WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

POLITECHNIKA
RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Programowanie aplikacji webowych

Dokumentacja projektu

Biblioteka

Paweł Łopocki

173660

3-EF-DI

Rzeszów 2025

Spis treści

1. Wstęp

- **Opis projektu**: Aplikacja webowa dla biblioteki, która umożliwia użytkownikom przeglądanie dostępnych książek i wypożyczanie ich online. Po dokonaniu wypożyczenia użytkownik otrzymuje informację, że książka jest gotowa do odbioru w bibliotece.
- **Cele aplikacji**:
    - Umożliwienie użytkownikom łatwego przeglądania książek.
    - Wyszukiwanie książek po tytule, autorze, gatunku itp.
    - Wypożyczanie książek online, z informacją o konieczności osobistego odbioru.
    - Zarządzanie dostępnością książek w bibliotece.
- **Technologie**:
    - **Backend**: Spring Boot (Java) – dla logiki aplikacji i komunikacji z bazą danych.
    - **Frontend**: React – do tworzenia interaktywnego interfejsu użytkownika.
    - **Baza danych**: MySQL do przechowywania informacji o książkach, użytkownikach, wypożyczeniach.

2. Wymagania

- **Wymagania systemowe**:
    - Java 17 (lub nowsza). Projekt wykonyany na wersji 23.0.1
    - Node.js(v18.20.5) oraz npm(10.8.2).
    - Maven(Apache Maven 3.9.9)

3. Instalacja

- Backend

```
>> git clone https://github.com/lewapFT9/Biblioteka_p_a_w.git
```

Następnie w folderze z projektem:

```
>> ./mvnw clean install
```

```
>> ./mvnw spring-boot:run
```

Backend uruchamia się na porcie 8080

- Frontend

```
>> git clone https://github.com/lewapFT9/Bibiloteka_front_PAW.git
```

Następnie w folderze z projektem:

```
>> npm install
```

```
>> npm start
```

Frontend uruchamia się na porcie 3000

4. Opis

Backend

Modele

1. Authors.java

- **Opis**: Klasa reprezentująca autora książek w bibliotece.
- **Funkcjonalność**: Klasa przechowuje dane autora, takie jak imię (name) i nazwisko (surname), oraz związane z nimi książki (booksS).

2. Books.java

- **Opis**: Klasa reprezentująca książkę w bibliotece.
- **Funkcjonalność**: Klasa przechowuje dane książki, takie jak tytuł (title), rok wydania (releaseYear), autor (idAuthor), kategoria (idCategory), liczba egzemplarzy (numberOfCopies), oraz wypożyczenia (rentalsS).

3. Categories.java

- **Opis**: Klasa reprezentująca kategorię książek w bibliotece.
- **Funkcjonalność**: Klasa przechowuje dane kategorii książek (name) oraz listę książek w danej kategorii (booksS).

4. DaneUzytkownika.java

- **Opis**: Klasa implementująca interfejs UserDetails z pakietu Spring Security, która reprezentuje użytkownika aplikacji.

- **Funkcjonalność**: Klasa ta jest wykorzystywana do zarządzania sesją użytkownika, autentykacją i autoryzacją w aplikacji. Zawiera dane użytkownika takie jak hasło i email.
- **Implementacja UserDetails**: Metody w tej klasie implementują wymagane metody interfejsu UserDetails, umożliwiając Spring Security zarządzanie sesjami użytkowników.
    - getUsername(): Zwraca email użytkownika.
    - getPassword(): Zwraca hasło użytkownika.
    - getAuthorities(): Zwraca role użytkownika (np. USER, ADMIN).

5. Rents.java

- **Opis**: Klasa reprezentująca wypożyczenie książki przez użytkownika w bibliotece.
- **Funkcjonalność**: Klasa przechowuje dane dotyczące wypożyczenia książki, takie jak data wypożyczenia (rentDate), data zwrotu (returnDate), użytkownik wypożyczający książkę (idUser), oraz książka (idBook).

6. Users.java

- **Opis**: Klasa reprezentująca użytkownika aplikacji bibliotecznej.
- **Funkcjonalność**: Klasa przechowuje dane użytkownika, takie jak imię (name), nazwisko (surname), numer telefonu (phoneNumber), email (email), rola (role), hasło (password), oraz wypożyczenia użytkownika (rentalsS).

Modele JPA są częścią systemu zarządzania biblioteką, który umożliwia przechowywanie i manipulowanie danymi o książkach, autorach, kategoriach, użytkownikach i wypożyczeniach. Wszystkie encje są odpowiednio połączone za pomocą relacji "jeden do wielu" lub "wiele do jednego", co pozwala na skuteczne zarządzanie danymi w bazie.

DTO

1. BookDTO.java

- **Opis**: Klasa DTO reprezentująca książkę, wykorzystywana do transferu danych o książkach z backendu do frontendowej części aplikacji (lub odwrotnie).
- **Pola**:
    - id: ID książki w systemie.
    - title: Tytuł książki.
    - author: Autor książki.
    - category: Kategoria, do której należy książka.
    - yearOfRelease: Rok wydania książki.
    - numberOfCopies: Liczba dostępnych egzemplarzy książki.

- **Funkcjonalność**: Klasa ta służy do transferu podstawowych informacji o książce (np. do wyświetlenia na stronie internetowej biblioteki). Zawiera konstruktor umożliwiający tworzenie obiektów DTO z danymi książki.

2. BookDTO2.java

- **Opis**: Kolejna klasa DTO, która służy do transferu danych o książkach, ale z dodatkowymi szczegółami dotyczącymi autora książki (imienia i nazwiska).
- **Pola**:
    - title: Tytuł książki.
    - category: Kategoria książki.
    - yearOfRelease: Rok wydania książki.
    - numberOfCopies: Liczba dostępnych egzemplarzy książki.
    - authorName: Imię autora książki.
    - authorSurname: Nazwisko autora książki.
- **Funkcjonalność**: Klasa ta jest bardziej szczegółowa niż BookDTO i zawiera informacje o autorze książki. Może być używana w przypadkach, gdzie oprócz podstawowych danych o książce, chcemy również przekazać dane autora.

3. RentDTO.java

- **Opis**: Klasa DTO reprezentująca wypożyczenie książki. Służy do przenoszenia danych związanych z wypożyczeniem książki, takich jak tytuł książki oraz daty wypożyczenia i zwrotu.
- **Pola**:
    - id: ID wypożyczenia.
    - title: Tytuł książki, którą wypożyczono.
    - rentDate: Data wypożyczenia książki.
    - returnDate: Data zwrotu książki.
- **Funkcjonalność**: Klasa ta służy do przenoszenia danych wypożyczenia książki, które mogą być wyświetlane użytkownikowi na stronie lub używane do zarządzania wypożyczeniami w aplikacji.

Klasy DTO umożliwiają efektywne przesyłanie tylko wymaganych danych pomiędzy różnymi warstwami aplikacji, zapewniając lepszą wydajność oraz lepszą organizację kodu, gdyż nie przesyłamy całych encji

Klasy konfiguracyjne

1. SecConfig.java

- **Opis**: Klasa konfiguracyjna dla Spring Security, która ustawia reguły bezpieczeństwa aplikacji. Definiuje, jak aplikacja powinna zarządzać sesjami, jak użytkownicy mogą się logować, oraz jak weryfikowane są ich dane uwierzytelniające.
- **Adnotacje**:
  - @Configuration: Oznacza, że klasa ta zawiera definicje beanów dla kontekstu aplikacji.
  - @EnableWebSecurity: Włącza konfigurację zabezpieczeń webowych Spring Security.
  - @EnableMethodSecurity: Włącza zabezpieczenia na poziomie metod (np. @PreAuthorize).
- **Kluczowe komponenty**:
  - **SecurityFilterChain securityFilterChain(HttpSecurity http)**:
    - Konfiguruje filtr zabezpieczeń HTTP.
    - Ustawia reguły autoryzacji (np. /login, /register są dostępne publicznie, a wszystkie inne ścieżki wymagają uwierzytelnienia).
    - Wyłącza CSRF (Cross-Site Request Forgery).
    - Ustawia politykę sesji jako **stateless** (wymusza brak sesji po stronie serwera, co jest typowe dla aplikacji REST).
    - Dodaje filtr JWT (jwtFilter) do łańcucha filtrów HTTP.
    - Konfiguruje CORS (Cross-Origin Resource Sharing), pozwalając na żądania z http://localhost:3000 (frontend w React).
  - **AuthenticationManager authenticationManager(AuthenticationConfiguration config)**:
    - Zwraca AuthenticationManager, który jest odpowiedzialny za uwierzytelnianie użytkowników.
  - **UserDetailsService userDetailsService()**:
    - Wstrzykuje i konfiguruje niestandardową implementację UserDetailsService, która ładuje użytkowników.
  - **AuthenticationProvider authenticationProvider()**:
    - Konfiguruje DaoAuthenticationProvider, który wykorzystuje UserDetailsService oraz PasswordEncoder do weryfikacji haseł.

- **PasswordEncoder passwordEncoder()**:
  - Ustawia BCryptPasswordEncoder do szyfrowania i porównywania haseł.

- **corsConfigurationSource()**:
  - Definiuje zasady CORS (dopuszczając metody HTTP GET, POST, PUT, DELETE oraz umożliwiając wysyłanie żądań z frontendów działających na porcie 3000).

2. JwtFilter.java

- **Opis**: Klasa filtra, który działa w łańcuchu filtrów Spring Security i służy do weryfikacji tokenów JWT dla każdego przychodzącego żądania. Jest to kluczowy element obsługi autentykacji za pomocą tokenów JWT.
- **Adnotacje**:
  - @Component: Dzięki tej adnotacji klasa jest automatycznie zarządzana przez Spring i może być wstrzykiwana do innych komponentów (np. do SecConfig).
- **Metody**:
  - **doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)**:
    - Sprawdza nagłówek Authorization w przychodzących żądaniach HTTP. Jeśli nagłówek zawiera token JWT (rozpoczynający się od Bearer), to filtr wyodrębnia token oraz nazwę użytkownika.
    - Jeśli token jest ważny i użytkownik nie jest jeszcze uwierzytelniony (brak obiektu Authentication w kontekście bezpieczeństwa), filtr weryfikuje token, ładując szczegóły użytkownika z CustomUserDetailsService.
    - Jeśli token jest poprawny, tworzy obiekt UsernamePasswordAuthenticationToken i ustawia go w kontekście bezpieczeństwa Spring Security, co pozwala na dostęp do zasobów wymagających uwierzytelnienia.
    - Następnie przekazuje żądanie do kolejnych filtrów w łańcuchu (filterChain.doFilter).

SecConfig.java: Odpowiada za ogólną konfigurację zabezpieczeń aplikacji, w tym zarządzanie sesjami, regułami dostępu oraz integrację z JWT.

JwtFilter.java: Filtr, który przetwarza nagłówki JWT w przychodzących żądaniach, weryfikuje tokeny i ustawia odpowiednią autentykację w kontekście Spring Security.

Obie klasy razem pozwalają na bezpieczną autentykację użytkowników za pomocą JWT, a także odpowiednie zarządzanie dostępem do zasobów aplikacji.

Repozytoria

1. AuthorRepo.java

- **Opis**: Repozytorium dla encji Authors, które zarządza danymi dotyczącymi autorów książek w systemie.
- **Metody**:
    - **findByFirstName(String firstName)**:
        - Zapytanie wyszukuje autorów na podstawie ich imienia lub nazwiska (ignoruje wielkość liter). Zwraca listę identyfikatorów autorów, którzy pasują do podanego imienia lub nazwiska.
        - Używa zapytania SQL (nativeQuery = true), w którym korzysta z operatora LIKE do dopasowywania.
    - **findByNameAndSurname(String name, String surname)**:
        - Wyszukuje autora na podstawie dokładnych wartości imienia i nazwiska, ignorując wielkość liter.
        - Zwraca opcjonalny obiekt Authors, jeśli autor o podanych danych istnieje.

2. BooksRepo.java

- **Opis**: Repozytorium dla encji Books, które zarządza książkami w bibliotece.
- **Metody**:
    - **findAllByCategory(int category)**:
        - Zapytanie wyszukuje wszystkie książki, które należą do określonej kategorii. Zwraca listę książek.
    - **findByAuthor(int author_id)**:
        - Wyszukuje wszystkie książki, które zostały napisane przez autora o określonym author_id. Zwraca listę książek.

3. CategoryRepo.java

- **Opis**: Repozytorium dla encji Categories, które zarządza kategoriami książek w bibliotece.
- **Metody**:
    - **findByName(String name)**:
        - Wyszukuje kategorię książek po nazwie (ignoruje wielkość liter). Zwraca obiekt Categories jako opcjonalny wynik, jeśli kategoria o podanej nazwie istnieje.

4. RentalsRepo.java

- **Opis**: Repozytorium dla encji Rents, które zarządza danymi o wypożyczeniach książek przez użytkowników.
- **Metody**:
    - **findByUserId(int id)**:
        - Wyszukuje wszystkie wypożyczenia dla użytkownika o określonym user_id. Zwraca listę obiektów Rents, które zawierają informacje o książkach wypożyczonych przez danego użytkownika.

5. UserRepo.java

- **Opis**: Repozytorium dla encji Users, które zarządza danymi użytkowników aplikacji.
- **Metody**:
    - **findByEmail(String email)**:
        - Wyszukuje użytkownika na podstawie jego adresu e-mail. Zwraca obiekt Users, jeśli użytkownik o danym adresie e-mail istnieje w bazie danych.

Wszystkie repozytoria są oparte na Spring Data JPA i rozszerzają interfejs JpaRepository, co zapewnia im standardowe operacje CRUD (Create, Read, Update, Delete) na bazie danych. Każde repozytorium zawiera specjalne zapytania, które pozwalają na bardziej zaawansowane operacje wyszukiwania

Serwisy

1. AuthorService.java

- **Opis**: Klasa odpowiedzialna za operacje na autorach książek.
- **Metody**:
    - **findAuthorByName(String name)**:
        - Wyszukuje autorów na podstawie częściowego imienia lub nazwiska (używając LIKE), a wynik jest zwracany w postaci listy identyfikatorów autorów.
    - **findAuthorToAddBook(String firstName, String lastName)**:
        - Wyszukuje autora na podstawie imienia i nazwiska. Jeśli autor nie istnieje, zwraca odpowiedź NOT_FOUND.
    - **addAuthor(String firstName, String lastName)**:
        - Dodaje nowego autora do bazy danych, jeśli taki autor jeszcze nie istnieje. Jeśli już istnieje, zwraca odpowiedź CONFLICT.
    - **getAuthorById(int id)**:
        - Pobiera autora na podstawie jego identyfikatora (id). Jeśli nie ma takiego autora, zwraca odpowiedź NOT_FOUND.
    - **returnAuthorToAddBook(String firstName, String lastName)**:
        - Sprawdza, czy autor o podanych imieniu i nazwisku już istnieje. Jeśli tak, zwraca autora, w przeciwnym razie tworzy nowego autora.
    - **deleteAuthorById(int id)**:
        - Usuwa autora z bazy danych na podstawie identyfikatora. Jeśli nie znajdzie autora, zwraca odpowiedź NOT_FOUND.
    - **updateAuthor(int id, Authors updateData)**:
        - Aktualizuje dane autora (np. imię, nazwisko) na podstawie identyfikatora. Jeśli autor nie istnieje, zwraca odpowiedź NOT_FOUND.

2. BooksService.java

- **Opis**: Klasa odpowiedzialna za operacje na książkach.
- **Metody**:
    - **getBooks()**:
        - Pobiera wszystkie książki z bazy danych i zwraca je w odpowiedzi.

- **getBookToDisplay(int id)**:
    - Pobiera książkę na podstawie identyfikatora i mapuje dane książki na DTO (BookDTO) do wyświetlenia. Jeśli książka nie istnieje, zwraca odpowiedź NOT_FOUND.
- **getBookById(int id)**:
    - Pobiera książkę na podstawie identyfikatora. Jeśli książka nie istnieje, zwraca odpowiedź NOT_FOUND.
- **getBooksByAuthor(String author)**:
    - Wyszukuje książki autora na podstawie imienia (używając repozytorium AuthorRepo). Jeśli autor istnieje, zwraca listę książek tego autora.
- **findTitleById(int id)**:
    - Wyszukuje tytuł książki na podstawie jej identyfikatora. Jeśli książka nie istnieje, zwraca status NOT_FOUND.
- **addBook(@Valid BookDTO2 dto)**:
    - Dodaje nową książkę do bazy danych, korzystając z przekazanego DTO (BookDTO2). Tworzy nowy obiekt książki, przypisuje autora i kategorię, a następnie zapisuje książkę.
- **deleteBook(int id)**:
    - Usuwa książkę na podstawie identyfikatora. Jeśli książka nie istnieje, zwraca odpowiedź NOT_FOUND.
- **updateBook(int id, BookDTO2 updateData)**:
    - Aktualizuje dane książki (np. tytuł, autor, kategoria, liczba kopii) na podstawie identyfikatora. Jeśli książka nie istnieje, zwraca odpowiedź NOT_FOUND.

3. CategoryService.java

- **Opis**: Klasa odpowiedzialna za operacje na kategoriach książek.
- **Metody**:
    - **getAllCategories()**:
        - Pobiera wszystkie kategorie z bazy danych i zwraca je w odpowiedzi.
    - **getBooksByCategory(int id)**:
        - Wyszukuje książki przypisane do danej kategorii na podstawie jej identyfikatora. Zwraca je w postaci listy obiektów DTO (BookDTO).

- **returnCategoryToAddBook(String categoryName)**:
  - Sprawdza, czy kategoria o podanej nazwie istnieje. Jeśli tak, zwraca kategorię, w przeciwnym razie tworzy nową kategorię.
- **addCategory(String categoryName)**:
  - Dodaje nową kategorię do bazy danych, jeśli kategoria o danej nazwie jeszcze nie istnieje. Jeśli już istnieje, zwraca odpowiedź CONFLICT.
- **deleteCategory(int id)**:
  - Usuwa kategorię na podstawie identyfikatora. Jeśli kategoria nie istnieje, zwraca odpowiedź NOT_FOUND.
- **updateCategory(int id, String categoryName)**:
  - Aktualizuje nazwę kategorii na podstawie identyfikatora. Jeśli kategoria nie istnieje, zwraca odpowiedź NOT_FOUND.
- **findById(int id)**:
  - Pobiera kategorię na podstawie identyfikatora. Jeśli kategoria nie istnieje, zwraca odpowiedź NOT_FOUND.

4. CustomUserDetailsService.java

- **Opis**: Klasa służąca do implementacji interfejsu UserDetailsService, który jest używany przez Spring Security do ładowania danych użytkownika na podstawie adresu e-mail.
- **Metody**:
  - **loadUserByUsername(String email)**:
    - Wyszukuje użytkownika na podstawie podanego adresu e-mail w repozytorium UserRepo. Jeśli użytkownik nie zostanie znaleziony, rzuca wyjątek UsernameNotFoundException. Zwraca obiekt DaneUzytkownika, który implementuje interfejs UserDetails i jest używany przez Spring Security do uwierzytelniania.

5. JWTService

- **Opis:** Ten serwis odpowiada za tworzenie, walidację i parsowanie tokenów JWT (JSON Web Token). JWT jest używane do zabezpieczania API i zapewniania, że użytkownik wykonujący zapytanie jest uwierzytelniony.
- **Metody**

- **Konstruktor**:
  - Generuje sekretne hasło za pomocą algorytmu HmacSHA256.
- **generateToken(String email)**:
  - Tworzy token JWT dla podanego adresu e-mail, z dodanymi roszczeniami oraz czasem wygaśnięcia.
- **getKey()**:
  - Zwraca sekretne hasło, które jest używane do podpisywania tokenu JWT.
- **extractUserName(String jwtToken)**:
  - Wyciąga nazwę użytkownika (adres e-mail) z tokenu JWT.
- **extractClaim(String jwtToken, Function<Claims, T> claimsResolver)**:
  - Wyciąga określone roszczenia (takie jak subiekt lub data wygaśnięcia) z tokenu.
- **validateToken(String jwtToken, UserDetails userDetails)**:
  - Waliduje token, porównując nazwę użytkownika i sprawdzając, czy token nie wygasł.
- **isTokenExpired(String jwtToken)**:
  - Sprawdza, czy token wygasł.
- **extractExpiration(String jwtToken)**:
  - Wyciąga datę wygaśnięcia tokenu.

6. RegisterService

- **Opis:** Ten serwis odpowiada za rejestrację użytkowników oraz ich autentykację. Umożliwia rejestrację użytkowników jako zwykłych lub administratorów oraz wydawanie tokenów JWT po udanej próbie logowania.
- **Metody**
  - **verify(Users uzytkownik)**:
    - Uwierzytelnia użytkownika za pomocą jego adresu e-mail i hasła przy użyciu AuthenticationManager. W przypadku sukcesu zwraca token JWT.
  - **register(Users uzytkownik)**:
    - Rejestruje nowego użytkownika, kodując jego hasło przy użyciu PasswordEncoder i zapisując użytkownika w bazie danych. Obsługuje również błędy związane z integralnością danych.

- **registerAdmin(Users uzytkownik)**:
    - Rejestruje użytkownika z rolą administratora oraz obsługuje specyficzne wyjątki, takie jak naruszenie integralności danych lub błędy walidacji.

7. RentalService

- **Opis:** Ten serwis zajmuje się wypożyczaniem i zwracaniem książek. Śledzi również książki, które zostały wypożyczone przez użytkowników.
- **Metody**
    - **rentBook(int id)**:
        - Umożliwia użytkownikowi wypożyczenie książki, jeśli jest ona dostępna. Liczba kopii książki jest zmniejszana, a rekord wypożyczenia jest tworzony w bazie danych.
    - **UserRents()**:
        - Zwraca listę książek, które obecnie są wypożyczone przez zalogowanego użytkownika.
    - **returnBook(int id)**:
        - Umożliwia użytkownikowi zwrot wypożyczonej książki. Ustawiana jest data zwrotu, a liczba dostępnych kopii książki jest zwiększana.

8. UserService

- **Opis:** Ten serwis współpracuje z repozytorium UserRepo i pozwala na pobieranie danych użytkowników oraz obsługę operacji związanych z użytkownikami.
- **Metody**
    - **getAllUsers()**:
        - Zwraca wszystkich użytkowników z bazy danych.
    - **getLoggedInUserDetails()**:
        - Pobiera szczegóły aktualnie uwierzytelnionego użytkownika z kontekstu bezpieczeństwa.
    - **getUserById(Integer id)**:
        - Pobiera użytkownika po jego identyfikatorze.

Klasy serwisowe odpowiedzialne są za logikę aplikacji i zapewniają operacje CRUD na danych (tworzenie, odczyt, aktualizacja, usuwanie) w powiązaniu z repozytoriami. **BooksService**, **AuthorService**, **CategoryService, UserService, RentalService** obsługują książki, autorów, kategorie, użytkowników oraz wypożyczenia natomiast **CustomUserDetailsService**, **JWTService** i **RegisterService** odpowiadają za autentykację użytkowników za pomocą tokenów JWT. **RegisterService** jest również wykorzystywana do rejestracji użytkowników i administratorów.

Kontrolery

1. **AuthorsController**

Kontroler odpowiedzialny za operacje związane z autorami książek. Umożliwia zarządzanie autorami przez administratorów.

- **POST /admin/author/add** – Dodaje nowego autora (tylko dla administratorów).
- **PATCH /admin/author/update/{id}** – Aktualizuje dane autora o podanym identyfikatorze (tylko dla administratorów).
- **DELETE /admin/author/delete/{id}** – Usuwa autora o podanym identyfikatorze (tylko dla administratorów).

2. **BooksController**

Kontroler zarządzający operacjami związanymi z książkami. Umożliwia użytkownikom i administratorom przeglądanie książek oraz administratorom ich dodawanie, edytowanie i usuwanie.

- **GET /books** – Zwraca listę wszystkich książek dostępnych w systemie.
- **GET /books/{id}** – Zwraca szczegóły książki o podanym identyfikatorze.
- **POST /books/byauthor** – Zwraca książki dla podanego autora (wymaga podania danych w ciele zapytania).
- **POST /admin/savebook** – Dodaje książkę do systemu (tylko dla administratorów).
- **GET /admin/allbooks** – Zwraca listę wszystkich książek w systemie (tylko dla administratorów).
- **DELETE /admin/deletebook/{id}** – Usuwa książkę o podanym identyfikatorze (tylko dla administratorów).
- **PATCH /admin/updatebook/{id}** – Aktualizuje dane książki o podanym identyfikatorze (tylko dla administratorów).

3. **CategoryController**

Kontroler odpowiedzialny za zarządzanie kategoriami książek. Umożliwia użytkownikom przeglądanie kategorii, a administratorom ich dodawanie, edytowanie i usuwanie.

- **GET /category** – Zwraca wszystkie kategorie książek.
- **GET /category/{id}/books** – Zwraca książki przypisane do kategorii o podanym identyfikatorze.
- **POST /admin/category/add** – Dodaje nową kategorię (tylko dla administratorów).
- **PATCH /admin/category/update/{id}** – Aktualizuje kategorię o podanym identyfikatorze (tylko dla administratorów).
- **DELETE /admin/category/delete/{id}** – Usuwa kategorię o podanym identyfikatorze (tylko dla administratorów).

4. **CustomErrorController**

Kontroler do obsługi błędów w aplikacji. Kiedy wystąpi błąd, ten kontroler przechwyci go i wyświetli szczegóły w odpowiednim formacie w widoku.

- **GET /error** – Przechwytuje błędy i wyświetla je na stronie z niestandardowym komunikatem o błędzie.

5. **LoginRegisterController**

Kontroler odpowiedzialny za rejestrację i logowanie użytkowników. Umożliwia rejestrację nowych użytkowników oraz administratorów, a także logowanie istniejących użytkowników.

- **POST /login** – Loguje użytkownika i zwraca token uwierzytelniający (używa metody verify z serwisu RegisterService).
- **POST /register** – Rejestruje nowego użytkownika (używa metody register z serwisu RegisterService).
- **POST /admin/register** – Rejestruje nowego administratora (używa metody registerAdmin z serwisu RegisterService).

6. **RentalsController**

Kontroler odpowiedzialny za zarządzanie wypożyczeniami książek. Umożliwia użytkownikom wypożyczenie książki, zwrot książki oraz sprawdzenie swoich wypożyczeń.

- **POST /books/{id}/rent** – Wypożycza książkę o podanym identyfikatorze.
- **GET /user/rents** – Zwraca listę książek wypożyczonych przez aktualnie zalogowanego użytkownika.
- **POST /user/rents/{id}** – Zwraca książkę o podanym identyfikatorze (zwraca wypożyczoną książkę).

7. UserController

Kontroler odpowiedzialny za operacje związane z użytkownikami. Umożliwia administratorom przeglądanie listy użytkowników w systemie.

- **GET /users** – Zwraca listę wszystkich użytkowników (dostępne tylko dla administratorów).

Autentykacja i autoryzacja: Większość z tych kontrolerów wymaga odpowiednich uprawnień użytkowników. Administratorzy mają dostęp do zarządzania autorami, książkami, kategoriami oraz użytkownikami, podczas gdy użytkownicy mogą wypożyczać książki i przeglądać dostępne zasoby.

Operacje CRUD: Kontrolery takie jak AuthorsController, BooksController oferują standardowe operacje CRUD (tworzenie, odczyt, aktualizacja, usuwanie) dla książek i autorów.

Obsługa błędów: Kontroler CustomErrorController przechwyci błędy aplikacji i dostarczy użytkownikowi informacje o błędzie w odpowiednim formacie.

aplication.properties

- Konfiguracja połączenia z bazą danych

```
spring.datasource.url=jdbc:mysql://autorack.proxy.rlwy.net:11507/railway?useSSL=false&serverTimezone=UTC spring.datasource.username=root
spring.datasource.password=BVDsiqtrduOaezUJhQCTZmRwOFaWUMrO spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

- Konfiguracja JPA i Hibernate

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect spring.jpa.show-sql=true spring.jpa.properties.hibernate.format_sql=true spring.jpa.hibernate.ddl-auto=update
```

Frontend

1. Plik: Home.js

Komponent Home służy do pobierania listy książek z backendu, wyświetlania ich w formie listy oraz umożliwiania nawigacji do strony szczegółów danej książki. Wykorzystuje React Hooks (useState, useEffect, useCallback) oraz fetch do komunikacji z backendem.

2. Plik: Rents.js

Komponent Rents odpowiada za pobieranie danych o wypożyczeniach użytkownika z backendu, ich wyświetlanie oraz umożliwienie zwrotu książek. Wykorzystuje hooki React (useState, useEffect) i bibliotekę axios do wysyłania zapytań HTTP.

3. Plik Register.js

Komponent Register odpowiada za obsługę rejestracji użytkowników. Formularz jest dynamiczny – przycisk jest zablokowany w trakcie przesyłania danych, a użytkownik otrzymuje komunikaty o błędach lub sukcesach.

4. Plik Login.js

Komponent Login odpowiada za obsługę logowania użytkownika w aplikacji. Jest to kluczowy element systemu uwierzytelniania.

5. Plik BookDetails.js

Komponent BookDetails służy do wyświetlania szczegółów książki, takich jak tytuł, autor, rok wydania, kategoria i liczba dostępnych kopii. Pozwala użytkownikowi na wypożyczenie książki oraz powrót do strony głównej.

6. Plik App.js

Komponent App stanowi główną konfigurację aplikacji React. Wykorzystuje React Router do nawigacji między stronami oraz implementuje mechanizm sprawdzania ważności tokenu (logowanie i sesje użytkowników).

7. Plik Navbar.js

Komponent Navbar odpowiada za wyświetlanie paska nawigacyjnego w aplikacji. Zawiera przyciski, które pozwalają użytkownikowi na interakcję z aplikacją, takie jak wylogowanie się i przejście do sekcji wypożyczeń. Komponent korzysta z React Router do nawigacji między stronami aplikacji.

Baza danych

Baza danych jest hostowana na platformie **Railway.com** i zawiera tabele związane z systemem zarządzania biblioteką.

Hosting i konfiguracja

Baza danych działa w środowisku **Railway.com**, które jest nowoczesną platformą do hostowania aplikacji i baz danych w chmurze. Railway oferuje automatyczne zarządzanie zasobami, skalowalność.

Kluczowe cechy hostingu:

- **Silnik bazy danych:** MySQL 9.1.0
- **Lokalizacja:** Railway Cloud
- **Zabezpieczenia:** Dostęp do bazy danych możliwy jest tylko przez autoryzowane połączenia.

Baza danych jest dostępna zdalnie przez adres **autorack.proxy.rlwy.net**, a aplikacja łączy się z nią poprzez unikalne poświadczenia dostarczane przez Railway.

```
spring.datasource.url=jdbc:mysql://autorack.proxy.rlwy.net:11507/railway?useSSL=false&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=BVDsiqtrduOaezUJhQCTZmRwOFaWUMrO
```

Rysunek 1 Parametry potrzebne do połączenia z bazą danych

Struktura

Struktura bazy danych została utworzona na podstawie modelu obiektowego w języku Java. Proces generowania tabel odbył się poprzez mechanizm migracji zarządzany przez Hibernate, który na podstawie adnotacji w klasach encji automatycznie stworzył i skonfigurował schemat bazy danych.

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update
```

Rysunek 2 Konfiguracja Hibernate

Relacje w bazie danych

- Każdy użytkownik (users) może wypożyczyć wiele książek (rentals).
- Każda książka (books) należy do jednej kategorii (categories).
- Każda książka ma przypisanego autora (authors).
- Każde wypożyczenie (rentals) odnosi się do jednej książki i jednego użytkownika.

Schemat relacji (ERD - Encja-Relacja)

- **users (1) → (N) rentals**
- **books (1) → (N) rentals**
- **authors (1) → (N) books**
- **categories (1) → (N) books**