników.

Istotną częścią takiego systemu jest baza danych. System bazy danych będzie związany z przechowywaniem wszystkich informacji o użytkownikach, materiałach bibliotecznych, autorach, wydawnictwach czy zdjęciach. Dobrze zaprojektowana baza danych pozwoli na szybki dostęp do danych oraz ich przedstawienie użytkownikom za pomocą interfejsu aplikacji klienckiej. Poza projektem samej bazy danych ważny jest również wybór odpowiedniego systemu do zarządzania nią. Ze względu na koszty finansowe za najodpowiedniejsze uznano oprogramowanie darmowe, które nie wymaga żadnej licencji. Jednocześnie oprogramowanie to pozwoli zapewnić skalowalność i łatwość zarządzania bazą danych aplikacji.

Wybrano styl interfejsu aplikacji, który spełnia postawione mu wymagania, a następnie upewniono się, że jest on spójny w przekroju całej aplikacji. W tym przypadku konieczne było to, aby był on czytelny, prosty i intuicyjny. Powinno pozwolić to na odpowiedni odbiór oprogramowania oraz ułatwienie użytkownikom używanie systemu do celów, do których został zaprojektowany.

Ponadto, zadbano, aby aplikacja posiadała osobne funkcje dostępne tylko dla osoby zarządzającej katalogiem bibliotecznym. Dotyczy to kontroli dodawanych treści poprzez ich zatwierdzanie przed upublicznieniem oraz potwierdzanie wykonania czynności bibliotecznych, takich jak wypożyczenie czy zwrot materiału bibliotecznego.

Kluczowym punktem był wybór odpowiedniego oprogramowania, które zostanie użyte do stworzenia aplikacji serwerowej i klienckiej. Wymagane było dokładne przeanalizowanie dostępnych na rynku rozwiązań technologicznych pod kątem wydajności, łatwości używania, bezpieczeństwa, wsparcia i przeznaczenia. Następnie należało wybrać taki zestaw technologii, który zapewni implementację systemu za pomocą najodpowiedniejszych do tego narzędzi. Jest to bardzo ważny punkt projektowania, ponieważ trudno jest zmienić używane technologie w trakcie trwającej już implementacji. Jest to także bardzo czasochłonne.

Bardzo ważnym aspektem było stworzenie WebAPI (ang. *Web Application Programming Interfaces*) serwera, czyli interfejsu sieciowego aplikacji serwerowej. Odpowiada on za komunikację serwera z innymi częściami systemu lub nawet innymi aplikacjami.

Należało pamiętać o sporządzeniu dokumentacji technicznej. Dokumentacja może zawierać opis bazy danych, kodu źródłowego aplikacji (opisy funkcji, klas, modułów), specyfikację techniczną, opis API oraz instrukcje konfiguracji i użytkowania. Dzięki temu łatwiej zrozumieć, jak zbudowana jest aplikacja, jak działa, a przede wszystkim jak powinna być używana i skonfigurowana. Dokumentacja techniczna powinna stanowić istotny element tworzenia oprogramowania, pozwala bowiem na łatwiejsze zarządzanie nim i rozwijanie go w przyszłości przez kolejnych programistów.

Po wykonaniu implementacji należało wykonać test stworzonego oprogramowania. Umożliwiło to znalezienie jak największej ilości możliwych błędów występujących podczas korzystania z aplikacji. Testy zostały wykonane w sposób manualny.

## 3.6. Baza danych

Każda bardziej złożona aplikacja powinna być wyposażona w bazę danych. Przechowują one informacje niezbędne do funkcjonowania aplikacji internetowych, mobilnych oraz innych rodzajów aplikacji. Bazy danych są niezbędne dla aplikacji dostarczających dane w czasie rzeczywistym oraz obsługujących tysiące użytkowników. Służą także do przechowywania dużych ilości informacji.

Podczas projektowania bazy danych do tworzonej aplikacji zadbano, by nie była ona skomplikowana, ale jednocześnie przechowywała wszystkie potrzebne informacje. Na Rysunku 3.2 przedstawiony jest diagram bazy danych użytej do obsłużenia aplikacji.

Obraz zawierający tekst, zrzut ekranu, diagram, design

Opis wygenerowany automatycznie

Rysunek 3.2. Schemat bazy danych  
*źródło: opracowanie własne*

Baza danych składa się z następujących tabel:

* book – zawiera podstawowe informacje o materiałach bibliotecznych, takie jak tytuł, ISBN, liczba stron, kategoria czy wydawca, a także bity określające czy dany materiał jest w formacie cyfrowym, publiczny lub usunięty;
* category – zawiera nazwy kategorii dostępnych w katalogu;
* publisher – zawiera nazwy wydawnictw;
* author – zawiera imiona i nazwiska autorów;
* book\_author – zawiera dwa klucze obce: id materiału bibliotecznego oraz id autora. Dzięki tej tabeli możliwe jest śledzenie powiązań między autorami i materiałami bibliotecznymi;
* notification – służy do przechowywania powiadomień. Używana jest do informowania użytkowników o różnych zdarzeniach, takich jak wypożyczenie książki czy przypomnienie o upływającym terminie zwrotu materiału bibliotecznego;
* user – tabela przechowująca informacje o użytkownikach oraz ich rolach;
* file\_data – tabela przechowująca informacje o zapisywanych plikach cyfrowych książki;
* image\_data – tabela przechowująca informacje o zdjęciach używanych jako okładki materiałów cyfrowych;
* deleted\_book – tabela służąca jako kopia materiałów bibliotecznych, które zostały usunięte. Zamiast całkowicie usuwać rekord z tabeli book, jego kopia jest wykonywana do tej tabeli, by w przyszłości móc mieć do tego rekordu dostęp.

# 4. Użyte technologie

Bardzo ważnym etapem podczas projektowania i tworzenia oprogramowania jest wybór użytych technologii. Rozwiązania wybrane w tej pracy zależą od wielu czynników i zagadnień, które miały wpływ na stworzenie niniejszego systemu. Jednocześnie najważniejszym aspektem był rodzaj tworzonego oprogramowania. Narzędzia techniczne należy używać zgodnie z ich przeznaczeniem. W tym samym czasie należy pamiętać o ich ograniczeniach [16].

Autor wybierając technologie użyte do stworzenia systemu kierował się ich znajomością, popularnością oraz pozycją na rynku. Pozwoliło to na wybranie stabilnych technologii, stale rozwijanych, jednocześnie o wyrobionej renomie i wysokiej jakości. Dodatkowo skróciło to czas pisania kodu, ze względu na brak potrzeby zaznajamiania się z nowymi rozwiązaniami. Przeanalizowano również użyteczność i przeznaczenie tych rozwiązań, by upewnić się, że będą one odpowiednie do stworzenia systemu bibliotecznego.

W niniejszym rozdziale omówione zostaną krótko najważniejsze technologie użyte podczas implementacji systemu bibliotecznego, wraz z uzasadnieniem ich wyboru.

## 4.1. Środowisko programistyczne

Wybranym i używanym przez autora pracy zintegrowanym środowiskiem programistycznym jest IntelliJ IDEA, który został stworzony przez firmę JetBrains. Wybór oprogramowania został dokonany ze względu na bogatą funkcjonalność tego programu oraz doświadczenie autora w pracy z tym narzędziem programistycznym. IntelliJ IDEA oferuje inteligentny edytor kodu wykrywający błędy w czasie rzeczywistym, sugestie oparte na kontekście czy możliwość bezpiecznej i automatycznej refaktoryzacji kodu. Program zawiera wiele wbudowanych narzędzi, takich jak debugger, terminal, narzędzie do baz danych czy interfejs do obsługi systemu kontroli wersji. Dodatkowo posiada pełne wsparcie dla wielu języków programowania, w tym użyte w projekcie Java i TypeScript oraz inne technologie pochodne [17]. Wraz z nowoczesnym i przejrzystym interfejsem, IntelliJ IDEA znacząco ułatwił pracę nad pisaniem kodu tworzonego systemu.

## 4.2. Technologie interfejsu użytkownika (frontend)

Frontend to część aplikacji, z którą użytkownik wchodzi w interakcję. Jest ona renderowana w przeglądarce internetowej. Do jej programowania używa się głównie HTML, CSS oraz JavaScript. Najczęściej są one wspomagane dodatkowymi, pochodnymi technologiami, które oferują znacznie więcej możliwości.

W niniejszym podrozdziale zostaną wymienione technologie frontend użyte podczas implementacji tworzonej aplikacji.



### 4.2.1. i18next

TypeScript jest nadzbiorem języka JavaScript. Oznacza to, że poprawnie napisany kod JavaScript będzie także w pełni poprawnym kodem TypeScript. Rozszerza on możliwości JavaScript o statyczne typowanie zmiennych, funkcji czy obiektów. Daje to możliwość wykrycia błędów kodu już na etapie jego kompilacji. Wprowadza również takie elementy jak interfejsy, klasy czy dziedziczenie. Przyspiesza on pracę i ułatwia tworzenia dużych, skomplikowanych aplikacji nastawionych na skalowalność [18]. Wybór tej technologii podyktowany jest łatwością zaimplementowania, dużą popularnością i bogatą dokumentacją techniczną.

### 4.2.2. React

React to biblioteka JavaScript służąca do budowania interfejsów użytkownika. Używany jest głównie do tworzenia aplikacji jednostronicowych, czyli takich które nie wymagają przeładowywania strony w trakcie jej użytkowania. Sceny, aplikacje czy strony składają się z komponentów, które są funkcjami JavaScript [19]. Kod React jest pisany w JavaScript lub TypeScript. Wprowadza on składnię JSX, która rozszerza składnię JavaScript. Przypomina ona język XML lub HTML. Dzięki temu, że jest to biblioteka o otwartym kodzie źródłowym, powstaje wiele innych bibliotek tworzonych przez społeczność. Oferują one gotowe komponenty lub funkcjonalności [20]. Ze względu na jej popularność, łatwość użycia oraz wydajność została użyta w implementacji aplikacji.

### 4.2.3. Material UI

Material UI jest biblioteką komponentów React o otwartym kodzie źródłowym. Dostarcza ona kolekcję predefiniowanych komponentów gotowych do użycia. Biblioteka umożliwia ich personalizację pod potrzeby programistów [21]. Biblioteka została użyta ze względu na jej znajomość przez autora pracy, dużą społeczność deweloperską, a także bogatą i dokładną dokumentację techniczną i popularność. Pozwoliło to na szybkie wdrożenie tej biblioteki w projekt i ułatwiło rozwiązywanie napotkanych problemów podczas implementacji. Użycie niniejszej biblioteki znacznie skróciło proces tworzenia interfejsu użytkownika, dzięki jednolitemu stylowi dostarczanych komponentów.

### 4.2.4. i18next

Do zaimplementowania internacjonalizacji aplikacji użyto biblioteki i18next. Jest to popularny framework używany do tłumaczeń aplikacji internetowych. Upraszcza on obsługę wielu języków w aplikacji oferując wiele przydatnych do tego funkcji. Zdecydowano użyć niniejszej biblioteki ze względu na łatwość użycia oraz dużą popularność, co ułatwiło znalezienie materiałów pomocnych w włączeniu jej do aplikacji.

## 4.3. Technologie serwerowe (backend)

W celu implementacji struktury serwerowej aplikacji powinno używać się stabilnych i wydajnych rozwiązań. Pozwola to na uzyskanie oprogramowania, które w pełni będzie spełniać wymagania użytkowników.

Podczas doboru technologii użytych do stworzenia aplikacji serwerowej autor zadbał, by były to rozwiązania o stabilnej pozycji na rynku, aktywnym wsparciu, a także oferujące najwyższą wydajność, bezpieczeństwo i skalowalność. Jednocześnie kierowano się doświadczeniem autora w zakresie łatwości używania danych technologii.

W niniejszym podrozdziale zostaną wymienione technologie serwerowe użyte podczas implementacji tworzonego systemu bibliotecznego.

### 4.3.1. Java

Java jest obecnie jednym z najbardziej popularnych języków programowania [22]. Posiada bogatą bibliotekę podstawową oraz wsparcie dla wielowątkowości. Jest to dojrzały i stabilny ekosystem, z dużą społecznością i wsparciem ze strony twórców. Zapewnia bezpieczeństwo, a także solidność i niezawodność. Z tych wszystkich względów oraz doświadczenia autora związane z pracą z tym językiem programowania postanowiono, że aplikacja serwerowa będzie oparta o ten język.

### 4.3.2. Spring

Spring to framework języka Java. Oferuje on dużo bibliotek i rozszerzeń, które przyspieszają pracę nad kodem Java. Użyte zostaną między innymi Spring Boot do automatycznej konfiguracji i łatwości uruchomienia, Spring Data do obsługi warstwy dostępu do bazy danych czy Spring Security do konfiguracji bezpieczeństwa aplikacji. Jest to wydajne, elastyczne i bezpieczne rozwiązanie. Pozwala na budowanie wielu rodzajów aplikacji, niezależnie od zastosowania. Wspiera asynchroniczne operacje. Spring został wybrany ze względu na łatwość używania, dużą społeczność programistów, a także na mnogość oferowanych funkcji.

### 4.3.3. Hibernate

Do pracy z bazą danych za przydatny uznany został framework Hibernate. Jest to jedna z wielu implementacji Java Persistence API. Służy on do mapowania obiektowo-relacyjnego (ORM). Pozwala to mapować klasy Java na tabele bazy danych. Dzięki temu łatwiej jest stworzyć strukturę bazy danych oraz zachować czysty i czytelny kod. Dodatkowo pozwala to zmniejszyć ilość błędów podczas procesu tworzenia aplikacji dzięki automatycznemu generowaniu kodu zapytań do bazy danych.

## 4.4. Baza danych

Jako odpowiedni system zarządzania relacyjną bazą danych wybrano MariaDB. Jest to darmowy system zarządzania bazą danych o otwartym kodzie źródłowym. Oferuje on wysoką wydajność zarówno dla dużych jak i małych baz danych. Jest to skalowalne i bezpieczne rozwiązanie. Jako jeden z niewielu systemów zarządzania bazą danych posiada wbudowane szyfrowanie danych [23]. Idealnie nadaje się do różnego rodzaju aplikacji, zarówno internetowych jak i mobilnych.

## 4.5. Kontrola wersji

Za pomocą oprogramowania do śledzenia i kontroli wersji można śledzić zmiany w kodzie i efektywnie nimi zarządzać. W razie pomyłki, można cofnąć wersję do poprzedniej działającej i łatwo wprowadzić poprawki. Do kontroli wersji w przypadku tworzonej aplikacji zdecydowano się użyć systemu kontroli wersji Git.

Git jest oprogramowaniem o otwartym kodzie źródłowym. Śledzi on zmiany w lokalnym środowisku. Następnie, w razie potrzeby, zmiany te można zapisać w chmurze za pomocą serwisu Github. Dzięki temu możliwa jest praca na kilku urządzeniach. Używanie Git jest proste. Aby skorzystać z podstawowych funkcji tego programu wystarczy zapoznać się z kilkoma komendami lub użyć aplikacji oferującej graficzny interfejs użytkownika. Autor pracy zdecydował się na użycie systemu kontroli wersji ze względu na potrzebę śledzenia historii zmian i pracy nad kodem na kilku urządzeniach. Z uwagi na bezpieczeństwo, popularność oraz doświadczenie autora z Git, został on wybrany jako preferowany system kontroli wersji.

## 4.6. Inne narzędzia programistyczne

Podczas pracy nad aplikacją przydatne są różne narzędzia programistyczne. Autor zdecydował się ograniczyć ilość używanych narzędzi do dwóch kluczowych: Postman oraz Chrome DevTools, pomimo dostępności wielu innych narzędzi tego typu. Pierwsze z nich zostało użyte do testowania aplikacji serwerowej. Drugie narzędzie posłużyło do przetestowania aspektów związanych z aplikacją internetową.

### 4.6.1. Postman

Postman to narzędzie służące do testowania API. Podczas pracy nad aplikacją było używane do tworzenia i testowania zapytań API. Miało to na celu analizę zwracanych przez nie wartości oraz wydajności. Dzięki temu narzędziu możliwe było przetestowanie aplikacji serwerowej bez interfejsu użytkownika i aplikacji klienckiej.

### 4.6.2. Chrome DevTools

Chrome DevTools to narzędzie wbudowane w przeglądarkę internetową Google Chrome. Służy do diagnostyki i debugowania aplikacji JavaScript. Narzędzie to zostało użyte do testowania stworzonej struktury stron i ich stylów aplikacji React. Dzięki konsoli, narzędziu do wykonywania kodu JavaScript, logowane były błędy. Konsola posłużyła także do sprawdzania poprawności kodu TypeScript.

# 5. Implementacja

Po zakończeniu etapu projektowania oraz wyborze odpowiednich technologii przystąpiono do etapu implementacji. To w tej fazie projekt został urzeczywistniony i powstała prawdziwa, działająca aplikacja. Jest to najdłuższy etap podczas tworzenia oprogramowania. Przed przystąpieniem do tego etapu należało upewnić się, że wszelkie założenia zostały uwzględnione w fazie projektowania, aby uniknąć niepotrzebnych opóźnień spowodowanych zmianą wizji co do przeznaczenia i działania tworzonego systemu.

Podczas tworzenia aplikacji postanowiono najpierw napisać kod części serwerowej (backend), a następnie kod części klienckiej (frontend). Powodem takiej decyzji jest fakt, że logika biznesowa i baza danych są najważniejszą częścią całej aplikacji. Można je przetestować bez gotowego interfejsu aplikacji internetowej.

## 5.1. Aplikacja serwerowa

Serwer jest najważniejszą częścią systemu. To tutaj odbywa się odbieranie, przetwarzanie żądań użytkowników i wysyłanie odpowiedzi zwrotnych. Wszelkie funkcje związane z dostępem do bazy danych, takie jak logowanie czy rejestracja są obsługiwane przez część serwerową. Żadna aplikacja nie może być uznana za kompletną bez tej części.

### 5.1.1. Struktura projektu Spring

Aby ułatwić pracę nad implementacją, zadbano o przejrzystą i logiczną strukturę katalogów w tworzonym projekcie. Odpowiednie nazwy folderów i plików pozwoliły na łatwiejsze rozbudowywanie projektu i pomogło w poruszaniu się w nim. Zmniejszyło to również czas potrzebny na przeszukiwanie zbiorów plików w poszukiwaniu interesujących autora fragmentów kodu czy plików. Praca nad kodem w uporządkowanej strukturze projektu była szybsza, bowiem nie trzeba było zastanawiać się, gdzie należy zmodyfikować, dodać lub usunąć kod. Ułożenie katalogów i plików oraz ich nazwy w znaczący sposób poprawiły organizację pracy.

Rysunek 5.1 przedstawia widok folderów aplikacji serwerowej. Kluczowe dla całej aplikacji serwerowej klasy znajdują się w folderach *controllers*, *services* oraz *repositories*. W katalogu *models* znajdują się klasy odpowiadające za strukturę danych w aplikacji. Klasy te są następnie mapowane przez system ORM na tabele bazy danych. Folder *config* zawiera klasy odpowiedzialne za konfigurację bezpieczeństwa aplikacji. Klasy związane z uwierzytelnianiem użytkowników są ulokowane w katalogu *auth*. Folder *exceptions* zawiera zdefiniowane przez autora klasy wyjątków. Używane są one w celu precyzyjniejszego reagowania na różne scenariusze podczas działania aplikacji i zapewnieniu lepszej kontroli nad błędami w niej występującymi. Aby uniknąć niejednoznacznych wartości lub literówek, które mogą wystąpić podczas pracy z ciągami znaków, stworzono folder enums. Zawiera on pliki z enumeracjami, czyli typami wyliczeniowymi, które definiują zestawy stałych wartości.

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Rysunek 5.1. Struktura katalogów aplikacji serwerowej  
*źródło: opracowanie własne*

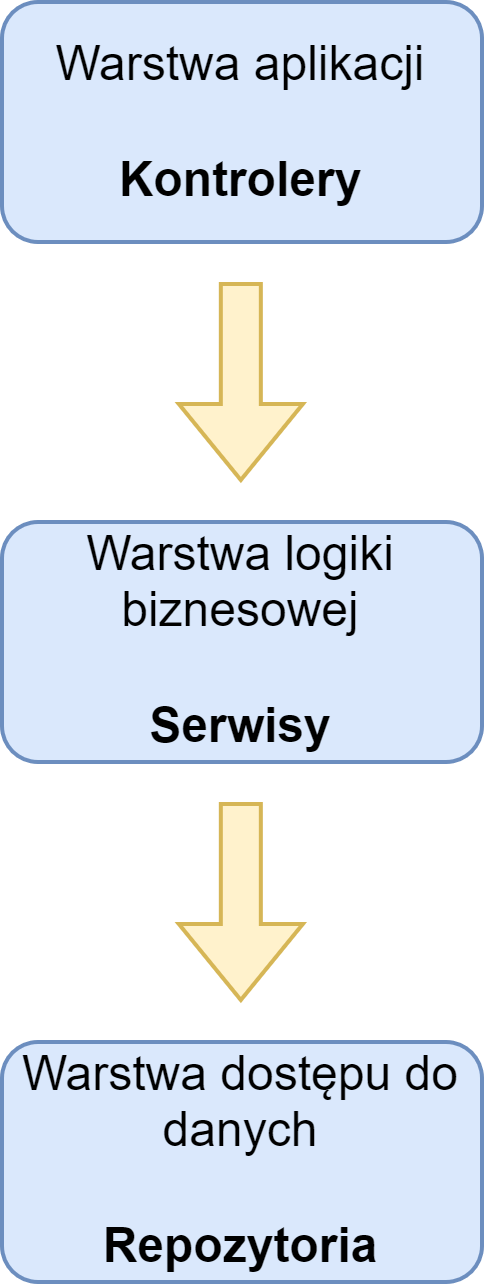
W strukturze katalogów przedstawionej na Rysunku 5.1 można zauważyć, że została ona zaprojektowana tak, by łatwo można było oddzielić klasy odpowiedzialne za poszczególne elementy aplikacji. Jest to związane z architekturą warstwową aplikacji, która została zastosowana w celu lepszej organizacji kodu.

### 5.1.2. Architektura warstwowa aplikacji

Model warstwowy to jeden z najpopularniejszych modeli architektury stosowanych w aplikacjach [24]. Dzieli on aplikację na techniczne warstwy. Stosuje się go w celu separacji odpowiedzialności. Każda z warstw odpowiada za inny element w aplikacji. Ich nazwy są dowolne, nie ma żadnego standardu, który określałby jak należy je nazywać. Każda z tych warstw zależy od poprzedniej.

W aplikacji serwerowej wyznacznikiem warstw są pakiety, które w projektach Java służą do grupowania klas. W aplikacji zastosowano podział na trzy warstwy, które składają się z kontrolerów, serwisów i repozytoriów:

* warstwa aplikacji – w tej warstwie znajdują się kontrolery. Odpowiadają ona za wywoływanie odpowiednich metod serwisów. Kontrolery przyjmują parametry, przetwarzają je a następnie przekazują je serwisom;
* warstwa logiki biznesowej – tutaj znajdują się serwisy, które zawierają całą logikę biznesową aplikacji. Serwisy wywołują metody z repozytoriów;
* warstwa dostępu do danych – tutaj odbywają się operacje na danych związane z bazą danych. Odpowiedzialne są za to repozytoria.

Na Rysunku 5.2 przedstawiono schemat przedstawiający architekturę warstwową aplikacji.

Rysunek 5.2. Schemat architektury warstwowej aplikacji  
*źródło: opracowanie własne*

Taki podział projektu aplikacji sprawił, że praca nad nią była łatwiejsza. Autor nie musiał zastanawiać się, gdzie stworzyć nową klasę. Jeśli kod w klasie był związany z ekspozycją funkcjonalności oznaczało to, że należy do warstwy kontrolerów. W przypadku unikalnej logiki biznesowej, należało utworzyć nowy serwis. Jeśli kod operował na danych z bazy danych, należało go umieścić w klasie w warstwie repozytoriów. Ważnym aspektem był fakt, że dla każdej encji w bazie danych należało stworzyć osobny kontroler, serwis i repozytorium. Osobne serwisy czy kontrolery stworzono także dla funkcjonalności związanymi z zapisem lub odczytem plików. Umożliwiło to na uzyskanie jeszcze klarowniejszej separacji odpowiedzialności konkretnych komponentów aplikacji serwerowej.

### 5.1.3. Repozytoria

Repozytoria są odpowiedzialne za wszelkie operacje związane z bazą danych. Spring zapewnia gotowe narzędzie do pracy z bazą danych, czyli Spring Data JPA. Jest to moduł jednego z najpopularniejszych komponentów frameworka Spring, czyli Spring Data. Pozwala on na zmniejszenie potrzeby pisania kodu związanego z dostępem do bazy danych praktycznie do minimum [25].

Dzięki Spring Data stworzenie repozytorium odpowiedzialnego za operacje na danych na konkretnej tabeli jest bardzo proste. Aby to zrobić należy stworzyć interfejs rozszerzający inny interfejs – *JpaRepository* (interfejs zapewniający API dla podstawowych operacji CRUD (z ang*. create, read, update, delete*) oraz sortowania). Jako, że interfejs *JpaRepository* jest generyczny należy podać klasę domenową, którą ma obsługiwać oraz typ identyfikatora [26]. Klasa domenowa to klasa Java, która reprezentuje tabelę bazy danych. Po ustaleniu tych parametrów programista ma dostęp do metod implementujących podstawowe operacje dostępu do bazy danych, takich jak *findAll*, która zwraca listę wszystkich rekordów z danej tabeli lub *deleteById(ID id)*, która jako parametr przyjmuje identyfikator rekordu, który ma zostać usunięty.

Dodatkowo, Spring Data JPA daje możliwość definiowania własnych metod w interfejsach repozytoriów za pomocą pewnej konwencji nazewniczej [27]. Procedurę tę realizuje się poprzez zdefiniowanie nazw tych metod w określony sposób, umożliwiający parserowi wbudowanemu w Spring Data odpowiednie zrozumienie działania danej metody. Nie wdając się w szerokie możliwości tej funkcjonalności, należy nazwać metodę w taki sposób, by jej nazwa zaczynała się od słów *findBy*. Następnie trzeba podać nazwę istniejącego pola w klasie domenowej, którą dane repozytorium obsługuje. W taki sposób zdefiniowana będzie metoda, która wykona wyszukiwanie w obsługiwanej tabeli na podstawie podanego pola. Dzięki temu mechanizmowi zapytania do bazy danych generowane są automatycznie na podstawie nazw metod repozytorium.

Przedstawiony poniżej kod jest implementacją jedno z repozytoriów stworzonego podczas pracy nad aplikacją.

@Repository (1)  
public interface UserRepository extends JpaRepository<User, Long> { (2)  
 Optional<User> findByEmail(String email); (3)  
 Optional<User> findByRefreshToken(String refreshToken); (4)  
} (5)

Jest to interfejs o nazwie *UserRepository*, rozszerzający interfejs generyczny *JpaRepository*, który obsługuje klasę *User*. Klasa ta reprezentuje tabelę *user* w bazie danych, czyli przechowuje informacje o użytkownikach. Typ identyfikatora w tej klasie to *Long*. Interfejs posiada zdefiniowane dwie metody zapytań specyficznych dla jego klasy domenowej. Obie te metody służą do wyszukiwania jednego rekordu użytkownika w bazie danych na podstawie przekazanego do nich parametru. Zwracają one obiekt parametryzowanej klasy *Optional*. W zależności czy rekord zostanie znaleziony w bazie danych obiekt ten będzie przechowywał pustą wartość lub obiekt klasy podanej jako parametr, w tym przypadku jest to klasa *User*. Metoda *findByEmail* przyjmuje parametr email typu *String* i generuje odpowiednie zapytanie, które następnie jest wykonywane w celu przeszukania bazy danych w poszukiwaniu rekordu zawierającego podaną wartość w polu *email*. Analogicznie działa metoda *findByRefreshToken*. Implementacja tych metod została wygenerowana automatycznie przez Spring Data JPA i autor nie był zobowiązany do podejmowania dodatkowych działań, aby zaczęły one działać. Należało jedynie upewnić się, że nazwy tych metod są utworzone poprawnie. Adnotacja *@Repository* wskazuje, że ten interfejs dostarcza mechanizm odpowiedzialny za operację związanymi z bazą danych [28].

Bardziej złożone zapytania zostały zaimplementowane przy użyciu specjalnej adnotacji Spring Data JPA *@Query* [29]. Pozwala ona określić zapytanie, które adnotowana metoda ma wykonać za pomocą języka JPQL lub SQL. W ten sposób zaimplementowano filtrowanie wyników w potrzebnych do tego miejscach aplikacji. Według wymagań funkcjonalnych, aplikacja miała umożliwiać filtrowanie wypożyczeń użytkowników. Jedno z takich zapytań do bazy danych implementuje poniższa metoda *findAllByUserIdAndReturned* interfejsu *BookLoanRepository*, repozytorium powiązanym z tabelą *book\_loan* bazy danych.

@Query( (1)

"SELECT bl FROM BookLoan bl " + (2)

"WHERE (:userId IS NULL OR bl.user.id = :userId) " + (3)

"AND (:returned IS NULL OR bl.returned = :returned) " + (4)

"AND (bl.book.isEbook = false) " + (5)

"AND (:accepted IS NULL OR bl.accepted = :accepted)") (6)

Page<BookLoan> findAllByUserIdAndReturned( (7)

@Param("userId")Long userId, @Param("returned") Boolean returned, (8)

@Param("accepted") Boolean accepted, Pageable pageable); (9)

} (10)

Metoda przyjmuje opcjonalne parametry *userId*, *returned* oraz *accepted*. Jeśli parametry te są niezerowe, czyli jest znana ich wartość, zapytanie zwraca wypożyczenia, które są zgodne z ich wartościami. W przeciwnym razie są one pomijane i nieuwzględniane w zapytaniu. Parametry te są przesyłane przez klienta, w przypadku tej funkcjonalności zawsze będzie to administrator aplikacji. Przykładowo dla parametru *userId* o wartości 7 oraz parametru *returned* o wartości *true* zwrócone zostaną wszystkie zakończone wypożyczenia użytkownika o identyfikatorze 7. Ostatni parametr *pageable* powiązany jest z mechanizmem paginacji, jednak w kontekście definiowania zapytania za pomocą adnotacji *@Query* jest nieistotny.

Dzięki Spring Data oraz Spring Data JPA działanie z bazą danych w aplikacji Java jest bardzo uproszczone i przyspieszone. Korzystanie z metod do operacji na tabelach bazy danych sprowadza się zazwyczaj do definiowania nowego interfejsu i wywoływania jego metod. To zapewnia dostęp do najczęściej używanych operacji, a w razie potrzeby możliwe jest również łatwe dodanie nowych metod.

### 5.1.4. Serwisy

Warstwa serwisów pełni rolę w obszarze logiki biznesowej, gdzie wywoływane są metody z repozytoriów, stanowiące integralną część innego kodu. Wszelkie funkcjonalności określone w wymaganiach funkcjonalnych są implementowane w metodach serwisów, po czym wywoływane są przez kontrolery. W niektórych przypadkach zdarza się, że jeden serwis może wywołać metodę innego serwisu.

Spring udostępnia programistom adnotację *@Service*, by oznaczyć komponenty odpowiedzialne za logikę biznesową aplikacji.

Stosując architekturę warstwową w aplikacji , stworzono osobny serwis dla każdej tabeli w bazie danych. Wyjątkiem były sytuacje, gdzie potrzebne było pojedyncze użycie danych z innej tabeli. W takich przypadkach wywoływano metody z odpowiedniego repozytorium.

Poniższy kod metody acceptLoan implementuje proces potwierdzania wypożyczenia.

public ResponseEntity<BookLoan> acceptLoan(Long loanId) { (1)

try { (2)

var bookLoan = bookLoanRepository.findById(loanId) (3)

.orElseThrow( (4)

() -> new EntityNotFoundException("No bookloan found with id: " + (5)

loanId)); (6)

bookLoan.setAccepted(true); (7)

bookLoan.setLoanDate(LocalDate.now()); (8)

bookLoan.setDueDate(bookLoan.getLoanDate().plusMonths(1)); (9)

bookLoanRepository.save(bookLoan); (10)

notificationService.sendBookLoanAcceptedNotification( (11)

bookLoan.getId()); (12)

return ResponseEntity.ok(bookLoan); (13)

} catch (EntityNotFoundException e) { (14)

return ResponseEntity.notFound().build(); (15)

} (16)

} (17)

Metoda przyjmuje jeden parametr *loanId* typu *Long*. Jest to identyfikator wypożyczenia. Jest on użyty do znalezienia odpowiedniego wypożyczenia w bazie danych. W tym celu wywoływana jest metoda *findById* z odpowiedniego repozytorium, w tym przypadku *BookLoanRepository*. Jeśli nie zostanie znaleziony żaden rekord, generowany jest wyjątek *EntityNotFoundException* i zwracana jest odpowiedź, czyli obiekt klasy *ResponseEntity* z odpowiednim statusem HTTP. W przeciwnym wypadku, metoda kontynuuje działanie, ustawiając pola *accepted*, *loanDate* oraz *dueDate* za pomocą *setterów*, czyli metod przypisującym wartość prywatnym polom klas. Pole *accepted* to flaga *boolean* przyjmująca wartość *true*, gdy wypożyczenie zostanie zaakceptowane i *false*, gdy zostanie odrzucone. Pola *loanDate* i *dueDate* to kolejno data wypożyczenia oraz data zwrotu, czyli data, gdy dany materiał biblioteczny powiązany z wypożyczeniem powinien zostać zwrócony. Następnym krokiem jest wywołanie metody serwisu powiadomień odpowiedzialnej za wysłanie powiadomienia o zaakceptowanym wypożyczeniu. Powiadomienie zostaje wysłane użytkownikowi powiązanemu z danym wypożyczeniem. W przypadku, gdy metoda nie napotka błędu i wszystkie operacje zakończą się sukcesem, zwracany jest odpowiedni status HTTP.

### 5.1.5. Kontrolery

Jedną z ostatnich warstw w aplikacji jest warstwa kontrolerów. Kontrolery są odpowiedzialne za wywoływanie odpowiednich metod serwisów i ich ekspozycję. Ponadto, odpowiadają za komunikację z klientem poprzez przetwarzanie zapytań HTTP oraz przesyłanie odpowiedzi w formacie JSON. Odpowiedź ta może zawierać dane z bazy danych, informacje o błędach lub inne informacje zwrotne zależne od logiki biznesowej w serwisach.

W Spring budowanie kontrolera jest proste i wymaga małej ilości konfiguracji. Należy użyć odpowiednich adnotacji. W aplikacji, podczas implementacji kontrolerów, użyto adnotacji *@RestController*. Jest to adnotacja, która sprawia, że klasy nią opatrzone są traktowane jak kontrolery, których metody domyślnie zwracają obiekty klas domenowych zamiast widoków [30][31]. W każdym kontrolerze użyto również adnotacji *@RequestMapping* na poziomie klasy, by zmapować wspólny adres URL dla wszystkich metod obsługujących zapytania HTTP [32]. Użyto również adnotacji Lombok *@RequiredArgsConstructor* w celu automatycznego generowania konstruktora klasy. Tworzy ona konstruktor z parametrem dla każdego niezainicjalizowanego pola finalnego [33]. Dzięki temu po raz kolejny zmniejszono ilość szablonowego kodu, który trzeba by było pisać w każdym z kontrolerów czy serwisów, chcąc odwołać się do metod z repozytoriów czy innych serwisów. Przy każdej z metod kontrolera zastosowano odpowiednie adnotacje mapujące je do określonych metod HTTP.

Poniżej znajduje się fragment kodu kontrolera powiadomień.

@RestController (1)

@RequiredArgsConstructor (2)

@RequestMapping(path="api/v1/notifications") (3)

public class NotificationController { (4)

private final NotificationService notificationService; (5)

@GetMapping("/user/{userId}") (6)

public List<Notification> getUserNotifications( (7)

@PathVariable Long userId) { (8)

return notificationService.getUserNotifications(userId); (9)

} (10)

@PostMapping("/reminder/{bookLoanId}") (11)

public ResponseEntity<Notification> sendReminderNotification( (12)

@PathVariable Long bookLoanId) { (13)

return notificationService.sendReminderNotification( (14)

bookLoanId); (15)

} (16)

//reszta metod kontrolera (17)

} (18)

Za pomocą adnotacji *@RequestMapping* zdefiniowano wspólny adres. Każda z metod została zmapowana do odpowiedniego żądania HTTP za pomocą skróconych wersji adnotacji *@RequestMapping*. Dodatkowo, gdy było to potrzebne, każdej metodzie zdefiniowano rozwinięcie wspólnego adresu API.

Metoda *getUserNotifications*, służąca do pobrania wszystkich nieodczytanych powiadomień użytkownika, została zmapowana pod adres *api/v1/notifications/user/{userId}*. W nawiasach klamrowych umieszczona została zmienna ścieżki. Ma ona taką samą nazwę jak przyjmowany przez metodę parametr typu *Long*, oznaczony adnotacją *@PathVariable*. Wskazuje ona, że parametr powinien zostać powiązany ze zmienną umieszczoną w ścieżce o takiej samej nazwie [34]. Zmienna ta przyjmuje różne wartości i jest ustawiana przez aplikację kliencką. Dzięki temu można tworzyć szablony adresu zapytań do API, co pozwala na dynamiczne generowanie adresów URL w zależności od aktualnych potrzeb aplikacji internetowej. W tym przypadku, metoda służy do pobrania nieodczytanych powiadomień dowolnego użytkownika. Metoda kontrolera wywołuje odpowiednią metodę serwisu *NotificationService* i przekazuje jej parametr *userId*.

### 5.1.6. Modele

Dzięki użyciu Spring Data JPA w implementacji aplikacji serwerowej praktycznie do zera zmniejszono ilość kodu SQL. Tworzenie tabel wykonywane jest automatycznie. Dzieje się tak, ponieważ klasy domenowe są mapowane na tabele bazy danych przez JPA (z ang. *Java Persistence API*). Na podstawie odpowiednich adnotacji framework Hibernate tworzy tabele i kolumny zgodne ze zdefiniowanymi mapowaniami. Rysunek 5.3 przedstawia uproszczone działanie tego procesu na podstawie klasy *Notification*, która odpowiada tabeli *notification* w bazie danych. Odpowiada ona za przechowywanie informacji o powiadomieniach użytkowników. Na potrzeby czytelności rysunku pominięto adnotacje niezwiązane z mechanizmem mapowania klas na tabelę.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek 5.3. Schemat mapowania klasy Java na tabelę bazy danych przez Hibernate  
*źródło: opracowanie własne*

Adnotacje mogą określać ograniczenia, relacje z innymi tabelami lub rozmiar i typ danych. Mogą być używane zarówno na poziomie klas, jak i pól. Każde z nich może zawierać jedną lub więcej adnotacji.

Każda z adnotacji widocznych na Rysunku 5.3 spełnia określoną funkcję w mapowaniu kodu Java na tabelę w bazie danych. Adnotacja *@Entity* użyta na poziomie klasy domenowej określa, że klasa na której została użyta jest encją [35]. Na podstawie jej nazwy w bazie danych tworzona jest tabela o takiej samej nazwie. Za określenie klucza podstawowego odpowiedzialna jest adnotacja *@Id* [36]. *@GeneratedValue* jest adnotacją, która pozwala wyznaczyć strategię automatycznego generowania klucza podstawowego [37]. Przy polach *recipient*, *bookLoan* oraz *book* umieszczone zostały adnotacje *@ManyToOne* oraz *@JoinColumn*. Pierwsza z nich służy do określenia asocjacji wiele do jednego z inną klasą encji [38]. Druga określa kolumnę do łączenia asocjacji [39]. W rezultacie w tabeli powstają kolumny będące kluczami obcymi o nazwach zdefiniowanymi w parametrze *name* adnotacji. *@CreationTimestamp* oznacza adnotowane pole jako znacznik czasu utworzenia [40]. Służy do automatycznego definiowana domyślnej wartości pola *timeStamp*. Wartość jest generowana podczas wstawiania danego wiersza do tabeli. Adnotacja *@Enumerated* informuje, że pole *notificationType* powinno być przechowywane w bazie danych jako typ wyliczeniowy [41]. Wszystkie te adnotacje są następnie używane przez Hibernate w celu automatycznego wygenerowania kodu SQL tworzącego tabelę oddającą strukturę klasy domenowej.

Użycie systemu ORM pozwoliło na zredukowanie błędów podczas operacji na bazie danych. Automatyzacja tego procesu usprawniła pisanie aplikacji i pozwoliła skupić się na implementacji kluczowych funkcjonalności.

### 5.1.7. Bezpieczeństwo aplikacji

Podczas implementacji aplikacji serwerowej zadbano o jej bezpieczeństwo. Autoryzacja i uwierzytelnianie zostało wdrożone za pomocą Spring Security, czyli komponentu frameworku Spring. Zapewnia on ochronę przed typowymi atakami informatycznymi. Jest uznawany za standard zabezpieczania aplikacji opartych o framework Spring [42].

W celu zaimplementowania mechanizmów uwierzytelniania i autoryzacji zastosowano standard JSON Web Token (JWT). Kod odpowiedzialny za bezpieczeństwo aplikacji zawiera się głównie w trzech klasach: *SecurityConfig*, *JwtAuthenticationFilter* oraz *JwtService*. Każda z nich ma inne zadanie. Cała logika związana z logowaniem i rejestrowaniem się użytkownika w systemie znajduje się w klasie *AuthService*.

Klasa *SecurityConfig* odpowiada za konfigurację zabezpieczeń aplikacji serwerowej. W klasie tej definiowane są odpowiednie ustawienia dotyczące bezpieczeństwa i natury działania aplikacji ze względu na użycie JWT. Wyłącza ona ochronę przed atakami CSRF (z ang. *Cross-Site Request Forgery*). Jako, że aplikacja jest bezstanowa, a tokeny przesyłane są w nagłówkach HTTP, do których atakujący nie ma łatwego bezpośredniego dostępu, nie jest ona wymagana. Konfigurowana jest również autoryzacja żądań HTTP. Klasa określa zasady dostępu do punktów końcowych API. Punkty końcowe związane z uwierzytelnianiem oraz tokenem odświeżania nie wymagają uwierzytelniania. Wszystkie inne punkty końcowe API wymagają uwierzytelnienia. Dodatkowo, zapytania *DELETE* i *PUT* do punktu końcowego */api/v1/books*, czyli związanego z materiałami bibliotecznymi, wykonywać może tylko użytkownik z rolą administratora. Dzięki temu, żaden użytkownik, bez względu na rolę, nie może uzyskać dostępu do zasobów aplikacji bez wcześniejszego uwierzytelnienia i autoryzacji.

Określona jest również polityka tworzenia sesji. Jest ona bezstanowa. Oznacza to, że między żądaniami użytkownika aplikacja nie będzie przechowywać żadnych stanów sesji. Następnie wykonywana jest konfiguracja klasy dostarczającej uwierzytelnienie, dodane jest stosowanie filtra JWT, który zostanie omówiony w dalszej części podrozdziału 5.1.7 oraz konfiguracja wylogowania.

Klasa *JwtAuthenticationFilter* służy do ustawienia filtra uwierzytelniającego żądania HTTP przy użyciu tokenów JWT. Dziedziczy ona po klasie *OncePerRequestFilter*, która gwarantuje wykonanie jej metody *doInternalFilter* przy każdym żądaniu HTTP [43]. Metoda ta pobiera nagłówek autoryzacji z żądania HTTP, a następnie sprawdza jego poprawność. W przypadku potwierdzenia tego, że nagłówek jest poprawny, metoda wyciąga z niego token JWT. Następie wyodrębniana jest z niego zakodowana nazwa użytkownika, czyli adres e-mail. Metoda sprawdza czy użytkownik o tym adresie e-mail jest już uwierzytelniony w systemie. W przypadku braku uwierzytelnienia, sprawdzana jest poprawność tokenu i ustawiane są szczegóły uwierzytelniania. Następnie żądanie jest przetwarzane przez inne filtry w łańcuchu.

W klasie *JwtService* znajdują się metody odpowiedzialne za operacje związane z JWT. Są w niej metody generujące tokeny JWT i tokeny odświeżające, wyodrębniające roszczenia z tokenów, pozyskujące klucz do ich podpisania i weryfikacji na podstawie klucza tajnego czy sprawdzające poprawność i ważność. Wszystkie metody zaimplementowane zostały z pomocą biblioteki JJWT (Java JSON Web Token). Metody te pozwoliły na stworzenie logiki odpowiedzialnej za uwierzytelnianie i autoryzację.

Tokeny JWT są generowane podczas logowania i rejestrowania się użytkownika. Gdy proces rejestracji użytkownika przebiegnie pomyślnie jest on domyślnie logowany do aplikacji i ma dostęp do jej zasobów. Za implementację logiki związanej z tymi funkcjonalnościami odpowiada klasa *AuthService*. Dostarcza ona metod obsługujących logowanie, rejestrację oraz dostarczenie tokenu odświeżającego użytkownikowi aplikacji klienckiej (w razie potrzeby).

Metoda *register* z serwisu autoryzacji otrzymuje jako parametr dane z formularza rejestracji z aplikacji internetowej. Wykorzystuje e-mail znajdujący się w otrzymanych danych, aby sprawdzić czy w bazie danych znajduje się już użytkownik o takim adresie e-mail. Jest to sytuacja niedozwolona, ponieważ adresy e-mail w bazie danych powinny być unikalne.

var userOptional = userRepository.findByEmail(request.getEmail()); (1)

if (userOptional.isPresent()) { (2)

return ResponseEntity.status(HttpStatus.CONFLICT). (3)

build(); (4)

} (5)

W celu sprawdzenia czy dany e-mail istnieje w bazie danych wywołana jest odpowiednia metoda repozytorium klasy domenowej *User*. Jeśli taki rekord zostanie znaleziony, działanie metody zostanie przerwane i zwrócona zostanie odpowiedź informująca o błędzie. W przeciwnym wypadku metoda kontynuuje działanie i sprawdza poprawność hasła podanego przez użytkownika w formularzu rejestracji.

String password = request.getPassword();

if (password.length() < 6 ||

!password.matches(".\*[a-z].\*") ||

!password.matches(".\*[A-Z].\*") ||

!password.matches(".\*\\d.\*") ||

!password.matches(".\*[^a-zA-Z0-9].\*")) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

Kolejnym procesem w realizacji rejestracji użytkownika jest wykonanie walidacji wprowadzonego przez niego hasła. W tym celu użyto wyrażeń regularnych, które pozwalają w wygodny sposób ustalić kryteria jakie musi spełnić ciąg znaków, by warunek logiczny został spełniony. Aby hasło zostało zaakceptowane musi spełnić następujące warunki:

* posiadać co najmniej 6 znaków;
* posiadać co najmniej jedną małą literę;
* posiadać co najmniej jedną wielką literę;
* posiadać przynajmniej jedną cyfrę;
* posiadać przynajmniej jeden znak specjalny niebędący literą ani cyfrą;

Gdy wszystkie te warunki zostaną spełnione, metoda kontynuuje swoje działanie. Następnym krokiem jest zbudowanie obiektu klasy domenowej *User* za pomocą danych przekazanych przez formularz HTTP w przeglądarce internetowej.

var user = User.builder() (1)

.firstName(request.getFirstName()) (2)

.lastName(request.getLastName()) (3)

.email(request.getEmail()) (4)

.password(passwordEncoder.encode(password)) (5)

.role(Role.USER) (6)

.build(); (7)

userRepository.save(user); (8)

Imię, nazwisko, adres e-mail są ustawiane na podstawie otrzymanych danych. Użytkownik zostaje przypisany do roli *USER*. Hasło jest najpierw szyfrowane, a dopiero potem przypisywane jest do użytkownika. Na koniec użytkownik zostaje zapisany w bazie danych przez wywołanie metody *save* z *UserRepository*.

Kolejnym krokiem jest wygenerowanie tokenu JWT oraz tokenu odświeżającego. W tym celu wywoływane są metody z klasy *JwtService* omawianej wcześniej w podrozdziale 5.1.7.

var jwtToken = jwtService.generateToken(user); (1)

var refreshToken = jwtService.generateRefreshToken(user); (2)

Następnie token odświeżający jest zapisywany w ciasteczku HTTP i dodawany jest do odpowiedzi, która zostanie zwrócona do przeglądarki.

Cookie refreshTokenCookie = new Cookie("refreshToken", refreshToken); (1)

refreshTokenCookie.setHttpOnly(true); (2)

refreshTokenCookie.setMaxAge( (3)

(int) ChronoUnit.SECONDS.between(LocalDateTime.now().atZone( (4)

ZoneId.systemDefault()), LocalDateTime.now().atZone( (5)

ZoneId.of("UTC")).plusDays(1))); (6)

refreshTokenCookie.setPath("/"); (7)

response.addCookie(refreshTokenCookie); (8)

Jego długość jest ustawiana na 24 godziny od momentu wygenerowania. Ciasteczko jest ustawione jako *httpOnly*. Sprawia to, że takie ciasteczko nie jest narażone na działanie kodów skryptowych po stronie klienta, takich jak JavaScript [44]. Pomaga to zapewnić bezpieczeństwo.

Na sam koniec klientowi zwracana jest odpowiedź w formacie JSON, która zawiera w sobie token JWT.

AuthenticationResponse authResponse = AuthenticationResponse.builder() (1)

.accessToken(jwtToken) (2)

.build(); (3)

return ResponseEntity.ok(authResponse); (4)

## 5.2. Aplikacja przeglądarkowa

Aplikacja przeglądarkowa jest tą częścią aplikacji, z którą użytkownik ma bezpośrednią interakcję. Udostępnia ona interfejs graficzny, który pozwala w wygodny i szybki sposób używać oferowanych przez nią funkcji. Jak wspomniano wcześniej w rozdziale 4, do implementacji tej części aplikacji użyto biblioteki JavaScript React służącej do budowania interfejsów użytkownika.

### 5.2.1. Struktura projektu React

Podobnie jak w aplikacji serwerowej, w projekcie React również zadbano o czytelną strukturę katalogów. Zastosowano podział według typu pliku. Pozwoliło to wprowadzić pewnego rodzaju separację odpowiedzialności. Struktura projektu jest łatwa do zrozumienia, a poruszanie się po nim jest szybkie i intuicyjne. Dodawanie nowych plików nie wymaga zastanawiania się, gdzie należy je umieścić. Na Rysunku 5.4. znajduje się widok katalogów w projekcie aplikacji klienckiej.

W folderze Api znajdują się pliki związane z konfiguracją komunikacji z aplikacją serwerową. Ustalany jest tam bazowy adres URL dla wszystkich zapytań. Tworzona jest także specjalna instancja axios o nazwie axiosPrivate. Stosowana jest ona do wykonywania zapytań do wszystkich chronionych punktów końcowych API serwera. Posiada ona specjalnie zdefiniowane w tym celu nagłówki HTTP oraz włączoną opcję umożliwiającą przekazywanie plików cookie między przeglądarką a serwerem. Folder Components przechowuje pliki związane z pojedynczymi komponentami React. Wszystkie zdefiniowane przez autora komponenty znajdują się w tym folderze. Do przechowywania plików definiujących stałe wartości, takie jak pliki użyte do zaimplementowania dwujęzykowości aplikacji, służy folder Constants. Bardzo ważnym katalogiem jest ten o nazwie Contexts. Znajdują się w nim pliki, w których zdefiniowane są konteksy React. Konteksty zostaną omówione w dalszej części podrozdziału 5.2. Równie istotny jest folder Hooks, który zawiera w sobie pliki odpowiadające za definiowanie hooków używanych potem na przestrzeni całej aplikacji. W katalogu Models znajdują się definicje typów i interfejsów opisujących kształt danych używanych w aplikacji. Widoki czyli kompletne strony, które potem widzi użytkownik, zbudowane z komponentów, znajdują się w folderze Pages. Pliki powiązane z motywem aplikacji przechowywane są w katalogu Theme. Różnego rodzaje funkcje pomocnicze znajdują się w katalogu Utils.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 5.4. Struktura katalogów w aplikacji React  
*źródło: opracowanie własne*

Nie ma jednego standardu struktury projektu React. Ważne jest, by struktura taka była prosta i czytelna. Struktura użyta przez autora pracy uwzględnia te cechy i nie jest skomplikowana. Wprowadzenie takiej struktury podyktowane było lepszą organizacją kodu, co pozytywnie wpływa na produktywność i poruszanie się w całym projekcie.

### 5.2.2. Konteksty React

Konteksty to jedno z najważniejszych zagadnień podczas pracy z biblioteką React. Pozwalają one na globalne zarządzanie stanami aplikacji. Umożliwiają przekazywanie różnego rodzaju danych przez całe drzewo komponentów bez potrzeby przekazywania parametrów do każdego komponentu osobno. Dzięki temu mechanizmowi o wiele łatwiej implementować wartości, które mają być takie same w całej aplikacji.

W aplikacji użyto kontekstów w celu zaimplementowania motywu jasnego i ciemnego, przechowywania informacji o aktualnie zalogowanym użytkowniku, routingu czy dynamicznych tłumaczeń. Niektóre z nich zostały stworzone przez autora pracy, inne zaś zostały dostarczone z odpowiednimi bibliotekami JavaScript.

### 5.2.3. Autoryzacja

Proces autoryzacji rozpoczyna się od pobrania informacji o użytkowniku podczas logowania lub rejestracji z aplikacji serwerowej. Te informacje są następnie przechowywane w pamięci aplikacji za pomocą kontekstu. Po pomyślnym przetworzeniu żądania logowania, serwer aplikacji wysyła odpowiedź w formacie JSON zawierającą token dostępu. Później token ten jest dekodowany w celu uzyskania informacji o roli, identyfikatorze użytkownika oraz nazwie użytkownika, czyli adresie e-mail. Na końcu dane te są użyte w celu ustawienia globalnego stanu kontekstu *AuthContext*. Dzięki temu dane te będą dostępne w dowolnym komponencie czy dowolnej stronie.

Jako, że token JWT po pewnym czasie traci ważność, nie może być on używany przez długi czas do uwierzytelniania żądań wykonywanych przez aplikację kliencką. W przypadku wygaśnięcia tokenu aplikacja serwerowa umożliwia wygenerowanie nowego tokenu dostępu poprzez podanie tokenu odświeżającego. Aplikacja React posiada zaimplementowany mechanizm wykorzystujący tę funkcję. Służy do tego hook *useRefreshToken*. Hook ten wykonuje zapytanie do punktu końcowego API *auth/refresh-token* serwera i w przypadku powodzenia żądania ustawia nowy token dostępu i przechowuje go w pamięci aplikacji. Jeśli serwer zwróci błąd z powodu nieważności tokenu odświeżania, jego braku lub innego powodu, użytkownik zostanie przekierowany do strony logowania z odpowiednim komunikatem o wygaśnięciu sesji użytkownika.

W celu zapewnienia stałego dostępu do aplikacji rozszerzono funkcjonalność instancji axios. Osiągnięto to poprzez integrację instancji *axiosPrivate* z mechanizmami uwierzytelniania w hooku *useAxiosPrivate*. Zapewnia on, że każde zapytanie wysłane z pomocą tej instancji posiada nagłówek *Authorization*. Nagłówek ten jest ustawiony na aktualny token dostępu. Jeśli odpowiedź zwróci status 403, a żądanie nie zostało jeszcze wykonane ponownie wysłane, hook ten używa funkcji zwracanej przez hook *useRefreshToken* i uzyskuje nowy token dostępu. Następnie nieudane żądanie jest wysyłane ponownie z nowym tokenem dostępu JWT. Funkcjonalność ta została zaimplementowana za pomocą przechwytywaczy (*interceptors*) biblioteki Axios. Pozwalają one na modyfikowanie i kontrolę żądań lub odpowiedzi HTTP zanim zostaną wysłane lub przetworzone [45].

### 5.2.4. Tłumaczenie

Aplikacja posiada dwa języki interfejsu do wyboru. Funkcja ta została zaimplementowana za pomocą biblioteki i18next.

W celu ustalenia możliwych języków stworzono stałą o nazwie *LANGUAGES* przechowującą tablicą obiektów. Każdy z tych obiektów reprezentuje dany język. Obiekty mają dwie właściwości: *label* (etykieta) oraz *code* (kod języka). Następnie w pliku i18n.ts dokonano konfiguracji biblioteki. W tym celu użyto wbudowanej funkcji *use* instancji i18n, która jako parametr przyjmuje moduł *initReactI18next*. Ułatwia to integrację z Reactem. Następnie użyto funkcji *init* instancji i18n do inicjalizacji z podanymi ustawieniami:

* *lng* – opcja ta domyślny ustawia język aplikacji. Domyślną wartości jest wartość klucza language przechowywane w pamięci przeglądarki. Jeśli w przeglądarce nie ma zapisanego języka, domyślnie zostanie ustawiony język polski;
* *fallbackLng* – język zapasowy. Ustawione na język polski podając kod języka „pl”;
* *interpolation* – opcja związana z interpolacją. Zastosowane ustawienie escapeValue na false oznacza, że interpolowane wartości nie będą uciekały znaków HTML. Potrzebne, by zabezpieczyć aplikację przed atakami XSS (Cross-Site Scripting);
* *resources* – służy do dostarczenia tłumaczeń. W implementacji aplikacji tłumaczenia w postaci klucz-wartość są przechowywane w oddzielnych plikach w formacie JSON.

Jako, że biblioteka oparta jest na kontekście, trzeba było w odpowiednim miejscu użyć dostawcy kontekstu I18next*. I18nextProvider* pobiera instancję i18next poprzez *prop* *i18n* i przekazuje ją za pomocą interfejsu API kontekstu. W tym celu należy przekazać *<App/>*, czyli główny komponent aplikacji, jako dziecko *I18nextProvider*.

<I18nextProvider i18n={i18n}> (1)

<App/> (2)

</I18nextProvider> (3)

Aby wyświetlane były różne ciągi znaków w zależności do języka, należy w odpowiedni sposób użyć dostarczonej przez bibliotekę funkcji *t*. Funkcja ta przyjmuje jako parametr klucz z obiektu tłumaczeń i zwraca przypisany do niego ciąg znaków. Aby można było użyć tej funkcji, należy w komponencie funkcyjnym użyć hooka *useTranslation*. Poniżej znajduje się przykładowe użycie.

<Link to="/register">{t("loginPage.noAccount")}</Link> (1)

Kod ten wyświetli hiperłącze biblioteki React Router przekierowujące do strony znajdującej się pod adresem */register*, czyli do strony z formularzem rejestracji konta użytkownika. Ciąg znaków, który zostanie wyświetlony zależny jest od aktualnie wybranego języka. W przypadku języka polskiego zostanie wyświetlony napis *Nie masz konta? Zarejestruj się*. Natomiast jeśli wybrany zostanie język angielski napis ten zmieni się na *No account? Register*.

Zmiana języka odbywa się w funkcji *toggleLanguage*. Użyta jest ona w komponencie *LanguageChooser*, który służy do wyświetlania aktualnie użytego języka i pozwala na jego zmianę. Poniżej znajduje się kod funkcji *toggleLanguage*.

const toggleLanguage = (e: SelectChangeEvent) => { (1)

const value = e.target.value; (2)

if (value) { (3)

i18n.changeLanguage(value); (4)

window.localStorage.setItem("language", value); (5)

} (6)

}; (7)

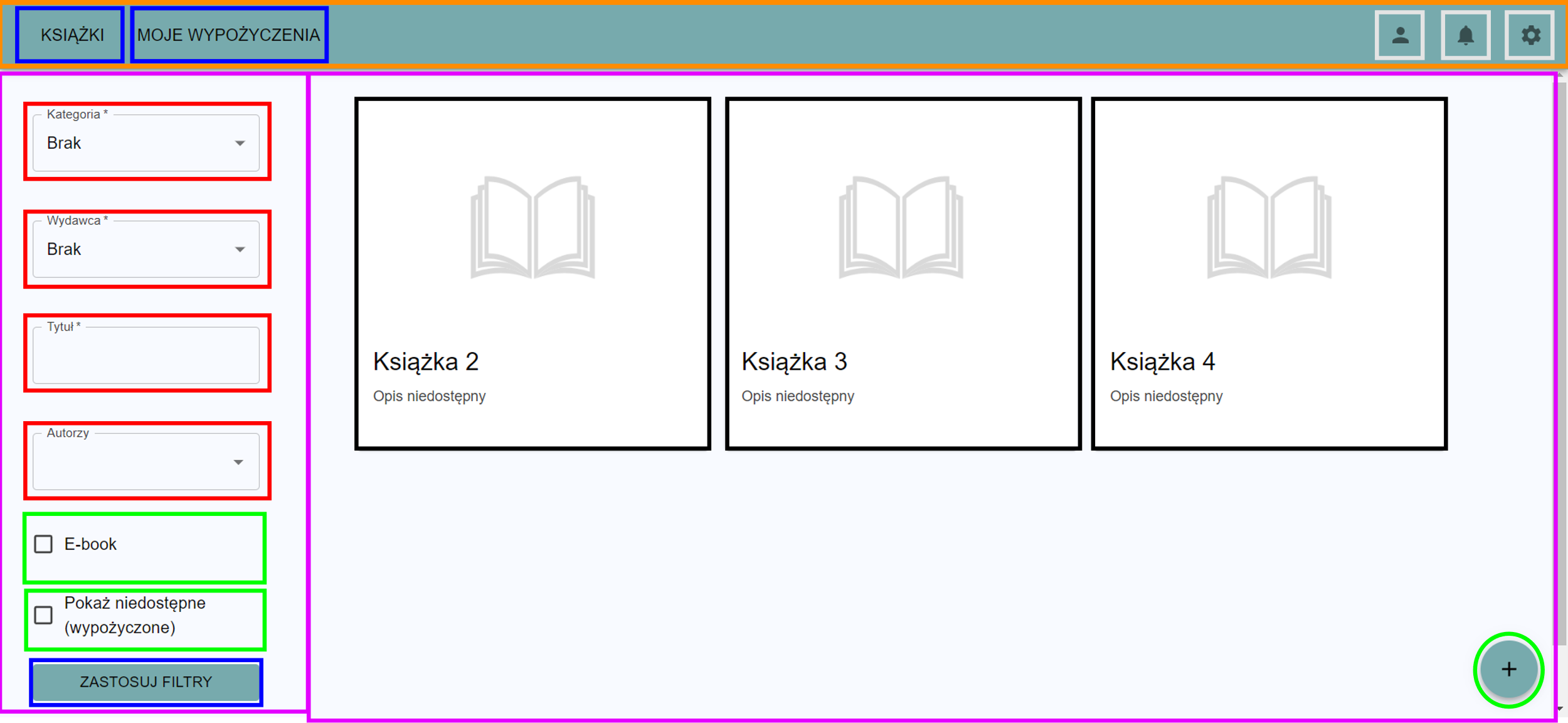
Funkcja ta przyjmuje parametr *e* typu *SelectChangeEvent*. Najpierw pobierany jest kod języka wybrany przez użytkownika za pomocą listy rozwijanej renderowanej przez komponent *LanguageChooser*. Następnie, jeśli wartość ta jest „niepusta”, wywołana zostaje metoda instancji i18 *changeLanguage*, która jako parametr otrzymuje wybrany kod języka. W celu zapamiętania języka i automatycznego ustawienia go podczas kolejnych wizyt kod języka jest także zapisywany w pamięci przeglądarki.

Dodanie nowych języków do aplikacji polegałoby na stworzeniu nowych plików JSON zawierających takie same klucze jak inne pliki z tłumaczeniami. Następnie wystarczy przetłumaczyć ciągi znakowe na języki docelowe. Wymagane byłoby również dodanie do stałej *LANGUAGES* nowego obiektu języka.

### 5.2.5. Interfejs aplikacji

Do budowy interfejsu aplikacji użyto biblioteki komponentów Material UI, która dostarcza rozbudowaną kolekcję gotowych i w pełni funkcjonalnych komponentów React. Pozwoliło to na osiągnięcie spójnego interfejsu o jednakowym stylu, a w razie potrzeby pozwalało na praktycznie pełną personalizację.

Budowanie stron w React polega na tworzeniu drzewa komponentów, które po wyrenderowaniu przez przeglądarkę tworzą interfejs widziany przez użytkownika. Komponenty mogą być w sobie zagnieżdżane. Każda strona to komponent funkcjonalny, który wywołuje szereg innych komponentów. Rysunek 5.5. przedstawia uproszczony podział widoku strony na komponenty. Jeden kolor odpowiada jednemu, oddzielnemu komponentowi. Większość tych komponentów składa się z jeszcze innych, mniejszych komponentów. Jednakże, dla zachowaniu czytelności rysunku, nie zostały one na nim uwzględnione.



Rysunek 5.5. Uproszczony podział widoku strony aplikacji na komponenty React  
*źródło: opracowanie własne*

Budowanie interfejsów za pomocą komponentów przyczynia się do efektywności, czytelności oraz elastyczności kodu. Część komponentów z Rysunku 5.5 jest używana także w innych częściach aplikacji.

Podczas implementowania interfejsu zadbano również o to, aby był on responsywny. Oznacza to, że dostosowuje się on do szerokości urządzenia, na którym jest on wyświetlany. Osiągnięcie tego celu ułatwiło użycie biblioteki Material UI. Komponenty dostarczane przez tę bibliotekę są projektowane z myślą o ekranach mobilnych, a następnie skalowane do większych rozmiarów za pomocą odpowiednich technik używanych w tym celu [46]. Dostarczone przez bibliotekę mechanizmy pozwalają dostosowywać wygląd poszczególnych komponentów w zależności od rozmiaru ekranu. Jednym z komponentów, który był kluczowy podczas projektowania układów stron, jest komponent Grid. Jego cechy upraszczają budowanie reponsywnych układów. Jest on kontenerem, który otacza inne komponenty. Najczęściej, żeby stworzyć układ przypominający ten z Rysunku 5.5 tworzy się wiele zagnieżdżonych w sobie komponentów Grid. Poniżej znajduje się przykładowy kod jednej z wielu instancji tego komponentu użytej podczas budowania interfejsu strony z Rysunku 5.5.

<Grid item xs={12} md={2} container justifyContent={"center"} > (1)

Najważniejsze względem responsywności są trzy parametry, które definiują ile kolumn z dwunastu dostępnych ma zajmować dany komponent, gdy szerokość urządzenia spełnia ograniczenia punktu przerwania (breakpoint). Atrybut *xs* określa ilość zajmowanych kolumn na ekranach od 0 pikseli do 599 pikseli, a atrybut *md* od 900 pikseli do 1199 pikseli. Takie użycie tych atrybutów oznacza, że na ekranach o szerokości między 600 pikseli a 899 pikseli komponent również zajmie 12 kolumn, natomiast na ekranach większych niż 1200 pikseli zawsze będzie zajmował 2 kolumny z 12. Odpowiednie użycie tego komponentu pozwoliło na uzyskanie responsywnego interfejsu. Na Rysunku 5.6 znajduje się interfejs tej samej strony, co na Rysunku 5.5 na ekranie urządzenia Samsung S21 FE, którego rozmiar obszaru przeglądarki, na której widoczna jest strona internetowa (*viewport*) wynosi 360 pikseli na 800 pikseli.

Rysunek 5.6. Interfejs aplikacji na urządzeniu mobilnym  
*źródło: opracowanie własne*

Obraz zawierający zrzut ekranu, tekst, diagram, design

Opis wygenerowany automatycznie

Znaczącą różnicą względem komputerowej wersji interfejsu jest fakt, że formularz definiowania filtrów pojawia się dopiero po rozwinięciu „akordeonu”. Poza tą zmianą wykorzystane są dokładnie te same komponenty co wcześniej.

Stworzenie wygodnego interfejsu dla urządzeń mobilnych było kluczowym aspektem uwzględnionym podczas fazy projektowania. Obecnie coraz więcej osób używa smartfonów czy tabletów do przeglądania różnych stron internetowych. W kwietniu 2024 roku ponad 60% ruchu na stronach internetowych pochodziło z urządzeń mobilnych [47]. Dlatego aplikacje internetowe powinny być gotowe do takiej sytuacji i oferować różne wersje interfejsu dostosowane do wielu rodzajów ekranów.

# 6. Testowanie

Podczas tworzenia oprogramowania nie można zapomnieć o jego testowaniu. Jest to niezwykle istotny etap w całym procesie tworzenia oprogramowania i nie powinien być pomijany.

Testy wykonuje się w celu identyfikacji błędów. Wykrycie jak największej liczby niedociągnięć i ich usunięciu pozwoli na zapewnienie wysokiej jakości tworzonego oprogramowania, a także zwiększenie jego niezawodności. Dzięki testowaniu można także dojrzeć aspekty projektowanej aplikacji, które można zoptymalizować pod kątem wydajności czy bezpieczeństwa. System, który jest stale testowany i naprawiany pozwoli na spełnienie potrzeb jego użytkowników i pozwoli na uniknięcie awarii w środowisku produkcyjnym. Należy jednak pamiętać, iż praktycznie niemożliwym jest znalezienie wszystkich potencjalnych błędów. Każde oprogramowanie, niezależnie jak dokładnie zostało zaprojektowane i zaimplementowane, może zawierać błędy i problemy, które ujawniają się dopiero podczas rzeczywistego użytkowania. Dlatego tak ważne jest, aby systemy poddawane były regularnym, niezależnym testom różnego rodzaju.

Ze względu na brak doświadczenia autora pracy ze zautomatyzowanymi testami, podczas tworzenia systemu testy były wykonywane manualnie. Testowane były pojedyncze metody czy funkcje. Starano się uwzględnić różne scenariusze mogące wystąpić podczas korzystania z aplikacji. W niniejszym rozdziale zostanie opisany sposób testowania aplikacji serwerowej i przeglądarkowej. Proces ten wykonywany był za pomocą specjalistycznych narzędzi.

## 6.1. Testowanie aplikacji serwerowej

Aplikacja serwerowa była testowana przy pomocy wbudowanego debuggera środowiska programistycznego w połączeniu z platformą do testowania interfejsów API Postman.

W celu testowania odpowiednich funkcji i przeanalizowania ich działania używano punktów przerwania debuggera oferowanych przez używane zintegrowane środowisko programistyczne. Punkty przerwania pozwalają na przerwanie wykonywania kodu w wyznaczonym przez programistę miejscu. Dzięki temu zobaczyć można stan zmiennych lub przeanalizować stos wywołań w określonym miejscu w kodzie. Pozwala to na dokładne przeanalizowanie działania aplikacji.

Testy odpowiedzi z punktów końcowych API wykonywane były za pomocą platformy Postman. Odbywało się to poprzez interfejs programu. Pozwala on na wykonywanie zapytań HTTP pod wybrany adres URL. Podczas testów wykonywano zapytania *GET, POST, DELETE* oraz *PUT* i sprawdzano czy odpowiedzi zwracane przez serwer są poprawne. Wykonywano również takie zapytania, które wywoływały różnego rodzaju błędy, by sprawdzić czy aplikacja serwerowa poprawnie je obsługuje. Zwracano uwagę na kody błędów otrzymywanych odpowiedzi. Program pozwala na ustawienie ciała żądania, dzięki czemu można było sprawdzić czy dane z formularzy z aplikacji przeglądarkowej będą poprawnie obsługiwane. Dodatkowo dzięki opcji ustawienia nagłówków żądania HTTP testowane było zabezpieczenia punktów końcowych API. Pozwoliło to na upewnienie się, że bez tokenu dostępu wygenerowanego przez serwer w nagłówku *Authorization,* żądania nie zostaną obsłużone i zostanie zwrócona odpowiedź z odpowiednim statusem HTTP.

Testy wykonywane za pomocą debuggera i aplikacji Postman jednocześnie pozwoliły na dokładne sprawdzenie działania aplikacji w różnych scenariuszach. Najpierw, w używanym zintegrowanym środowisku programistycznym, ustawiano punkt przerwania w interesującym autora miejscu w kodzie. Następnie za pomocą interfejsu aplikacji Postman definiowano odpowiednie zapytanie HTTP do danego punktu końcowego API. Jeśli zapytanie zostało poprawnie zdefiniowane, zintegrowane środowisko programistyczne zatrzymywało wykonanie kodu na wybranych przez autora linijkach kodu. Przy pomocy interfejsu IDE analizowano kod linijka po linijce w poszukiwaniu błędów czy niezgodności, a także możliwych optymalizacji kodu. W przypadku ich zlokalizowania, dokonywane były zmiany w kodzie lub w wysyłanych żądaniach HTTP.

## 6.2. Testowanie aplikacji przeglądarkowej

Aplikacja React była testowana przy pomocy narzędzia wbudowanego w przeglądarkę internetową Google Chrome, czyli Chrome Dev Tools. Pomocna okazała się też sama przeglądarka, za pomocą której przetestowano ogólne korzystanie z aplikacji.

Dzięki poziomowi zaawansowania i rozbudowania, narzędzie Chrome Dev Tools pozwoliło na przetestowanie różnych aspektów aplikacji internetowej React. Dzięki konsoli można było wyświetlić wartości zmiennych czy stałych, wyniki funkcji, dane pobrane z serwera lub inne przydatne informacje za pomocą instrukcji umieszczonych w kodzie. Pozwoliło to na dokładne przeanalizowanie kodu i naprawienie powstających podczas pracy błędów kompilacji.

Wbudowany w Chrome Dev Tools eksplorator struktury HTML i stylów CSS umożliwił dokładne zapoznanie się ze strukturą zbudowaną za pomocą komponentów React. Edytor stylów CSS pozwolił na dynamiczne usuwanie i dodawanie właściwości CSS z różnych znaczników HTML. Dzięki temu można było obserwować, jak te działania wpływają na wygląd wyświetlanej strony. W rezultacie możliwe było szybkie dostosowanie wyglądu strony bezpośrednio z poziomu przeglądarki. Po znalezieniu odpowiedniej kombinacji ustawionych właściwości, dokonywano korekty w kodzie.

Za pomocą narzędzia pozwalającego na wyświetlenie mobilnej wersji strony zbadano i przetestowano responsywność aplikacji. Narzędzie umożliwia wybór określonych rozmiarów ekranów lub pozwala na dynamiczną zmianę rozmiaru obszaru wyświetlającego stronę internetową. Umożliwiło to sprawdzenie, przy jakim rozmiarze ekranu aplikacja zmienia interfejs. Możliwe było również ocenienie, czy punkty przerwania są dobrze zdefiniowane. Dodatkowo sprawdzono, czy interfejs na różnych ekranach zachowuje się w odpowiednio i przewidywalnie. Ważne było upewnienie się, że interfejs jest czytelny i intuicyjny zarówno na mniejszych, jak i średnich ekranach.

Poza narzędziem Chrome Dev Tools, które do używania wymaga specjalistycznej wiedzy, do testowania tej części aplikacji użyta została również sama przeglądarka Google Chrome. Z pomocą tego oprogramowania zbadane zostało ogólne wrażenie podczas korzystania z aplikacji internetowej. Przeanalizowano również, czy interfejs zmienia się w zależności od roli użytkownika po zalogowaniu. W niektórych miejscach aplikacji użytkownik z rolą administratora powinien posiadać dodatkowe lub całkowicie inne elementy interfejsu. Dla użytkownika z rolą inną niż administrator te części interfejsu nie powinny być widoczne. Przeglądarka umożliwiła dokładne sprawdzenie, czy ten aspekt działa zgodnie z oczekiwaniami, oraz wprowadzenie ewentualnych korekt w kodzie.

# 7. Podsumowanie

Biblioteki i ich organizacja pracy przeszły bardzo długą drogę. Od czasów, gdy wszystkie czynności biblioteczne wykonywane były ręcznie i niezależnie od siebie, stale wprowadzano wiele udoskonaleń. Nieopisany okazał się postęp technologiczny i rozwiązania informatyczne w tym zakresie. Komputeryzacja bibliotek rozwiązała wiele problemów dotyczących organizacji czy wydajności takich placówek. Nowoczesne biblioteki wychodzą poza ramy tradycyjnej placówki wypożyczającej książki czy inne materiały biblioteczne i oferują szereg nowych usług, bez których ciężko dzisiaj wyobrazić sobie ich działanie.

Celem pracy był projekt i implementacja systemu do obsługi czytelni elektronicznej. Cel pracy został zrealizowany pomyślnie.

Wszystkie wymagania funkcjonalne i niefunkcjonalne zdefiniowane przed fazą implementacji zostały zrealizowane i przetestowane. Dokładne zaprojektowanie aplikacji i bazy danych oraz przeanalizowanie czynników mających wpływ na stworzenie dobrego oprogramowania było kluczowe dla osiągnięcia tych założeń. Wybrane technologie okazały się być odpowiednie do postawionego zadania i pozwoliły na zaimplementowanie wszystkich kluczowych funkcjonalności.

Istniejące systemy biblioteczne, poza opisanymi w niniejszej pracy funkcjonalnościami, oferują wiele innych rozwiązań służących do zarządzania biblioteką. Należy jednak pamiętać, iż systemy te były tworzone, testowane i udoskonalane przez zespoły programistów przez lata. W tworzonej przez autora aplikacji zaimplementowano te funkcje, które zostały uznane za najistotniejsze. Jest to minimalny zakres, który powinna spełniać aplikacja służąca do automatyzacji procesów bibliotecznych.

W fazie rozwoju systemu należałoby wprowadzić funkcje dodatkowo usprawniające jego użytkowanie. Przykładem takiej funkcji mogłoby być skanowanie kodów kreskowych na materiałach bibliotecznych, co umożliwiłoby szybkie wyszukiwanie powiązanych wypożyczeń lub automatyczne uzupełnianie informacji o danym materiale. Dane mogłyby być pobierane z zewnętrznego API lub od dostawcy danych bibliograficznych. Taka funkcjonalność pozwoliłaby jeszcze bardziej zredukować czas potrzebny na rejestracje zwrotów i wypożyczeń, ułatwiłaby śledzenie materiałów i dodawanie nowych pozycji.

Kolejną funkcjonalnością, która uczyniłaby użytkowanie aplikacji przyjemniejszym i praktyczniejszym, byłaby możliwość dodawania zdjęć okładek za pomocą aparatu telefonu komórkowego. System mógłby wykrywać, że aplikacja internetowa jest używana na takim urządzeniu, i wówczas wyświetlić odpowiedni element interfejsu i aktywować tę opcję. Przed pierwszym użyciem aplikacja musiałby uzyskać uprawnienie do użycia aparatu urządzenia.

Inną przydatną dla czytelników funkcją mogłoby okazać się wprowadzenie systemu ocen materiałów bibliotecznych i jego integracja z popularnymi serwisami związanymi z czytelnictwem. Użytkownikom pojawiałaby się ocena danego materiału bibliotecznego z różnych serwisów oraz liczba ocen. Mogłoby to pomóc z podjęciem decyzji czy dany materiał jest godny uwagi. Dodatkowo istniałaby opcja rozwinięcia tej funkcjonalności o możliwość oceniania i recenzowania materiałów. Następnie recenzje te byłyby możliwe do przeglądania podczas wyświetlania szczegółów materiału bibliotecznego. Taka funkcjonalność wzbogaciłaby zasoby systemu i biblioteki o cenne informacje zwrotne.

Możliwości rozwoju aplikacji jest bardzo wiele i są one praktycznie nieograniczone. Mając to na uwadze, w aplikacji zawarto najważniejsze i najbardziej kluczowe funkcjonalności. Wszystkie założenia znajdujące się w zakresie niniejszej pracy zostały spełnione. Cel pracy uznaje się za osiągnięty.

# 8. Bibliografia

[1] Główny Urząd Statystyczny, *Biblioteki publiczne w 2022 r*., Warszawa, 2023.

[2] Radwański A., *Jak komputeryzować bibliotekę*. Warszawa: Wydawnictwo SBP, 2000.

[3] Główny Urząd Statystyczny, *Kultura i dziedzictwo narodowe w 2022 r.*, Warszawa, Kraków, 2023.

[4] Lucidea, *The Integrated Library System (ILS) Primer*, https://lucidea.com/special-libraries/the-integrated-library-system-ils-primer/, [07.05.2024].

[5] Radwański A., *System biblioteczny – czym jest i z czym się wiąże*, Biuletyn EBIB, 2022, nr 5 (206)/2022, https://ebibojs.pl/index.php/ebib/article/download/823/853/, [02.06.2024].

[6] Winogrodzka K., *Raport o stanie komputeryzacji bibliotek publicznych w roku 2012*. Bibliotekarz 2013, nr 2, s. 4–12, https://pliki.sbp.pl/ac/Bibliotekarz\_2013\_02.pdf, [05.05.2024]

[7] Akademia Nauk Stosowanych w Elblągu, *O bibliotece*, https://ans-elblag.pl/o-bibliotece/, [07.05.2024].

[8] Biblioteka Narodowa, *FAQ*, https://www.bn.org.pl/projekty/omnis/faq, [07.05.2024].

[9] Biblioteka Narodowa, *Wdrożenia*, https://www.bn.org.pl/dla-bibliotekarzy/narodowy-program-rozwoju-czytelnictwa-2.0/budowa-ogolnokrajowej-sieci-bibliotecznej-poprzez-zintegrowany-system-zarzadzania-zasobami-bibliotek/wdrozenia, [07.05.2024].

[10] Żyłowski D., *Biblioteka wdraża nowy system wypożyczania książek*, https://bibliotekaelblaska.pl/biblioteka-wdraza-nowy-system-wypozyczania-ksiazek/, [07.05.2024].

[11] Instytut Książki, *Informacje o MAK+*, http://www.makplus.pl/index.php/informacje-o-systemie/informacje-o-mak, [07.05.2024].

[12] Visure Solutions*, Czym są wymagania funkcjonalne: przykłady, definicja, kompletny przewodnik*, https://visuresolutions.com/pl/blog/functional-requirements/, [09.05.2024].

[13] Visure Solutions, *Wymagania niefunkcjonalne: typy, przykłady i podejścia*, https://visuresolutions.com/pl/requirements-management-traceability-guide/non-functional-requirements/, [09.05.2024].

[14] thestory, *Przypadki Użycia*, https://thestory.is/pl/proces/faza-rozwoju-koncepcji/przypadki-uzycia/, [23.05.2024].

[15] Rutkowska M., *Diagram przypadków użycia: wprowadzenie*, https://www.altkomsoftware.com/pl/blog/diagram-przypadkow-uzycia-wprowadzenie/, [23.05.2024].

[16] Czarnota D., *Jakie czynniki wpływają na wybór technologii do projektu IT?*, https://impicode.pl/blog/jakie-czynniki-wplywaja-na-wybor-technologii-do-projektu-it/, [14.05.2024].

[17] JetBrains, *Features overview*, https://www.jetbrains.com/idea/features/, [14.05.2024].

[18] Mendys M., *Co to jest TypeScript i dlaczego sprawdzi się w Twoich projektach?,* https://www.droptica.pl/blog/co-jest-typescript-i-dlaczego-sprawdzi-sie-w-twoich-projektach/, [14.05.2024].

[19] React, *Components and Props*, https://legacy.reactjs.org/docs/components-and-props.html, [14.05.2024].

[20] UXPin, *How Does React Work?*, https://www.uxpin.com/studio/blog/how-react-works/, [14.05.2024].

[21] Material UI, *Material UI – Overview*, https://mui.com/material-ui/getting-started/, [12.05.2024].

[22] Statista, *Most used programming languages among developers worldwide as of 2023*, https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/, [24.05.2024].

[23] MariaDB, *About MariaDB Software*, https://mariadb.com/kb/en/about-mariadb-software/, [24.05.2024].

[24] O’Reilly, *Software Architecture Patterns*, https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html, [20.05.2024].

[25] Dąbrowski M., *Spring Data JPA – wprowadzenie*, https://nullpointerexception.pl/spring-data-jpa-wprowadzenie/, [15.05.2024].

[26] Spring, *Defining Repository Interfaces*, https://docs.spring.io/spring-data/jpa/reference/repositories/definition.html, [15.05.2024].

[27] Spring, *Defining Query Methods*, https://docs.spring.io/spring-data/jpa/reference/repositories/query-methods-details.html, [15.05.2024].

[28] Spring, *Annotation Interface Repository*, https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Repository.html, [16.05.2024].

[29] Spring, *JPA Query Methods*, https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html#jpa.query-methods.at-query, [19.05.2024].

[30] Spring, *Annotation Interface RestController*, https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html, [16.05.2024].

[31] Spring, *Building a RESTful Web Service*, https://spring.io/guides/gs/rest-service, [16.05.2024].

[32] Spring, *Mapping Requests*, https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html, [16.05.2024].

[33] Project Lombok, *@NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor*, https://projectlombok.org/features/constructor, [16.05.2024].

[34] Spring, *Annotation Interface PathVariable*, https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PathVariable.html, [17.05.2024].

[35] Oracle, *Annotation Type Entity*, https://docs.oracle.com/javaee%2F7%2Fapi%2F%2F/javax/persistence/Entity.html, [17.05.2024].

[36] Oracle, *Annotation Type Id*, https://docs.oracle.com/javaee%2F6%2Fapi%2F%2F/javax/persistence/Id.html, [17.05.2024].

[37] Oracle, *Annotation Type GeneratedValue*, https://docs.oracle.com/javaee%2F6%2Fapi%2F%2F/javax/persistence/GeneratedValue.html [17.05.2024].

[38] Oracle, *Annotation Type ManyToOne*, https://docs.oracle.com/javaee%2F6%2Fapi%2F%2F/javax/persistence/ManyToOne.html, [17.05.2024].

[39] Oracle, *Annotation Type JoinColumn*, https://docs.oracle.com/javaee%2F6%2Fapi%2F%2F/javax/persistence/JoinColumn.html, [17.05.2024].

[40] Hibernate, *Annotation Type CreationTimestamp*, https://docs.jboss.org/hibernate/orm/6.2/javadocs/org/hibernate/annotations/CreationTimestamp.html, [17.05.2024].

[41] Oracle, *Annotation Type Enumerated*, https://docs.oracle.com/javaee%2F6%2Fapi%2F%2F/javax/persistence/Enumerated.html, [17.05.2024].

[42] Spring, *Spring Security Overview*, https://docs.spring.io/spring-security/reference/index.html, [17.05.2024].

[43] Spring, *Class OncePerRequestFilter*, https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html, [18.05.2024].

[44] Jakarta, *Class Cookie*, https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/http/cookie#setSecure-boolean-, [18.05.2024].

[45] Axios, *Interceptors*, https://axios-http.com/docs/interceptors, [19.05.2024].

[46] Material UI, *Usage*, https://mui.com/material-ui/getting-started/usage/, [19.05.2024].

[47] Howarth J., *Internet Traffic from Mobile Devices*, https://explodingtopics.com/blog/mobile-internet-traffic, [20.05.2024].

# 9. Spis rysunków

[Rysunek 1.1. Biblioteki publiczne w Polsce oraz liczba czytelników na 1 bibliotekę 7](#_Toc168316950)

[Rysunek 3.1. Diagram przypadków użycia projektowanej aplikacji 19](#_Toc168316951)

[Rysunek 3.2. Schemat bazy danych 23](#_Toc168316952)

[Rysunek 5.1. Struktura katalogów aplikacji serwerowej 31](#_Toc168316953)

[Rysunek 5.2. Schemat architektury warstwowej aplikacji 32](#_Toc168316954)

[Rysunek 5.3. Schemat mapowania klasy Java na tabelę bazy danych przez Hibernate 38](#_Toc168316955)

[Rysunek 5.4. Struktura katalogów w aplikacji React 43](#_Toc168316956)

[Rysunek 5.5. Uproszczony podział widoku strony aplikacji na komponenty React 47](#_Toc168316957)

[Rysunek 5.6. Interfejs aplikacji na urządzeniu mobilnym 48](#_Toc168316958)

# 10. Spis tabel

[Tabela 2.1. Zestawienie funkcjonalności oferowanych czytelnikowi przez wybrane systemy biblioteczne 14](#_Toc168317424)

[Tabela 3.1. Minimalne i zalecane wymagania komputera dla instalacji projektowanej aplikacji 20](#_Toc168317425)