

**Politechnika Lubelska**  
**Wydział Elektrotechniki i Informatyki**  
**Katedra Informatyki**



**Laboratorium:** Programowanie aplikacji w chmurze obliczeniowej

**Temat projektu:** System do katalogowania i zarządzania książkami z funkcjami społecznościowymi.

**Lublin, 30.05.2022 r.**

# Projekt zaliczeniowy – raport

---

## *Wykonanie aplikacji w chmurze obliczeniowej*

Autorzy raportu:

1. Karol Hetman
2. Michał Grzeszuk

Grupa: IO 6.2

Rok: 3

Tryb studiów: stacjonarne

Uwagi projektowe:

.....

# SPIS TREŚCI

<b>SPIS TREŚCI</b>	<b>3</b>
<b>1. PROJEKT APLIKACJI – OPIS I WSTĘPNE ZAŁOŻENIA PROJEKTOWE</b>	<b>5</b>
1.1. Wstęp	5
1.2. Aktorzy	5
1.2.1. Opis aktorów	5
1.2.2. Hierarchia aktorów	5
1.3. Wymagania funkcjonalne	5
1.4. Wymagania niefunkcjonalne	6
<b>2. WYKORZYSTANE TECHNOLOGIE</b>	<b>7</b>
2.1. Java + Spring	7
2.1.1. Spring boot	7
2.1.2. Spring Data JPA	7
2.1.3. Spring Security	7
2.1.4. Spring OpenAPI + Swagger UI	7
2.2. Hibernate	7
2.3. MySQL	8
2.4. HTML5 + CSS3 + Bootstrap	8
2.5. JavaScript + Vue.js	8
2.6. Docker + Docker Compose	8
<b>3. PROJEKT BAZY DANYCH</b>	<b>9</b>
3.1. Diagram ERD	9
3.2. Opis Tabel	9
3.3. Wypełnienie danymi	11
<b>4. INTERFEJS REST API</b>	<b>12</b>
<b>5. WDROŻENIE SYSTEMU</b>	<b>16</b>

<b>6. INTERFEJS UŻYTKOWNIKA</b>	<b>20</b>
<b>7. PODSUMOWANIE</b>	<b>27</b>

# 1. PROJEKT APLIKACJI – OPIS I WSTĘPNE ZAŁOŻENIA PROJEKTOWE

## 1.1. Wstęp

Celem pracy jest opracowanie i implementacja projektu aplikacji internetowej, na wzór takich serwisów jak *lubimyczytac.pl* lub *goodreads.com*, której zadaniem jest umożliwienie użytkownikowi prowadzenia katalogu książek, które przeczytał. Poszczególni użytkownicy będą mogli ocenić daną książkę w skali 1-5 oraz wystawić jej opcjonalną recenzję. Ponadto jeżeli użytkownik nie znajdzie interesującej go książki w bazie danych aplikacji, będzie mógł ją dodać samodzielnie. Aby mieć dostęp do wyżej wymienionych funkcjonalności użytkownik musi najpierw założyć konto a następnie się na nie zalogować. W systemie zalogowani użytkownicy będą podzieleni na dwie kategorii: user i admin. Admin poza wszystkimi prawami przysługującymi zwykłemu użytkownikowi będzie miał dostęp do funkcjonalności która jest zarezerwowana tylko dla niego tj. możliwość usuwania książek oraz edycji i usuwania ocen i komentarzy innych użytkowników. W przypadku użytkowników nieposiadających konta w serwisie ich możliwości będą ograniczały się jedynie do przeglądania książek oraz profili innych użytkowników.

## 1.2. Aktorzy

### 1.2.1. Opis aktorów

1. Gość - aktor odwiedzający serwis który nie posiada konta, ma ograniczone uprawnienia. Ma możliwość samodzielnego założenia konta w celu zwiększenia swoich uprawnień.
2. Użytkownik - aktor posiadający wszystkie uprawnienia aktora *Gość* oraz uprawnienia pozwalające mu na zarządzanie swoim profilem. Może mieć wpływ na stan wewnętrzny systemu poprzez np. dodanie nowych treści do serwisu.
3. Administrator - aktor posiadający najwyższe uprawnienia w systemie, pozwalające mu na zarządzanie tym systemem. Posiada również wszystkie uprawnienia przysługujące innym aktorom.

### 1.2.2. Hierarchia aktorów

Gość ← Użytkownik ← Administrator

Gdzie ← oznacza kierunek dziedziczenia: nadklasa ← podklasa.

## 1.3. Wymagania funkcjonalne

1. Rejestracja i logowanie - funkcjonalność umożliwiająca *Gościowi* założenie konta a następnie zalogowanie się na nie w celu uzyskania dostępu do funkcjonalności przeznaczonych tylko dla zarejestrowanych użytkowników.
2. Przeglądanie katalogu książek - funkcjonalność umożliwiająca *Gościowi* i wszystkim aktorom od niego dziedziczącym na przeglądanie dostępnego katalogu książek w systemie.

3. Przeglądanie ocen użytkowników - funkcjonalność umożliwiająca *Gościowi* i wszystkim aktorom od niego dziedziczącym przeglądanie ocen i komentarzy wystawionych przez *Użytkownika* i *Administradora*.
4. Przeglądanie danych użytkownika - funkcjonalność umożliwiająca *Gościowi* i wszystkim aktorom od niego dziedziczącym przeglądanie danych na temat danego *Użytkownika* i *Administradora*.
5. Dodawanie książki do systemu - funkcjonalność umożliwiająca *Użytkownikowi* i wszystkim aktorom od niego dziedziczącym dodanie do systemu nowej książki
6. Usunięcie i modyfikacja książki - funkcjonalność umożliwiająca *Administratorowi* usunięcie lub modyfikację książki która znajduje się w systemie.
7. Dodanie oceny i komentarza do systemu - funkcjonalność umożliwiająca *Użytkownikowi* i wszystkim aktorom od niego dziedziczącym dodanie oceny i opcjonalnego komentarza na temat wybranej książki.
8. Usuwanie i modyfikacja komentarzy i ocen książki - funkcjonalność umożliwiająca *Użytkownikowi* usunięcie i modyfikację oceny i komentarza którego jest autorem. *Administrator* posiada tą samą funkcjonalność jednak umożliwia mu ona również zarządzanie komentarzami i ocenami dodanymi przez innych użytkowników.

#### 1.4. Wymagania нефункционалне

1. Jako język programowania po stronie serwera powinien być wykorzystany język Java w wersji co najmniej 17 wraz z szkieletem programistycznym Spring Boot 2.6.7.
2. System po stronie serwera powinien udostępniać REST API.
3. Jako serwer webowy system powinien wykorzystywać serwer Tomcat 9.0.12.
4. Jako narzędzie budowania i zarządzania zależnościami dla języka Java powinien zostać wykorzystany Apache Maven 3.8.
5. System powinien wykorzystywać MySQL jako system zarządzania bazą danych.
6. Hasła użytkowników powinny być przechowywane w bazie danych w formie skrótu.
7. Do obliczania skrótów haseł powinna być wykorzystana funkcja bcrypt.
8. Do autentykacji i autoryzacji powinny być wykorzystane tokeny JWT weryfikowane po stronie serwera.
9. Po stronie klienta system powinien wykorzystywać język znaczników HTML5, język arkuszy stylów CSS3 oraz język programowania JavaScript w wersji co najmniej ES6.
10. Strona internetowa powinna zostać zbudowana w technice SPA (Single-page application), przy wykorzystaniu szkieletu programistycznego Vue.js 3.
11. Jako menedżer pakietów języka JavaScript powinien być wykorzystany npm.
12. Do konteneryzacji aplikacji powinno zostać wykorzystane narzędzie Docker 20.10.7 oraz docker-compose 1.29.2.
13. Strona internetowa powinna posiadać jednolitą oprawę graficzną.
14. Strona internetowa powinna być responsywna.
15. System powinien działać na komputerach z systemem Windows, Linux oraz na telefonach i tabletach z systemem Android.
16. System obsługiwany na komputerach powinien działać na przeglądarkach Chrome, Firefox oraz Opera.

## 2. WYKORZYSTANE TECHNOLOGIE

### 2.1. Java + Spring

Spring Framework jest szkieletem programistycznym służącym do tworzenia aplikacji w języku Java. Spring powstał w 2003 roku będąc bezpośrednią odpowiedzią na Java EE która charakteryzowała się wysokim stopniem złożoności i pewnymi niedociągnięciami. Do dnia dzisiejszego Java EE otrzymała wiele poprawek które uprościły pracę w tej technologii jednak Spring pozostała preferowanym wyborem dla większości deweloperów. Zawdzięcza to nie tylko temu że jest wciąż prostszy w użyciu od Java EE ale również ze względu na cały ekosystem Spring, który jest platformą dla wielu innych projektów takich jak Spring Security, Spring Data czy Spring Cloud. W prezentowanej pracy zostały wykorzystane liczne projekty wchodzące w skład platformy Spring.

#### 2.1.1. Spring boot

Spring Boot to projekt którego celem jest dostarczenie gotowych, predefiniowanych konfiguracji dla Spring Framework w celu uproszczenia uruchomienia aplikacji. W projektowanej aplikacji Spring wykorzystywany jest do stworzenia i REST API które będzie wykorzystane do komunikacji pomiędzy backendem i frontendem aplikacji.

#### 2.1.2. Spring Data JPA

Spring Data JPA jest narzędziem które ułatwia korzystanie i interakcję z bazą danych w projektach pisanych przy użyciu języka Java. Pozwala on na uniknięcie przez programistę pisania warstwy DAO (Data Access Object), więki dostarczeniu kilku rodzajów interfejsów repozytoriów. Na podstawie sygnatur metod zdefiniowanych przez programistę w tych interfejsach zostanie wygenerowanych kod który odpowiada za interakcję z bazą danych.

#### 2.1.3. Spring Security

Spring Security to wysoce konfigurowalne narzędzie do uwierzytelniania i kontroli dostępu. W prezentowanej pracy Spring Security został użyty w celu implementacji funkcjonalności logowania, rejestracji oraz autentykacji i autoryzacji na podstawie tokenów JWT. Spring Security dostarcza również kilka popularnych funkcji haszujących, w tym funkcję bcrypt która została wykorzystana w tej pracy.

#### 2.1.4. Spring OpenAPI + Swagger UI

Biblioteka pomagająca w automatyzacji generowania dokumentacji API w projektach wykorzystujących Spring Boot. W aplikacji biblioteka ta została wykorzystana w połączeniu ze Swagger UI w celu wygenerowania interfejsu graficznego pozwalającego na bezpośrednią interakcję z REST API.

### 2.2. Hibernate

Hibernate jest frameworkiem stanowiącym najpopularniejszą implementację Java Persistence API (JPA) - oficjalny standard dla mapowania relacyjno obiektowego dla języka Java. Mapowanie relacyjno obiektowe to proces konwersji obiektów na tabele w bazie danych, bez potrzeby stosowania języka SQL. Hibernate jest implementacją JPA która jest domyślnie wykorzystywana przez Spring.

## 2.3. MySQL

MySQL to stworzony przez firmę Oracle otwartoźródłowy system zarządzania relacyjną bazą danych który wykorzystuje język SQL. Jak we wszystkich relacyjnych bazach danych opartych na języku SQL, dane przechowywane są w tabelach składających się z wierszy i kolumn. MySQL umożliwia tworzenie, zarządzanie i odpytywanie bazy danych oraz zapewnienia zabezpieczenia przed nieautoryzowanym dostępem korzystając z mechanizmów uwierzytelniających. W prezentowanym projekcie MySQL jest wykorzystywany do przechowywania i agregowania danych o poszczególnych bytach obecnych w systemie.

## 2.4. HTML5 + CSS3 + Bootstrap

HTML to hipertekstowy język znaczników który jest stosowany do definiowania struktury i treści strony internetowej. Przeglądarki internetowe odczytują składnię języka i interpretują ją, a następnie wyświetlają. CSS czyli kaskadowe arkusze stylów, służą do opisu prezentacji strony internetowej zdefiniowanej uprzednio przy wykorzystaniu HTML. Bootstrap to biblioteka CSS która zawiera zestaw przydatnych narzędzi ułatwiających tworzenie responsywnego interfejsu graficznego użytkownika zgodnie z filozofią *mobile-first design*.

## 2.5. JavaScript + Vue.js

JavaScript to skryptowy oraz wieloparadygmataowy, interpretowany język programowania, umożliwiający obsługę dynamicznego tworzenia treści na stronie internetowej. Jest językiem lekkim i najczęściej wykorzystywanym jako język który jest wykonywany w przeglądarce internetowej, jako część strony internetowej. Vue.js jest frameworkiem wykorzystującym język JavaScript który upraszcza tworzenie interaktywnych stron internetowych. Vue.js koncentruje się na ułatwieniu w tworzeniu aplikacji typu SPA - Single Page Application. W projekcie framework ten został wykorzystany do stworzenia interfejsu użytkownika w postaci aplikacji internetowej. Dzięki zastosowaniu Vuex (biblioteka pozwalająca na tworzenie sesji dla użytkownika) możliwe było zaimplementowanie funkcjonalności autoryzacji użytkownika.

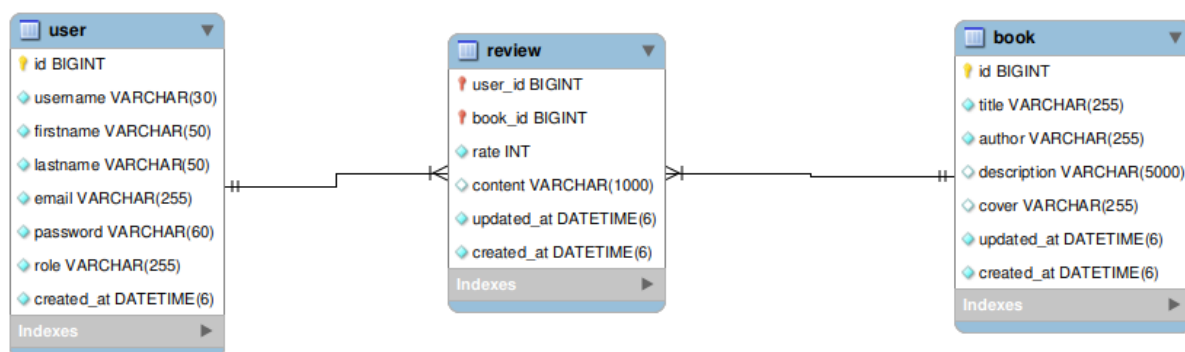
## 2.6. Docker + Docker Compose

Docker to otwartoźródłowe oprogramowanie do tworzenia oraz zarządzania wirtualnymi kontenerami aplikacji. Kontenery pozwalają na odseparowanie aplikacji od wykorzystywanej przez nas infrastruktury co pozwala na łatwiejsze i tańsze budowanie i uruchamianie aplikacji. Docker Compose jest narzędziem do definiowania i uruchomienia wielokontenerowej aplikacji. W tworzonym projekcie Docker wraz z Docker Compose zostały zastosowane do stworzenia konfiguracji która umożliwia zbudowanie i uruchomienie stworzonego projektu przy pomocy jednego polecenia.



### 3. PROJEKT BAZY DANYCH

#### 3.1. Diagram ERD



#### 3.2. Opis Tabel

book			
Nazwa	Typ danych	Więzy integralności	Opis
id	BIGINT	PRIMARY KEY, AUTO INCREMENT	Unikatowy identyfikator książki, identyfikator sztuczny.
title	VARCHAR(255)	NOT NULL	Tytuł książki.
author	VARCHAR(255)	NOT NULL	Imię i nazwisko autora książki.
description	VARCHAR(255)	-	Opcjonalny opis treści książki.
cover	VARCHAR(255)	-	Adres URI do okładki książki.
updated_at	DATETIME(6)	NOT NULL	Data ostatniej modyfikacji danych na temat danej książki.
created_at	DATETIME(6)	NOT NULL	Data dodania danej książki do bazy danych.

user			
Nazwa	Typ danych	Więzy integralności	Opis
id	BIGINT	PRIMARY KEY, AUTO INCREMENT	Unikatowy identyfikator użytkownika, identyfikator sztuczny.

username	VARCHAR(30)	NOT NULL, UNIQUE	Unikalna nazwa użytkownika, wykorzystywana do logowania się do systemu i identyfikacja użytkownika.
firstname	VARCHAR(50)	NOT NULL	Imię użytkownika.
lastname	VARCHAR(50)	NOT NULL	Nazwisko użytkownika.
email	VARCHAR(255)	NOT NULL, UNIQUE	Adres email użytkownika.
password	VARCHAR(60)	NOT NULL	Hasło przechowywane w bazie danych jako skrót, do hashowania wykorzystywana jest funkcja bcrypt a więc skrót będzie miał zawsze 60 znaków.
role	VARCHAR(255)	NOT NULL	Rola użytkownika w systemie.
created_at	DATETIME(6)	NOT NULL	Data utworzenia konta przez użytkownika.

review			
Nazwa	Typ danych	Więzy integralności	Opis
user_id	BIGINT	PRIMARY KEY	Identyfikator użytkownika który wystawił recenzję książki.
book_id	BIGINT	PRIMARY KEY	Identyfikator książki dla której została wystawiona dana ocena. Razem z polem <i>user_id</i> pole to stanowi klucz główny złożony tej tabeli.
rate	INT	NOT NULL	Ocena wystawiona przez użytkownika danej książki.
content	VARCHAR(1000)	-	Opcjonalna recenzja/komentarz do oceny.
updated_at	DATETIME(6)	NOT NULL	Data ostatniej aktualizacji recenzji.
created_at	DATETIME(6)	NOT NULL	Data dodania recenzji.

### 3.3. Wypełnienie danymi

Baza danych jest automatycznie wypełniana danymi podczas uruchamiania aplikacji. Każde uruchomienie aplikacji powoduje usunięcie wszystkich danych z bazy, stworzenie wszystkich tabel na nowo oraz wypełnienie ich przykładowymi danymi. W systemie dostępni są predefiniowani użytkownicy:

Login	Hasło	Rola
kowalski123	password123	ADMIN
hetman123	password123	USER
grzeszuk123	password123	USER

## 4. INTERFEJS REST API

W systemie dostępne są role:

- USER
- ADMIN

Dla uproszczenia została przyjęta obecność jeszcze jednej roli, OWNER. Zakładamy że OWNER posiada wszystkie uprawnienia roli USER oraz uprawnienia do dodatkowych akcji na zasobach które stworzył tj. jeżeli dany USER dodał recenzję to z perspektywy tego zasobu ma on rolę OWNER która daje mu prawa do usunięcia bądź modyfikacji tej recenzji.

USER może modyfikować tylko te zasoby których jest właścicielem. ADMIN może modyfikować dowolne zasoby. Jeżeli jakaś operacja wymaga uprawnień USER to znaczy że ta operacja może być wykonana przez użytkownika z rolą USER oraz z rolą ADMIN, w przypadku OWNER jest analogicznie. Jeżeli dana operacja wymaga roli ADMIN, tylko ADMIN może ją wykonać jako że jest najwyższą rolą w systemie.

Metoda	Ścieżka	Wymagana minimalna Rola	Opis
POST	/api/auth/login	-	Endpoint pozwalający na zalogowanie się użytkownikowi, jeżeli podany login i hasło są zgodne z danymi przechowywanymi w bazie, zostanie zwrócona odpowiedź o statusie 200 oraz token JWT do autoryzacji użytkownika. W przeciwnym wypadku zostanie zwrócony status 401 z odpowiednią wiadomością.
POST	/api/auth/register	-	Endpoint pozwalający na rejestrację użytkownika. Jeżeli rejestracja zakończy się powodzeniem zostanie zwrócony status 200. Jeżeli użytkownik o podanym loginie lub adresie email już istnieje zostanie zwrócony status 409.
POST	/api/auth/valid	-	Endpoint pomocniczy, może zostać wykorzystany do sprawdzenia czy token który użytkownik posiada jest poprawny. Jeżeli token jest

			poprawny zostanie zwrócony status 200, w przeciwnym razie status 401.
GET	/api/books	-	Endpoint zwracający listę ze wszystkimi książkami. Książki mogą być zwracane stronami. Jeżeli w systemie nie ma żadnych książek zostanie zwrócona pusta lista.
GET	/api/books/:id	-	Endpoint zwracający książkę o podanym id. Jeżeli książka o podanym id nie istnieje, zostanie zwrócony status 404.
POST	/api/books	USER	Endpoint służący do utworzenia nowej książki.
PUT	/api/books/:id	ADMIN	Endpoint służący modyfikacji danych na temat książki o podanym identyfikatorze. Jeżeli książka o podanym id nie istnieje zostanie zwrócony status 404.
DELETE	/api/books/:id	ADMIN	Endpoint służący do usunięcia książki o podanym id. Po pomyślnym usunięciu książki zostanie zwrócony status 204. Jeżeli książka o podanym id nie istnieje zostanie zwrócony status 404.
GET	/api/users	-	Endpoint zwracający wszystkich użytkowników. Użytkownicy mogą być zwracani stronami. Jeżeli w systemie nie ma żadnych użytkowników zostanie zwrócona pusta lista.
GET	/api/users/:username	-	Endpoint zwracający dane na temat użytkownika o podanym <i>username</i> . Jeżeli użytkownika o podanej nazwie nie ma, zostanie zwrócony status 404.
PUT	/api/users/:username/grant/:role	ADMIN	Endpoint służący do przyznania użytkownikowi o

			podanym <i>username</i> roli <i>role</i> .
PUT	/api/users/:username/change-pass	USER	Endpoint służący do zmiany hasła użytkownika o podanym <i>username</i> . Jeżeli użytkownik o podanej nazwie nie istnieje zostanie zwrócony status 404. Jeżeli podane stare hasło nie odpowiada obecnemu hasłu lub nowe hasło jest niezgodne z hasłem potwierdzającym zostanie zwrócony status 409.
DELETE	/api/users/:username	ADMIN	Endpoint służący do usunięcia użytkownika o podanym <i>username</i> . Jeżeli usunięcie się powiedzie zostanie zwrócony status 204. Jeżeli użytkownik o podanej nazwie nie istnieje zostanie zwrócony status 404.
GET	/api/reviews/user/:username	-	Endpoint zwracający wszystkie recenzje użytkownika o podanym <i>username</i> . Dane mogą być zwracane stronami. Jeżeli użytkownik o podanej nazwie nie istnieje zostanie zwrócony status 404. Jeżeli użytkownik nie wystawił żadnych recenzji zostanie zwrócona pusta lista.
GET	/api/reviews/book/:bookId	-	Endpoint zwracający wszystkie recenzje dotyczące książki o podanym <i>bookId</i> . Dane mogą być zwracane stronami. Jeżeli książka o podanym <i>bookId</i> nie istnieje zostanie zwrócony status 404. Jeżeli książka nie ma żadnych recenzji zostanie zwrócona pusta lista.
GET	/api/reviews/book/:bookId/stats	-	Endpoint zwracający statystyki książki o podanym <i>bookId</i> . Jeżeli książka o podanym <i>bookId</i> nie istnieje zostanie zwrócony status 404.
GET	/api/reviews/:username/:bookId	-	Endpoint zwracający recenzję

			użytkownika o podanym <i>username</i> wystawioną książkę o podanym <i>bookId</i> . Jeżeli taka recenzja nie istnieje zostanie zwrócony status 404.
POST	/api/reviews/:username/:bookId	USER	Endpoint służący do dodania nowej recenzji. Recenzja jest wystawiana przez użytkownika o podanym <i>username</i> dla książki o podanym <i>bookId</i> . Jeżeli podane <i>username</i> lub <i>bookId</i> jest nieprawidłowe zostanie zwrócony status 404.
PUT	/api/reviews/:username/:bookId	OWNER	Endpoint służący modyfikacji recenzji wystawionej przez użytkownika o podanym <i>username</i> dla książki o podanym <i>bookId</i> . Jeżeli taka recenzja nie istnieje zostanie zwrócony status 404.
DELETE	/api/reviews/:username/:bookId	OWNER	Endpoint służący do usunięcia recenzji wystawionej przez użytkownika o podanym <i>username</i> dla książki o podanym <i>bookId</i> . Jeżeli usunięcie się powiedzie zostanie zwrócony status 204. Jeżeli taka recenzja nie istnieje zostanie zwrócony status 404.

## 5. WDROŻENIE SYSTEMU

Dla przygotowanego systemu został utworzony plik *docker-compose.yml* w celu umożliwienia łatwiejszego zbudowania i uruchomienia tego systemu. W dalszej części tego rozdziału dla uproszczenia aplikacja napisana przy wykorzystaniu Spring'a, będzie nazywana *backendem*, aplikacja napisana przy użyciu Vue.js, której kod wykonuje się po stronie klienta będzie nazywana *frontendem*. Dla backendu został utworzony plik Dockerfile.

```
1 FROM maven:3.8-openjdk-18
2 WORKDIR /backend
3 COPY . .
4 RUN mvn clean install
5 CMD mvn spring-boot:run
```

Rys. 7.1. Dockerfile dla backendu

Tworzy on obraz na podstawie obrazu z narzędziem Maven 3.8 i OpenJDK 18. COPY kopiuje kod aplikacji, następnie uruchamiane jest polecenie *mvn clean install* które odpowiada za skompilowanie kodu źródłowego. *clean* usunie wszystkie wcześniej skompilowane pliki Java z rozszerzeniem *.class*. Następnie maven pobierze wszystkie zależności określone w pliku *pom.xml* oraz rozpocznie kompilację, przeprowadzi testy oraz spakuje skompilowane pliki do jednego pliku *.jar*. Po uruchomieniu kontenera na podstawie tak utworzonego obrazu zacznie wykonywać się program z backendem.

```
1 FROM node:14.19-alpine3.14
2 WORKDIR /frontend
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 RUN npm run build
7 EXPOSE 8080
8 CMD ["npm", "run", "serve"]
```

Rys. 7.2. Dockerfile dla frontendu

W przypadku frontendu Dockerfile tworzy obraz na podstawie obrazu zawierającego Node.js w wersji 14.19. Pierwsze polecenie COPY kopiuje wszystkie pliki zawierające informacje o wykorzystywanych przez aplikację paczkach, polecenie *npm install* pobiera wszystkie paczki zdefiniowane w uprzednio skopiowanych plikach. Następnie kopiowany jest kod źródłowy aplikacji i uruchamiane jest polecenie *npm run build* które rozpoczyna budowanie projektu. Port 8080 jest domyślnym portem wykorzystywanym przez Vue.js. Uruchomienie kontenera na podstawie tak zbudowanego obrazu spowoduje uruchomienie polecenia *npm run serve* - uruchomienie aplikacji frontend.

W celu automatycznego zbudowania środowiska wielokontenerowego dla stworzonej aplikacji na które będzie składać się baza danych, backend oraz frontend został utworzony plik *docker-compose.yml*



```

1  version: "3.8"
2
3  services:
4    back:
5      build: ./backend
6      image: spring-app-img:v1
7      container_name: Spring-boot-app
8      restart: on-failure
9      networks:
10       - backend
11      ports:
12       - 8080:8080
13      volumes:
14       - .m2:/root/.m2
15      environment:
16        SPRING_APPLICATION_JSON: '{
17          "spring.datasource.url" : "jdbc:mysql://db:3306/UserManagementDB",
18          "spring.datasource.username" : "KH_MG_user",
19          "spring.datasource.password" : "password",
20          "spring.datasource.driver-class-name" : "com.mysql.cj.jdbc.Driver",
21          "spring.jpa.hibernate.ddl-auto" : "none",
22          "spring.sql.init.mode" : "always",
23          "spring.mvc.pathmatch.matching-strategy" : "ant_path_matcher"
24        }'
25      stdin_open: true
26      tty: true
27      depends_on:
28        db:
29          condition: service_healthy

```

Pierwszy zdefiniowany serwis odpowiada za stworzenie kontenera dla backendu. Buduje on obraz backendu, cały kod backendu wraz z plikiem Dockerfile umieszczone są w katalogu *backend*. Kontener będzie resetowany w przypadku gdy zwróci on niezerowy *exit code*. Domyślnie Spring korzysta z portu 8080 i taki właśnie port jest mapowany. Tworzony jest ponadto wolumen, katalog *.m2* to lokalne repozytorium Maven'a w którym przechowuje wszystkie artefakty, dzięki stworzeniu tego wolumenu Maven nie będzie musiał przy każdym budowaniu pobierać od nowa wszystkich zależności, jeżeli kontener z backendem zostanie usunięty. Zdefiniowane zmienne środowiskowe zawierają całą konfigurację z pliku *application.properties* - plik zawierający konfigurację Springa. *stdin open* i *tty* to polecenia które odpowiadają za udostępnienie interaktywnej powłoki. Backend jest zależny od bazy danych i nie zostanie uruchomiony dopóki baza danych nie będzie gotowa na przyjmowanie połączeń.

```

31     front:
32         build: ./frontend
33         image: vuejs-app-img:v1
34         container_name: VueJS-app
35         restart: on-failure
36         networks:
37             - frontend
38         ports:
39             - 8081:8080
40         volumes:
41             - /app/node_modules
42         depends_on:
43             - back
44

```

Kolejnym serwisem zdefiniowanym w docker-compose jest frontend. Jest on budowany na podstawie pliku obrazu stworzonego z Dockerfile umieszczonego w katalogu *frontend* wraz z całym kodem frontendu. Port 8080 frontendu jest mapowany na port 8081, dzięki temu otworzona aplikacja internetowa będzie dostępna pod adresem <http://localhost:8081>. Frontend zależy od backendu i zostanie uruchomiony dopiero po uruchomieniu backendu.

```

45     db:
46         image: mysql:8.0
47         container_name: Cloud-92876-92874
48         restart: unless-stopped
49         networks:
50             backend:
51                 ipv4_address: 10.0.10.3
52         volumes:
53             - db:/var/lib/mysql
54         ports:
55             - 3307:3306
56         environment:
57             MYSQL_ROOT_PASSWORD: password
58             MYSQL_USER: KH_MG_user
59             MYSQL_PASSWORD: password
60             MYSQL_DATABASE: UserManagementDB
61         healthcheck:
62             test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
63             timeout: 5s
64             retries: 40
65

```

Ostatnim serwisem aplikacji jest baza danych. Jest ona tworzona na podstawie gotowego obrazu *mysql:8.0* pobranego z DockerHub. Baza danych wykorzystuje tę samą sieć co backend z tą różnicą że w przypadku bazy danych jej adres IP został zdefiniowany w

docker-compose, w przypadku backendu adres ten jest wybierany i przyznawany automatycznie. `/var/lib/mysql` to katalog w którym przechowywane są pliki bazy danych, dzięki utworzeniu wolumenu w tym folderze zapobiegniemy utracie danych po usunięciu kontenera. Domyślny port wykorzystywany przez MySQL jest mapowany na port 3307 a następnie ustawiane jest hasło dla konta root'a oraz tworzony jest dodatkowy użytkownik (który będzie miał uprawnienia root'a) oraz tworzona jest nowa baza danych. Ponadto dla serwisu `db` został zdefiniowany `healthcheck`. Dzięki temu kontener będzie miał ustawiony `health status` który Docker będzie testował wykonując polecenie zdefiniowane w `test` - Docker będzie pingował co 5 sekund bazę danych w celu sprawdzenia czy działa. Jeżeli po 40 próbach status kontenera nie zmieni się na `healthy` kontener przyjmie status `unhealthy`. Zdefiniowanie takiego `healthcheck` ma na celu spowodowanie aby backend czekał aż baza danych zakończy inicjalizację i będzie gotowa na przyjmowanie połączeń. Bez `healthcheck` Spring rozpoczyna próbę nawiązania połączenia z bazą danych zaraz po tym jak ta została uruchomiona ale nie jest jeszcze gotowa na przyjmowanie połączeń.

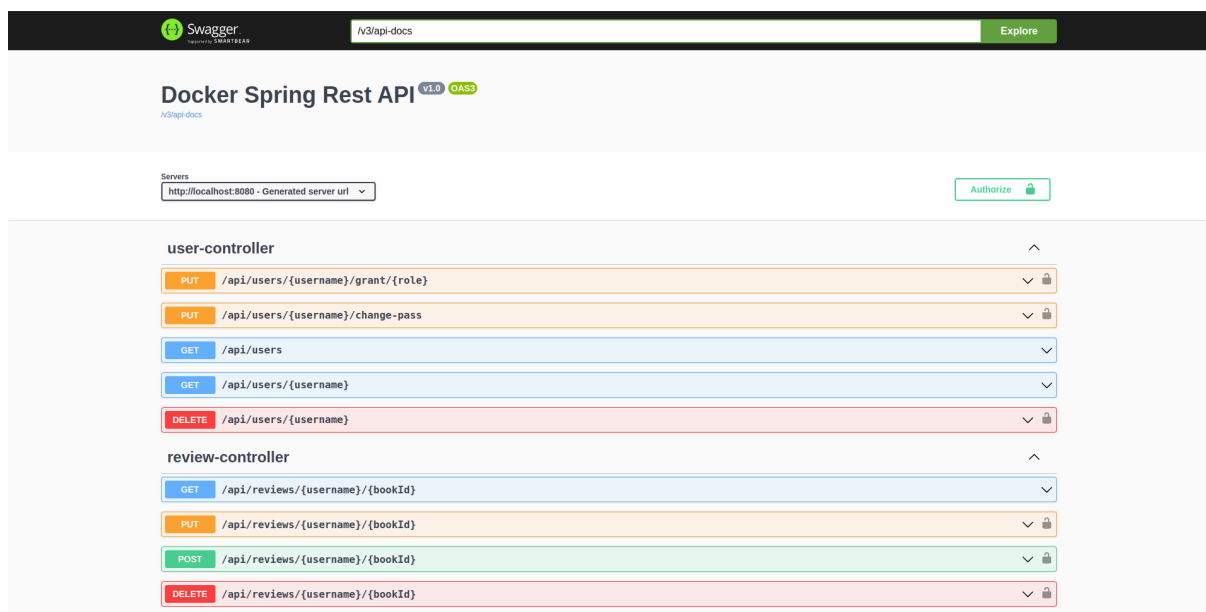
```
66 networks:
67     backend:
68         name: Bridge-Zadanie-1
69         driver: bridge
70         ipam:
71             config:
72                 - subnet: 10.0.10.0/24
73     frontend:
74         driver: bridge
75
76 volumes:
77     db:
```

Zgodnie z wytycznymi do zadania została utworzona sieć `backend` korzystająca ze sterownika `bridge` i wykorzystująca podsieć 10.0.10.0/24. Adresy z tej podsieci zostaną przyznane serwisom `back` i `db`. Dodatkowo zdefiniowana została sieć `frontend` z której korzysta serwis `front`. Podsieć dla tej sieci zostanie wybrana automatycznie. Tworzony jest również nazwany wolumen dla bazy danych, dzięki temu wolumen nie zostanie usunięty po usunięciu kontenera z bazą danych a więc wszystkie dane zostaną zachowane.

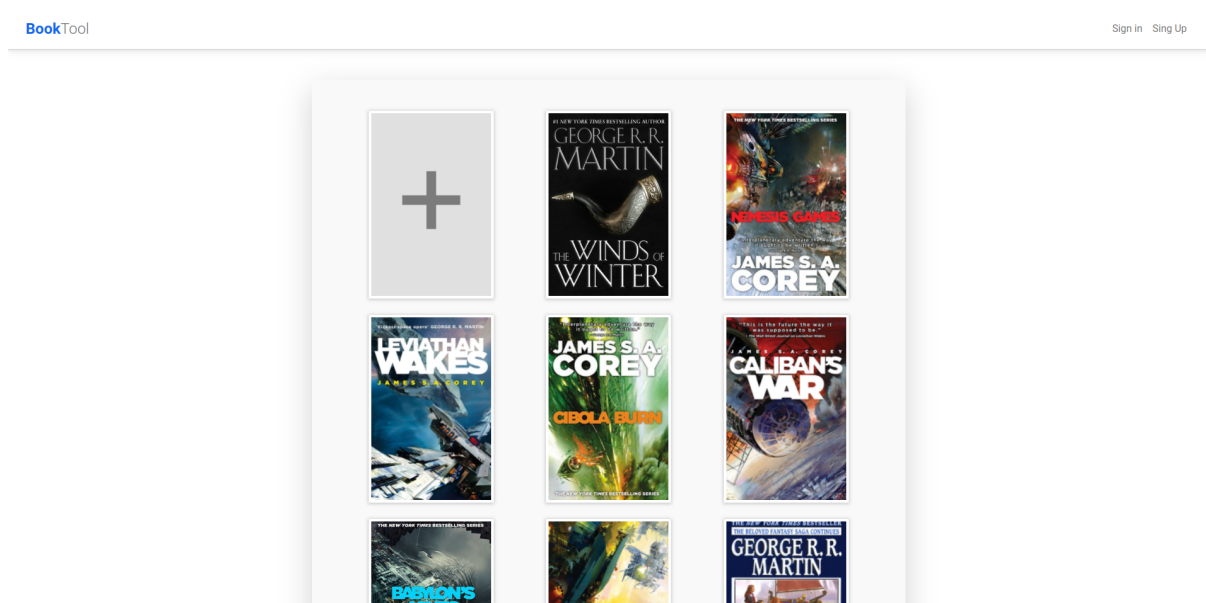
Po wydaniu polecenia `docker-compose up` rozpocznie się budowanie i uruchamianie poszczególnych kontenerów z serwisami aplikacji. Po zakończeniu inicjalizacji aplikacja będzie dostępna pod adresami:

- <http://localhost:8081> - strona główna aplikacji internetowej.
- <http://localhost:8080/swagger-ui/index.html> - Swagger UI - interfejs graficzny do wizualizacji i bezpośredniej interakcji z udostępnianym REST API.

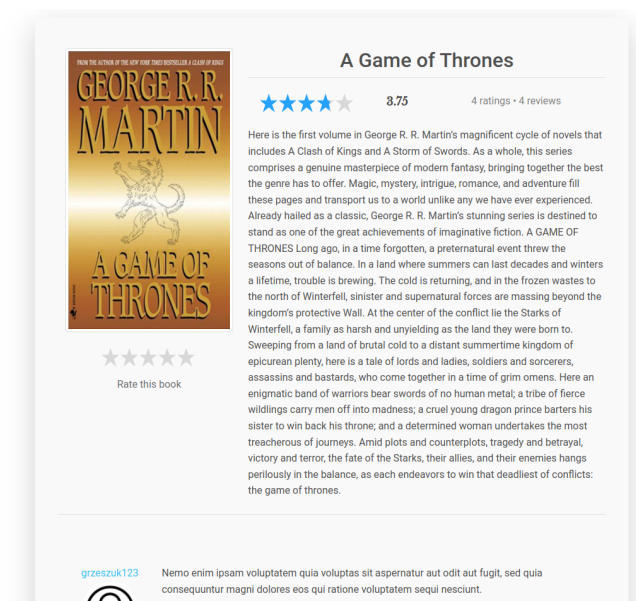
## 6. INTERFEJS UŻYTKOWNIKA



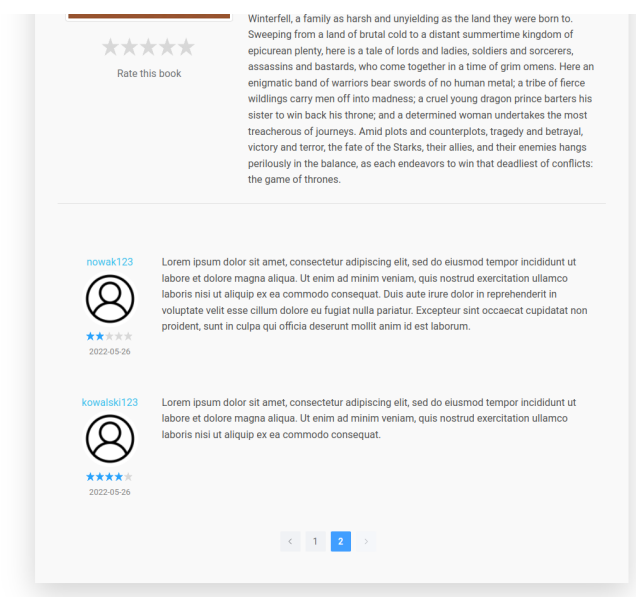
Rys. 6.1. Widok Swagger UI - interfejs graficzny do bezpośredniej interakcji z REST API.



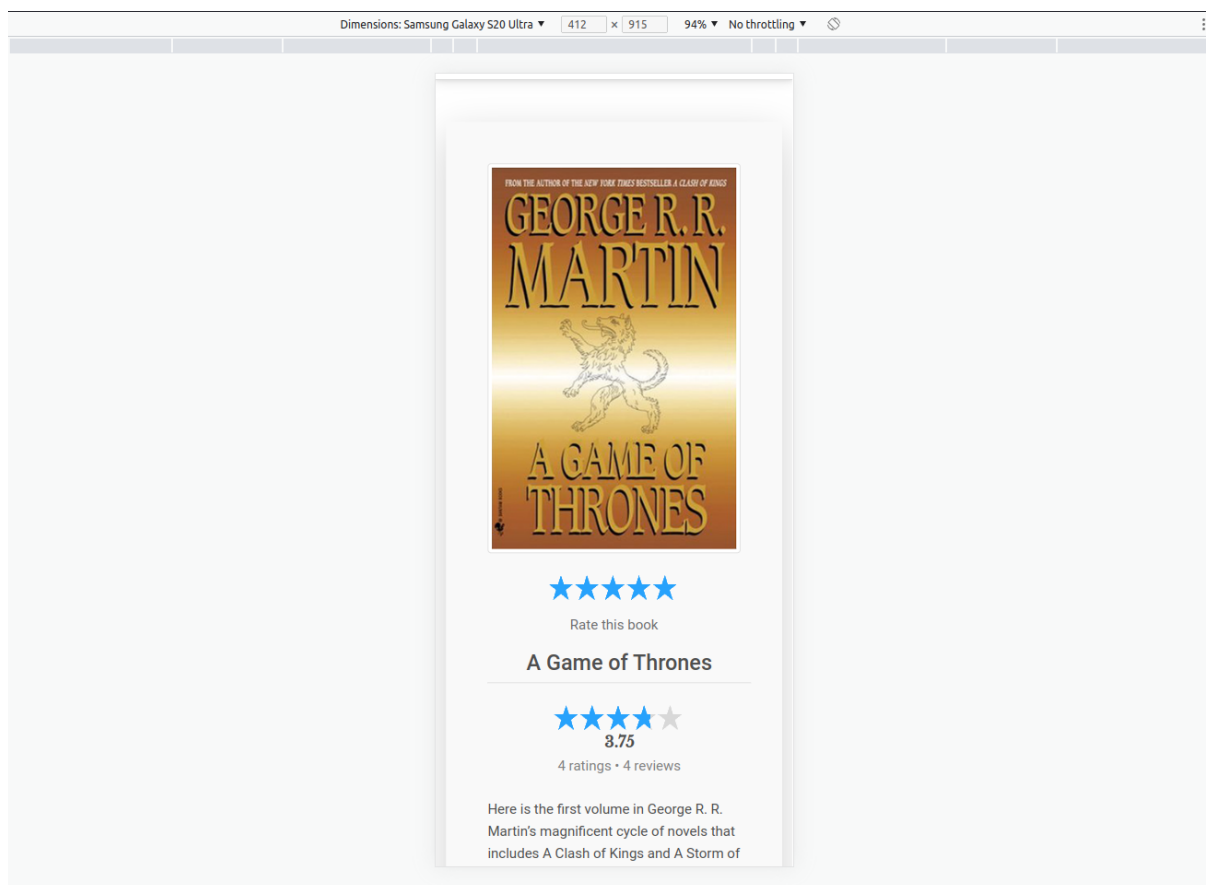
Rys. 6.2. Strona główna aplikacji internetowej - katalog książek.



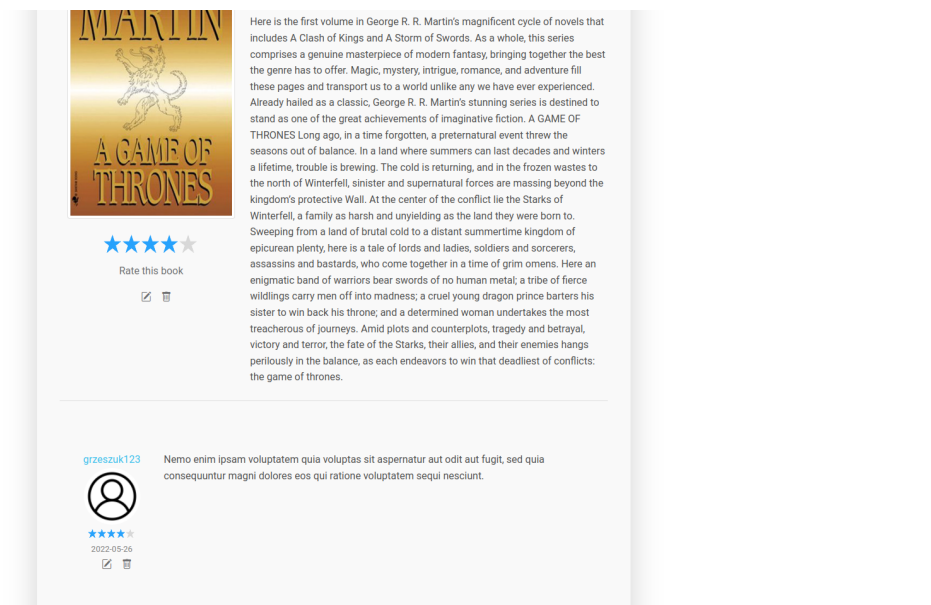
Rys. 6.3. Strona z szczegółami na temat konkretnej książki.



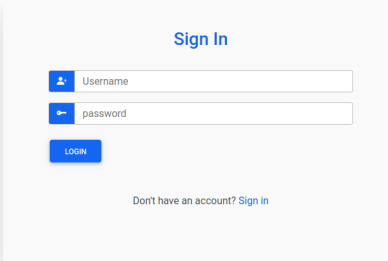
Rys. 6.4. Sekcja komentarzy.



Rys. 6.5. Strona z szczegółami książki na urządzeniu mobilnym - responsywność strony.



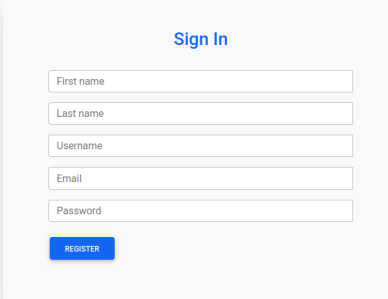
Rys. 6.6. Widok użytkownika z rolą ADMIN - ikonki umożliwiające edycję i usuwanie książek i komentarzy.



Sign In

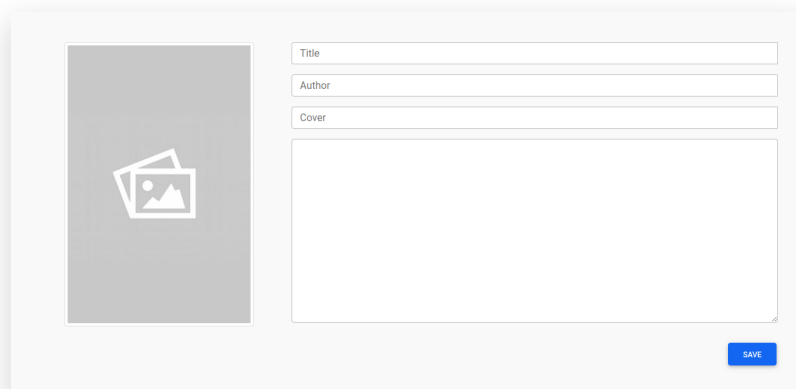
Don't have an account? [Sign in](#)

Rys. 6.7. Ekran logowania.



Sign In

Rys. 6.8. Ekran Rejestracji.



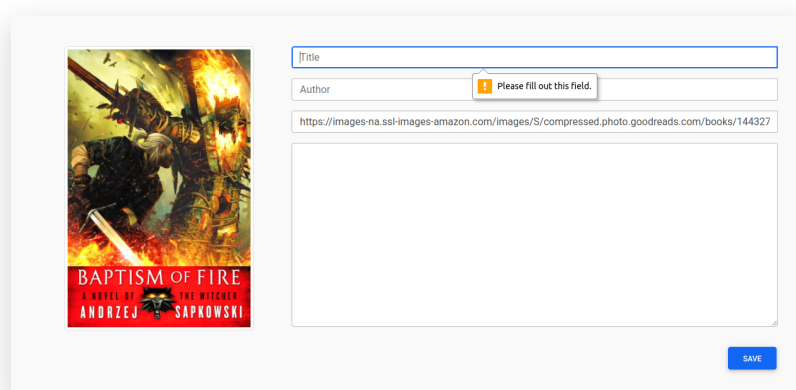
Title

Author

Cover

SAVE

Rys. 6.9. Formularz do edycji i utworzenia nowej książki.



Title

Author Please fill out this field.

<https://images-na.ssl-images-amazon.com/images/S/compressed.photo.goodreads.com/books/144327>

SAVE

Rys. 6.10. Walidacja danych w formularzu. Po podaniu adresu URL do okładki została ona wstawiona do okna podglądu.




Rating *"The Winds of Winter"*


★★★★★

SAVE

Rys. 6.11. Formularz do wystawienia oceny i recenzji książki.



hetman123



★★★★★

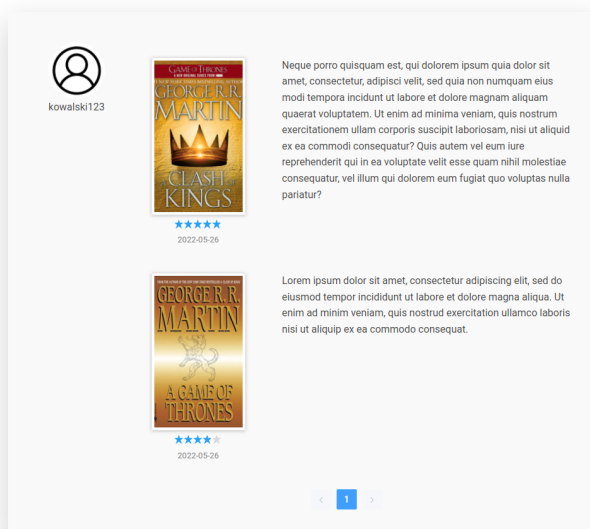
2022-05-26

☒ ☐

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

< 1 >

Rys. 6.12. Widok profilu zalogowanego użytkownika - ikonki do edycji i usunięcia recenzji.



Rys. 6.13. Widok profilu innego użytkownika niż aktualnie zalogowany.

## 7. PODSUMOWANIE

Celem naszej pracy było zaimplementowanie aplikacji umożliwiającej użytkownikom gromadzenie informacji o przeczytanych książkach oraz ocena i recenzja innych książek i ich różnych wydań. Każdy użytkownik ma również możliwość dodawania swoich ulubionych książek, których nie znalazł w serwisie. Spełnione zostały także takie funkcjonalności jak logowanie, rejestracja oraz identyfikowanie użytkownika czy posiada uprawnienia usera lub admina.

Spring Framework pomógł nam stworzyć aplikację łączącą się z bazą danych oraz wystawiającą REST API. HTML5, CSS3 i Bootstrap umożliwiły nam stworzenie czytelnego dla użytkownika interfejsu graficznego. Dodatkowo interfejs jest w pełni responsywny. Vue.js jest frameworkiem języka JavaScript, dzięki któremu udało nam się sprawdzić aby interfejs użytkownika był dynamiczny oraz stworzyć weryfikację użytkownika w serwisie i określenie jego roli.

Całość została zamknięta i otoczona plikiem Dockerfile oraz docker-compose. Docker pozwala na tworzenie oraz zarządzanie wirtualnymi kontenerami aplikacji. Docker-compose pozwolił nam za pomocą jednego pliku zbudować i uruchomić kilka kontenerów jednocześnie tym samym, uruchamiając całą stworzoną aplikację za pomocą jednego polecenia. Nasz plik docker-compose definiuje trzy serwisy:

- serwis warstwy backendu - aplikacja w języku Java napisana przy użyciu szkieletu Spring która odpowiada za komunikację z bazą danych i wystawia REST API.
- serwis warstwy frontendu - interfejs graficzny użytkownika w postaci Single Page Application, wykonany przy użyciu HTML, CSS, Bootstrap oraz Vue.js
- serwis z bazą danych - baza danych MySQL przechowująca dane trwale systemu.

Udało nam się spełnić założenia początkowe projektu, dzięki czemu stworzyliśmy kompleksową, responsywną i czytelną stronę dla miłośników książek. Projekt jest w pełni funkcjonalny, obejmując najważniejsze cechy takich aplikacji jak np. zapis i odczyt z bazy danych, weryfikacja użytkownika, możliwość logowania czy ułatwione wdrożenie poprzez zastosowanie technologii Docker.