# Projekt czytelnia

### Autorzy

Marcin Bereta, Hubert Gancarczyk, Magdalena Skrok, Laura Wiktor

# Opis

Projekt zakłada stworzenie aplikacji obsługującej działalność czytelni, oferującej trzy poziomy dostępu: użytkownika, pracownika i administratora.

Użytkownik ma możliwość rejestracji w systemie, podając swój adres email, nick oraz ustawiając hasło zgodne z regułami. Dzięki temu będzie mógł zalogować się do systemu, wyświetlić polecane książki, ocenić te które chociaż raz czytał, wyświetlić statystyki, sprawdzić okładkę wybranej pozycji, dodatkowo będzie mógł wyświetlić książki danego autora.

Funkcje pracownika obejmują dodanie nowych książek, kategorii czy autora do bazydanych. Natomiast uprawnienia administratora pozwalają na zarządzanie członkami czytelni, zmiany ich ról, dodatkowa administrator posiada te same funkcje co pracownik.

Cała aplikacja jest oparta na języku Java oraz wykorzystuje framework Spring Boot. Struktura projektu została logicznie podzielona na frontend i backend. Backend odpowiada za połączenie z bazą danych oraz obsługę systemu, natomiast frontend zajmuje się warstwą wizualną, zapewniając interakcję użytkownika z aplikacją.

Aby uruchomić projekt należy uruchomić LibraryApp.

# Schemat bazy danych

Obraz zawierający zrzut ekranu, tekst, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznie

Baza danych zawiera sześć tabel obsługujących funkcjonalność aplikacji. Tabela member przechowuje informacje o pracowniku, administratorze lub czytelniku. Tabela Book odpowiada za dane dotyczące pojedynczego egzemplarza książek będących na stanie czytelni. Reading\_room przechowuje dane o korzystających z czytelni, a Rate pozwala na ocenę danej książki przez czytelnika. Tabela Author przechowuje informacje o autorach dostępnych pozycji. Tabela Category dostępne kategorie książek.

# Schemat aplikacji z zależnościami Spring’a

Obraz zawierający tekst, zrzut ekranu, diagram

Opis wygenerowany automatycznie

# Schemat obiektowy aplikacji

Obraz zawierający zrzut ekranu, tekst, miejsce parkingowe/przestrzeń

Opis wygenerowany automatycznie

## Widoki

**Widok użytkownika** - będzie udostępniał najlepsze statystyki, możliwość oceny czytanych książek. Aktualnie udostępnia ich listę. Z widoku użytkownika można przejść do widoku konkretnej książki.  
**Widok pracownika** - będzie umożliwiał rozpoczęcie wizyty czytelnika jak i jej zakończenie, ponadto będzie dostarczał możliwość aktualnego podglądu czytelni - użytkowników i posiadanych przez nich książek.  
**Widok administratora** - umożliwia dodanie nowej książki do czytelni oraz nowego autora. Ma możliwość zmiany funkcji każdego użytkownika.  
**Widok książek** - pozwala przeglądać dostępne pozycje, będzie umożliwiał wyświetlenie wszystkich książek podanego autora lub wybranej kategorii.  
**Widok dodawania książki** - pozwala dodać nową książkę do czytelni.  
**Widok dodawania autora** - pozwala dodać nowego autora do czytelni.  
**Widok dodawania kategorii** - pozwala dodać nową kategorię do czytelni.  
**Widok administratora** - umożliwia dodanie nowej książki do czytelni oraz nowego autora. Ma możliwość zmiany funkcji każdego użytkownika.  
**Widok obecnie wypożyczonych książek** - pozwala wyświetlić wszystkie wypożyczone książki przez użytkownika.  
**Widok listy użytkowników** - pozwala wyświetlić wszystkich użytkowników.  
**Widok oceniania książki** - pozwala ocenić książkę.  
**Widok wypożyczonych książek** - pozwala wyświetlić wszystkie wypożyczone książki aktualnie.  
**Widok wszystkich wypożyczeń** - pozwala wyświetlić wszystkie wypożyczenia.  
**Widok statystyk** - pozwala wyświetlić statystyki.  
**Widok profilu użytkownika** - pozwala wyświetlić profil użytkownika i nim zarządzać.

## Kontrolery:

**BaseController** - abstrakcyjna klasa kontrolera interfejsu użytkownika. Zawiera metody obsługujące nawigację między scenami oraz obsługę logowania.  
**Metody:**  
void redirectToScene(String scenePath, String title, Stage stage)  
void handleSuccessfulLogin(Member member)  
Node getStage()  
void initialize(URL url, ResourceBundle resourceBundle)

**LoginController** - kontroler obsługujący widok logowania dostępny po uruchomieniu aplikacji i przejściu przez odnośnik dotyczący posiadania już konta. W przypadku poprawnych danych loguje użytkownika i przenosi go do nowego widoku, w przeciwnym przypadku wyświetla stosowny komunikat.  
**Metody:**  
void onKeyPressed()  
void redirectToSignUp()  
void onLoginClick()  
void hideErrorMessage()  
void showErrorMessage(String message)  
void handleSuccessfulLogin(Member loggedInUser)  
Node getStage()

**RegisterController** - kontroler obsługujący widok rejestracji dostępny po uruchomieniu aplikacji. W przypadku poprawnych danych rejestruje nowego użytkownika i przenosi go do nowego widoku, w przeciwnym przypadku wyświetla stosowny komunikat. Posiada też opcję przejącia do widoku logowania dla zarejestrowanych użytkowników.  
**Metody:**  
void onKeyPressed()  
void redirectToLogin()  
void onRegisterClick()  
void hideErrorMessage()  
void showErrorMessage(String message)  
void handleSuccessfulLogin(Member loggedInUser)  
Node getStage()

**AddNewAuthorController** - klasa dziedzicząca po BaseController służąca do dodawania nowego autora.  
**Metody:**  
void goToAdminPanel()  
void addAuthorClick()  
Node getStage()

**AddNewCategoryController** - klasa dziedzicząca po BaseController służąca do dodawania nowej kategorii.  
**Metody:**  
void goToAdminPanel()  
void addCategoryClick()  
Node getStage()

**AddNewBookController** - klasa dziedzicząca po BaseController służąca do dodawania nowej książki dla istniejących już w bazie autorów.  
**Metody:**  
void goToAdminPanel()  
void addBookClick()  
void initialize(URL url, ResourceBundle resourceBundle)  
Node getStage()

**AdminPanelController** - klasa dziedzicząca po BaseController służąca do obsługi wyświetlenia możliwości akcji dostępnych dla administratora.  
**Metody:**  
void redirectToMemberList()  
void redirectToAddBook()  
void redirectToAddAuthor()  
void redirectToDashboard()  
Node getStage()

**BookDetailsController** - klasa dziedzicząca po BaseController wyświetlenia szczegółowych informacji odnośnie książki i jej wypożyczenia.  
**Metody:**  
void setBookDetails(Book book)  
void updateView()  
void rentBook()  
void reserveBook()  
void goBackToBookList()  
void updateReserveButtonText(Book bookDetails)  
void updateRentButtonText(Book bookDetails)  
void rateBook()  
void showRates()  
void showRateButton()  
Node getStage()

**BookListController** - klasa dziedzicząca po BaseController wyświetlenia wszystkie książki w bibliotece.  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void displayBooks()  
void setupListView()  
Callback<ListView, ListCell> createListCellFactory()  
Node getStage()

**CurrentRentedBooksController** - klasa dziedzicząca po BaseController wyświetlenia wszystkie książki obecnie wypożyczone przez użytkownika, umożliwia ich zwrot.  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void displayBooks()  
void setupListView()  
Callback<ListView, ListCell> createListCellFactory()  
Node getStage()

**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void redirectToBooksList()  
void redirectToCurrentRentedBooks()  
void redirectToRentedBooksHistory()  
Node getStage()

**MemberListController** - klasa dziedzicząca po BaseController wyświetlenia wszystkich użytkowników biblioteki.  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void displayMembers()  
void setupListView()  
Callback<ListView, ListCell> createListCellFactory()  
Node getStage()

**NavbarController** - klasa dziedzicząca po BaseController wyświetla navbar.  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void redirectToBooksList()  
void redirectToCurrentRentedBooks()  
void redirectToRentedBooksHistory()  
void redirectToDashboard()  
void redirectToAdminPanel()  
Node getStage()

**RentedBooksHistoryController** - klasa dziedzicząca po BaseController wyświetlenia historię wypożyczeń książek.  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void displayBooks()  
void setupListView()  
Callback<ListView, ListCell> createListCellFactory()  
Node getStage()

**RateController** - klasa dziedzicząca po BaseController służy do oceninia książki  
**Metody:**  
void setBook(Book book)  
void addRateClick()  
void goToBook()  
void updateView()  
Node getStage()

**RentedBookController** - klasa służąca do wyświetlania książki wypożyczonej  
**Metody:**  
void setReadingRoom(ReadingRoom readingRoom)

**RentedBooksHistoryController** - klasa dziedzicząca po BaseController służy do wyświetlania wszystkich wypożyczonych książek  
**Metody:**  
void initialize(URL url, ResourceBundle resourceBundle)  
void initializeReadingRooms()  
void setupFilters()  
void displayReadingRooms(List readingRooms)  
void handleClear()  
void handleFilter()  
void handleSearch()  
Node getStage()

## Serwisy

**MemberService** - obsługuje logikę biznesową związaną z obsługą czytelnika - logowanie, rejestracja oraz pobranie listy najnowszych członków.  
**Konstruktor:** MemberService(MemberRepository memberRepository)  
**Metody:**  
ReturnModel register(String username, String password, String email)  
ReturnModel login(String email, String password)  
List<Member> getLatestMembers(int memberCount)

**BookService** - obsłuży logikę biznesową książek - książki danego autora, danej kategorii lub o danym tytule oraz dodanie nowej książki.  
**Konstruktor:** BookService(BookRepository bookRepository, AuthorRepository authorRepository, CategoryRepository categoryRepository)  
**Metody:**  
ReturnModel addBook(Book book, Member member)  
List getFreeBooks()  
Book getBookById(int id)  
ReturnModel addCategory(String name)  
List getAllBooks()  
List getBooksByTitle(String title)  
List getBooksByAuthor(Author author)  
List getBooksByCategory(String category)  
List getBooksWithCustomQuery(Filters filters)

**ReadingRoomService** - obsłuży logikę biznesową czytelni - początek i koniec wizyty oraz aktualnie trwające wizyty jak i aktualną wizytę konkretnego użytkownika.  
**Konstruktor:** ReadingRoomService(ReadingRoomRepository readingRoomRepository, MemberRepository memberRepository, BookRepository bookRepository)  
**Metody:**  
ReturnModel rentBook(Book book, Member member)  
List getRentedBooksByMember  
void returnBook(Book book, Member member)  
List getAllRentedBooks()  
List getAllReturnedBooks()  
List getUserHistory(Member member)  
List getCurrentRentedBooksByMember(Member member) List getUserHistoryWithFilters(Filters filters)  
List getStats()  
ReturnModel reserveBook(Book book, Member member)

**AuthorService** - obsłuży logikę authora - książki danego autora, danej kategorii lub o danym tytule oraz dodanie nowej książki.  
**Konstruktor:** AuthorService( AuthorRepository authorRepository)  
**Metody:**  
List getAllAuthors()  
List getAuthorByName(String authorName)  
ReturnModel addAuthor(Author author)

**EmailSenderService** - obsłuży logikę wysyłania maili .  
**Metody:**  
void sendEmail(String to, String subject, String text)  
void sendBookReminderMail(Book book)  
void sendRentConfirmationMail(Book book)  
void sendUpInQueueMail(Book book)

**RateService** - obsłuży logikę oceniania książki.  
**Konstruktor:** RateService(RatesRepository ratesRepository, MemberRepository memberRepository)  
**Metody:**  
ReturnModel addReview(Rate rate)  
List getReviewsByBookId(Integer bookId)  
List getBookStatistics()

## Persystencje

**MemberRepository** - rozszerza domyślny interfejs JpaRepository o metodę findByName i findById zwracającą obiekt Member na podstawie podanego adresu email, o ile istnieje lub na podstawie podanego Id.

**AuthorRepository** - rozszerza domyślny interfejs JpaRepository o metodę findByName, zwracającą authora, jeżeli taki istnieje.

**CategoryRepository** - rozszerza domyślny interfejs JpaRepository o metodę findByName, zwracającą kategorię, jeżeli taki istnieje.

**CategoryRepository** - rozszerza domyślny interfejs JpaRepository o metody:  
List findByMemberAndBook(Member member, Book book)  
List findByBookId(Integer bookId)  
List<Object[]> findAverageRatings()

**BookRepository** - rozszerza domyślny interfejs JpaRepository o metody:  
List findByTitle(String title)  
List findByAuthor(Author author)  
List findByCategory(String category)  
List findByNullMember()

**ReadingRoom** - rozszerza domyślny interfejs JpaRepository o metody:  
List getRentedBooksByUser(Member member)  
List getCurrentReadingBooksByUser(Member member)  
List getAllRentedBooksByUser(Member member)  
List getCurrentRentedBooks()  
List getPreviouslyRentedBooks()  
List getReadingRoom(Book book, Member member)

## Modele

**Rate** - reprezentujący konkretną opinię - tabelę Rate z bazy, poza podstawowymi metodami będzie posiadał metodę *checkRate*, która sprawdzi czy wartość wystawionej opinii znajduje się w przedziale 0-5.

**Member** - reprezentujący członka czytelni - tabelę Member z bazy, poza podstawowymi metodami posiada metody *verifyEmail* i *verifyPassword*, które sprawdzają poprawność danych wejściowych.  
**Konstruktor:** Member(String name, String email, String password)

**Book** - reprezentujący książkę - tabelę Book z bazy, posiada podstawowe metody.  
**Konstruktor:** Book(String title, String author, String description, String bookCover, String category)

**Author** - reprezentujący autora - tabelę Author z bazy, posiada podstawowe metody addBook, która pozwala dodać do niego napisaną przez niego książkę.  
**Konstruktor:** Author(String name, String description, String nick)

**ReadingRoom** - reprezentujący instancję wypożyczenia - tabelę ReadingRoom z bazy, posiada podstawowe metody.  
**Konstruktor:** ReadingRoom(Member member, Book book, Date start\_date)

**Category** - reprezentujący kategorię - tabelę Category z bazy. **Konstruktor:** Category(String name)

## Pozostałe zaimplementowane klasy potrzebne do działania aplikacji:

### MemberConfigurator

Jest to klasa konfiguracyjna, która w przypadku braku jakiegokolwiek członka biblioteki doda do niej przykładowego użytkownika.

### Permissions

Enum zawierający dostępne funkcje w aplikacji – admin, pracownik, członek biblioteki.

### ReturnModel

Klasa reprezentuje model zwracany przez aplikację, zawiera on obiekt, wiadomość i kod z nim związany.  
**Konstruktor:** ReturnModel(ReturnObject object, String message, int code)

### ReturnObject

Interfejs posiadający jedną metodę *toJson* konwertującą obiekty implementuje ten interfejs do formatu JSON.

### Filters

Klasa służacą do przekazywania filtrów do bazy danych.  
**Konstruktor:** Filters(Category category, Author author, String title, boolean free)  
Filters(Category category, Author author, String title, boolean free, Member member)

### Statistics

Klasa służacą do przekazywania statystyk do frontendu.  
**Konstruktor** Statistics(Double average, Integer count, Book book)  
Statistics(Integer count, Integer day, Integer month, Integer year)

### SpringContext

Singleton przechowujący kontest aplikacji Spring.  
**Konstruktor:** SpringContext()  
**Metody:**  
SpringContext getInstance()  
ConfigurableApplicationContext getContext()  
void setContext(ConfigurableApplicationContext context)

### UserState

Singleton przechowujący informacje o zalogowanym użytkowniku.  
**Konstruktor:** UserState()  
**Metody:**  
UserState getInstance()  
Member getLoggedInUser()  
void setLoggedInUser(Member loggedInUser)

### Validator

Klasa obsługuje sprawdzanie poprawności podawanych przez użytkownika danych.  
**Konstruktor:** Service()  
**Metody:**  
boolean validateEmail(String email)  
boolean validateUsername(String username)  
boolean validatePassword(String password)

### LibraryApp

Klasa obsługuje frontend aplikacji - interfejs użytkownika.  
**Metody:**  
void start(Stage primaryStage)  
void configureStage(Stage primaryStage)  
void init()  
void main(String[] args)

# Wygląd aplikacji

### Widok rejestracji

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

### Widok logowania

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

### Widok listy książek

Obraz zawierający tekst, zrzut ekranu, Strona internetowa

Opis wygenerowany automatycznie

### Widok książki z szczegółami

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

### Widok wypożyczonych książek

Obraz zawierający tekst, zrzut ekranu, osoba

Opis wygenerowany automatycznie

### Widok historii wypożyczeń

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

### Widok panelu administratora

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

### Widok zarządzania użytkownikami

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

### Widok formularza dodawania autora

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

### Widok formularza dodawania książki

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

### Widok statystyk

Obraz zawierający tekst, zrzut ekranu, diagram, Równolegle

Opis wygenerowany automatycznie

### Widok panelu użytkownika

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

### Widok listy książek z filtrami

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, Reklama internetowa

Opis wygenerowany automatycznie

### Widok oceniania książki

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

### Widok formularza oceniania

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

### Widok powiadaminia mailem

Obraz zawierający tekst, zrzut ekranu, Strona internetowa

Opis wygenerowany automatycznie

### Widok formularza dodawania kategorii

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie