Projekt aplikacji do zarządzania serialami

Realizowany w ramach kursu „Projektowanie Oprogramowania”

Marta Fiedorowicz
Krzysztof Marczyński

Spis treści

(1)
Wizja, słownik

Aplikacja do zarządzania oglądanymi serialami

Wizja

1. Wprowadzenie

W dokumencie opisano wizję systemu, który będzie umożliwiał przeglądanie informacji o serialach. Zalogowani użytkownicy będą mogli oznaczać ulubione seriale i obejrzane odcinki oraz dodawać do nich komentarze. Użytkownik będzie mógł również przeglądać statystyki dotyczące oglądanych przez niego seriali w tym zbiorcze podsumowanie łącznej liczby godzin spędzonej na oglądaniu seriali. System będzie wysyłał użytkownikom informacje o premierach nowych odcinków obserwowanych przez nich seriali. Opis odcinków seriali będzie zawierał krótki opis fabuły, datę premiery, czas trwania oraz listę stron internetowych na których można obejrzeć odcinek.

2. Pozycjonowanie

2.1 Sformułowanie problemu

Problem	*Problem z pamiętaniem faktów potrzebnych przy oglądaniu seriali*
Dotyczy	*Widzów seriali, twórców seriali*
Wpływ problemu	*Widz nie wie jaki odcinek włączyć kolejny* *Widz nie wie ile czasu zajmie aktywność* *Widz nie wie na jakiej platformie ma oglądać serial*
Pomyślne rozwiązanie	*Będzie zwalniało widzów z konieczności pamiętania faktów o serialach*

2.2 Opis pozycji produktu

Dla	*Osób oglądających seriale, używających nowych technologii*
Który	*Chce łatwiej zarządzać oglądanymi serialami*
(Nazwa produktu)	*Webowa aplikacja rozrywkowa*
Który	*Skraca czas potrzebny do zapisywania informacji o serialach*
Inaczej niż	*Filmweb*
Nasz produkt	*Skupia się na serialach i przez to jest dużo łatwiejszy w obsłudze, oraz potrafi wyświetlać różne informacje zbiorcze*

3. Opis udziałowców i użytkowników

3.1 Podsumowanie udziałowców

Nazwa	Opis	Odpowiedzialności
Zarząd	*Osoby zarządzające projektem*	*Przygotowanie projektu* *Zarządzanie finansami* *Zarządzanie kadrą* *Sprawdzanie rozwoju* *Marketing* *Zapewnienie spójności wymagań*

3.2 Podsumowanie użytkowników

Nazwa	Opis	Odpowiedzialności
Użytkownik	*Osoba używająca systemu*	*Dodaje nowe seriale do bazy danych* *Zgłasza problemy moderatorowi* *Używa założonego konta* *Zaznacza obejrzane odcinki* *Komentuje seriale* *Komentuje odcinki* *Ocenia odcinki* *Ocenia seriale* *Przegląda statystyki*
Moderator	*Osoby zarządzające danymi w gotowym projekcie*	*Wprowadza nowe seriale do systemu* *Reaguje na problemy zgłoszone przez użytkowników*

4. Opis produktu

4.1 Potrzeby i cechy

Potrzeba	**Priorytet**	**Cechy**	**Planowane wydanie**
Użytkownik potrzebuję widzieć listę seriali i ich odcinków	must	Wyszukiwanie seriali Przeglądanie odcinków dostępnych dla danego serialu	1.0
Użytkownik potrzebuje pamiętać ostatni odcinek oglądanego serialu	must	Zaznaczanie obejrzanych odcinków	1.0
Użytkownik musi pamiętać oglądane seriale	must	Zaznaczanie oglądanych seriali	1.0
Użytkownik chce pamiętać jakie wrażenie zrobił na nim dany odcinek/serial	must	Ocenianie poszczególnych odcinków i seriali	1.0
Użytkownik chce mieć dostęp do danych z różnych urządzeń	must	Tworzenie kont użytkownika Logowanie się do kont użytkownika Dane użytkownika mają być przechowywane na serwerze	1.0
Użytkownik może pamiętać kiedy pojawiają się kolejne odcinki	should	Harmonogram dat premier odcinków	1.1
Użytkownik chce dzielić się z innymi ludźmi komentarzami na tematy serialowe	could	Tworzenie tematycznych wątków Możliwość zostawiania komentarzy	1.2
Użytkownik chce móc zareagować na niewłaściwe zachowania innych użytkowników	should	Zgłaszanie użytkowników lub ich wypowiedzi moderatorowi	1.1
Użytkownik chce móc sprostować fałszywe dane	should	Zgłaszanie fragmentu danych moderatorowi z sprostowaniem	1.1
Użytkownik chce oglądać statystyki	could	Tworzenie statystyk użytkowników Tworzenie statystyk seriali	1.2
Użytkownik chce otrzymywać powiadomienia dotyczące pojawiania się nowych odcinków	could	Wysyłanie powiadomień o nowych informacjach dotyczących seriali	1.3
Użytkownik chce móc porozumieć się bezpośrednio z innym użytkownikiem przez wysłanie mu wiadomości	won't	-	-
Moderator chce łatwo wykonywać swoją pracę	could	Tworzenie kolejki priorytetowej zgłoszeń uwag użytkowników Sztuczna inteligencja wskazuje potencjalnie szkodliwe komentarze	1.4

4.2 Inne wymagania produktowe

Wymaganie	**Priorytet**	**Planowane wydanie**
System działa poprawnie w przeglądarce Google Chrome i Firefox	must	1.0
System działa poprawnie w przeglądarce Edge i Opera	should	1.2
System działa poprawnie w przeglądarce Internet Explorer i Safari	could	1.3
System działa na urządzenia mobilnych Android	should	1.1
System działa na urządzeniach mobilnych iOS	should	1.1
Kopia bazy danych seriali i danych wprowadzonych użytkowników jest regularnie wykonywana	must	1.0
System jest dostępny w angielskiej wersji językowej	must	1.0
System wyświetla spersonalizowane reklamy	could	1.2
System ma czytelny i ładny interfejs	could	1.2
Aplikacja webowa jest w pełni responsywna i wygodna do używania niezależnie od wielkości ekranu urządzenia klienckiego	should	1.1
Aplikacja webowa udostępnia część funkcji offline	could	1.3

SŁOWNIK

1. **Data premiery**
   Data upublicznienia odcinka lub sezonu w telewizji lub w internecine

2. **Filmweb**
   Serwis służący do oceniania i komentowania filmów oraz seriali. Nie służy jednak do śledzenia informacji dotyczących konkretnych odcinków seriali.

3. **Komentarz**
   Subiektywna ocena użytkownika dotycząca serialu lub odcinka wyrażona krótkim i zwięzłym tekstem.

4. **Moderator**
   Osoba nadzorująca przestrzeganie kultury przez użytkowników oraz weryfikująca spójność publikowanych w obrębie aplikacji informacji z rzeczywistością.

5. **Ocena**
   Subiektywna ocena użytkownika dotycząca serialu lub odcinka wyrażona w skali od 1 do 10.

6. **Odcinek**
   Część serialu prezentowana w postaci jednego filmu. Typowa długość odcinka zawiera się między 20 a 40 minut.

7. **Powiadomienie**
   Informacja dostarczana użytkownikowi o aktualnościach dotyczących obserwowanych przez niego seriali. Może na przykład dotyczyć informacji o premierze nowego odcinka serialu.

8. **Serial**
   Program telewizyjny, wieloodcinkowy film. Istotą serialu jest udostępnianie go odbiorcom w odcinkach, zwykle w regularnych odstępach czasu. Seriale uporządkowane są w sezony, które natomiast zawierają odcinki.

9. **Sezon**
   Zbiór odcinków publikowanych w zgodności tematycznej (prowadzenie spójnych wątków tematycznych i historii w obrębie sezonu) lub w zgodności czasu (publikowanie określonej liczby odcinków w stałych odstępach czasowych a następnie dłuższa przerwa przed rozpoczęciem kolejnego sezonu).

10. **Statystyka Serialu**
    Zbiorcze informacje dotyczące popularności serialu.

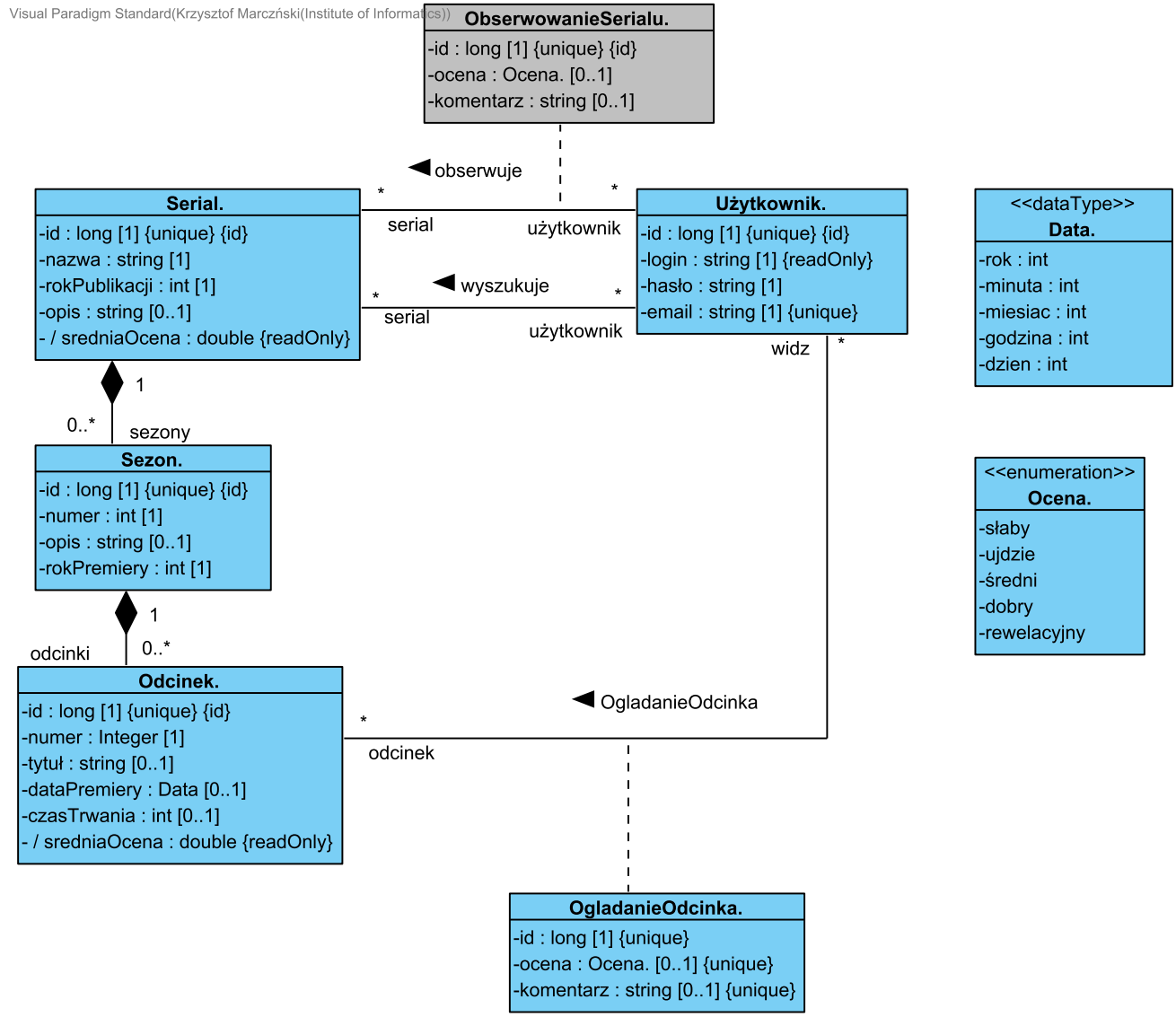
11. **Statystyka Użytkownika**
    Podsumowanie informacji dotyczących aktywności użytkownika w obrębie aplikacji (np. łączna liczba godzin spędzona na oglądaniu seriali na podstawie oznaczonych w aplikacji obejrzanych odcinków).

12. **Użytkownik**
    Osoba używająca aplikacji w celu zarządzania obejrzanymi przez siebie serialami.

(2)

Model domenowy. Reguły biznesowe

1. Diagram klas



1.1. Data.

1.2. ObserwowanieSerialu.

1.3. Ocena.

1.4. Odcinek.

1.5. OgladanieOdcinka.

1.6. Serial.

1.7. Sezon.

1.8. U#ytkownik.

Reguły biznesowe

Ograniczenia struktury

Użytkownik **musi** mieć unikatowy login.

Użytkownik **musi** mieć unikatowy email.

Hasło użytkownika **nie może** zawierać polskich znaków.

Hasło użytkownika **musi** składać się z minimum 8 znaków.

Adres email użytkownika **musi** zawierać dokładnie jeden symbol ‘@’

Data premiery odcinka **nie może** być starsza niż data premiery serialu.

Odcinek **musi** mieć przydzielony numer.

Sezon **musi** mieć przydzielony numer.

Numer sezonu **musi** być unikalny w obrębie serialu.

Numer odcinka **musi** być unikalny w obrębie sezonu.

Ocena **musi** być w określonym formacie.

Serial **musi** mieć przydzieloną nazwę.

Serial **musi** mieć przydzielony rok publikacji.

Ograniczenia operacji

Widz **może** ocenić odcinek **tylko** kiedy go obejrzał.

Użytkownik **może** ocenić serial **tylko** kiedy go obserwuje.

Serial **nie może** zostać wprowadzony do bazy seriali, jeśli jego nazwa i rok nie są unikalne.

Użytkownik **musi** znać nazwę serialu, żeby go wyszukać.

Użytkownik **nie może** zmienić loginu.

Wnioski

Użytkownik **musi być uznany** za zainteresowanego serialem **jeśli** obejrzał przynajmniej 30% wszystkich dostępnych odcinków.

Wysłanie powiadomienia o premierze odcinka **może być wykonane** dla użytkownika **kiedy** użytkownik obejrzał wszystkie poprzednie dostępne odcinki.

Użytkownik **musi być uznany** za priorytetowego jeśli kliknął w przynajmniej 3 reklamy wyświetlane w aplikacji.

Jeśli widz oznaczy odcinek jako obejrzany, serial **musi** być uznany za obserwowany.

Obliczenia

Średnia ocena odcinka **musi być obliczona** jako średnia ocen wszystkich użytkowników.

Średnia ocena serialu **musi być obliczona** jako średnia ocen wszystkich użytkowników.

To czy zbanować użytkownika **musi być obliczone** równaniem: użytkownik miał lub ma 10 zgłoszonych komentarz i moderator usunął przynajmniej połowę lub moderator tak decyduje.

Wskazówki

Odcinek **nie musi** mieć ustalonej premiery.

Użytkownik **może** zostawić komentarz do odcinka.

Użytkownik **może** zostawić komentarz do serialu.

Odcinek **może** posiadać opis.

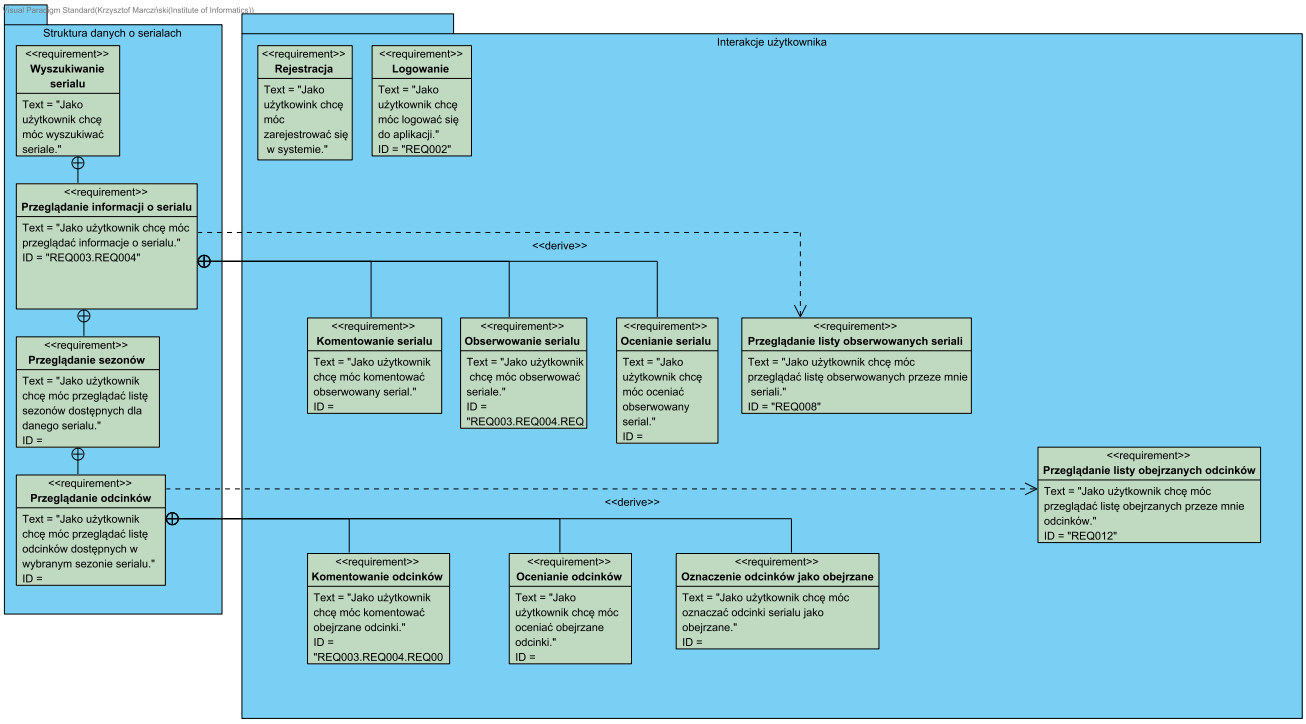
Sezon **może** posiadać opis.

Serial **może** posiadać opis.

(3)

Specyfikacja wymagań

1. Requirement Diagram



1.1. Interakcje u#ytkownika

1.2. Komentowanie odcinków

ID: REQ003.REQ004.REQ005.REQ006.REQ013
Jako u#ytkownik chc# móc komentowa# obejrzane odcinki.

1.3. Komentowanie serialu

ID: REQ003.REQ004.REQ010
Jako u#ytkownik chc# móc komentowa# obserwowany serial.

1.4. Logowanie

ID: REQ002
Jako u#ytkownik chc# móc logowa# si# do aplikacji.

1.5. Obserwowanie serialu

ID: REQ003.REQ004.REQ007
Jako u#ytkownik chc# móc obserwowa# seriale.

1.6. Ocenianie odcinków

ID: REQ003.REQ004.REQ005.REQ006.REQ014
Jako u#ytkownik chc# móc ocenia# obejrzane odcinki.

1.7. Ocenianie serialu

ID: REQ003.REQ004.REQ009
Jako u#ytkownik chc# móc ocenia# obserwowany serial.

1.8. Oznaczenie odcinków jako obejrzane

ID: REQ003.REQ004.REQ005.REQ006.REQ011
Jako u#ytkownik chc# móc oznacza# odcinki serialu jako obejrzane.

1.9. Przegl#danie informacji o serialu

ID: REQ003.REQ004
Jako u#ytkownik chc# móc przegl#da# informacje o serialu.

1.10. Przegl#danie listy obejrzanych odcinków

ID: REQ012
Jako u#ytkownik chc# móc przegl#da# list# obejrzanych przeze mnie odcinków.

1.11. Przegl#danie listy obserwowanych seriali

ID: REQ008
Jako u#ytkownik chc# móc przegl#da# list# obserwowanych przeze mnie seriali.

1.12. Przegl#danie odcinków

ID: REQ003.REQ004.REQ005.REQ006
Jako u#ytkownik chc# móc przegl#da# list# odcinków dost#pnych w wybranym sezonie serialu.

1.13. Przegl#danie sezonów

ID: REQ003.REQ004.REQ005
Jako u#ytkownik chc# móc przegl#da# list# sezonów dost#pnych dla danego serialu.

1.14. Rejestracja

ID: REQ001
Jako u#ytkowink chc# móc zarejestrowa# si# w systemie.

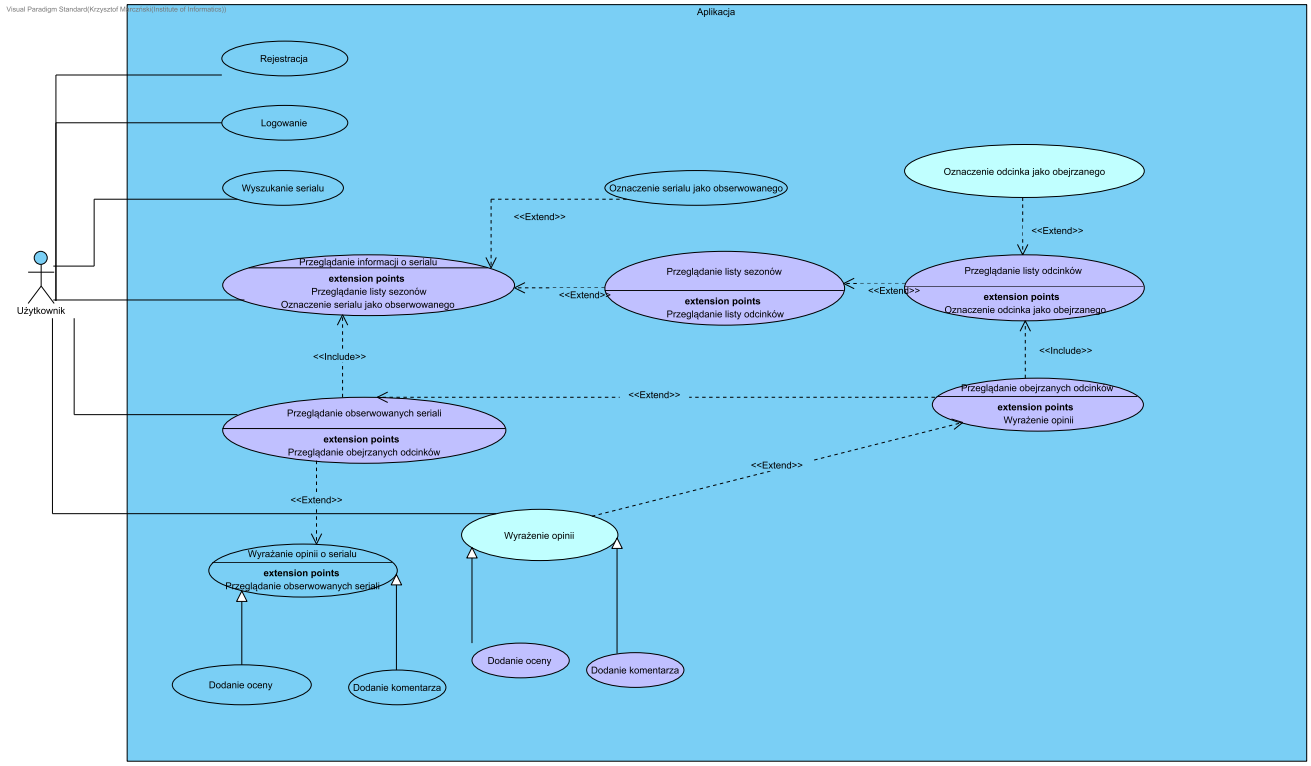
1.15. Struktura danych o serialach

1.16. Wyszukiwanie serialu

ID: REQ003
Jako u#ytkownik chc# móc wyszukiwa# seriale.

(4)
Diagram przypadków użycia

1. Diagram przypadkow uzycia



1.1. Aplikacja

1.2. Dodanie komentarza

ID: UC11

1.3. Dodanie oceny

ID: UC10

1.4. Logowanie

ID: UC12

U#ytkownik loguje si# do systemu, #eby system wy#wietla# jego dane. W tym celu wpisuje login i has#o.

1.5. Oznaczenie odcinka jako obejrzanego

ID: UC14

U#ytkownik mo#e oznaczy# odcinek jako obejrzany.

1.6. Oznaczenie serialu jako obserwowanego

ID: UC02

U#ytkownik oznacza serial jako obserwowany, to znaczy b#dzie otrzymywa# powiadomienia o nowo#ciach z nim zwi#zanych.

1.7. Przegl#danie informacji o serialu

ID: UC07

U#ytkownik mo#e przegl#da# informacje o wyszukanych lub obserwowanych przez niego serialach. Musi wykona# ten krok, #eby mie# dost#p do innych funkcji takich jak zaznaczenie odcinka jako obejrzanego.

1.8. Przegl#danie listy odcinków

ID: UC09

U#ytkownikowi wy#wietlana jest lista odcinków dla danego sezonu.

1.9. Przegl#danie listy sezonów

ID: UC08

U#ytkownikowi pokazywana jest lista sezonów dla danego serialu.

1.10. Przegl#danie obejrzanych odcinków

ID: UC16

U#ytkownik mo#e przegl#da# obejrzane odcinki, w formie listy odcinków z zaznaczeniem tych, które ju# obejrza#.

1.11. Przegl#danie obserwowanych seriali

ID: UC15

U#ytkownik mo#e przegl#da# obserwowane seriale. Ma dost#p równie# do zbiorowych statystyk takich jak suma ogl#danych odcinków.

1.12. Rejestracja

ID: UC13

U#ytkownik musi si# zarejstrowa#, #eby korzysta# z serwisu. Wymy#la sobie login i has#o, oraz musi zaakceptowa# regulamin serwisu.

1.13. U#ytkownik

ID: AC01

1.14. Wyra#anie opinii o serialu

ID: UC18

1.15. Wyra#enie opinii

ID: UC03

U#ytkownik mo#e wyrazi# opini# o serialu w formie komentarza lub oceny. Ocena jest wyliczeniem.

1.16. Wyszukanie serialu

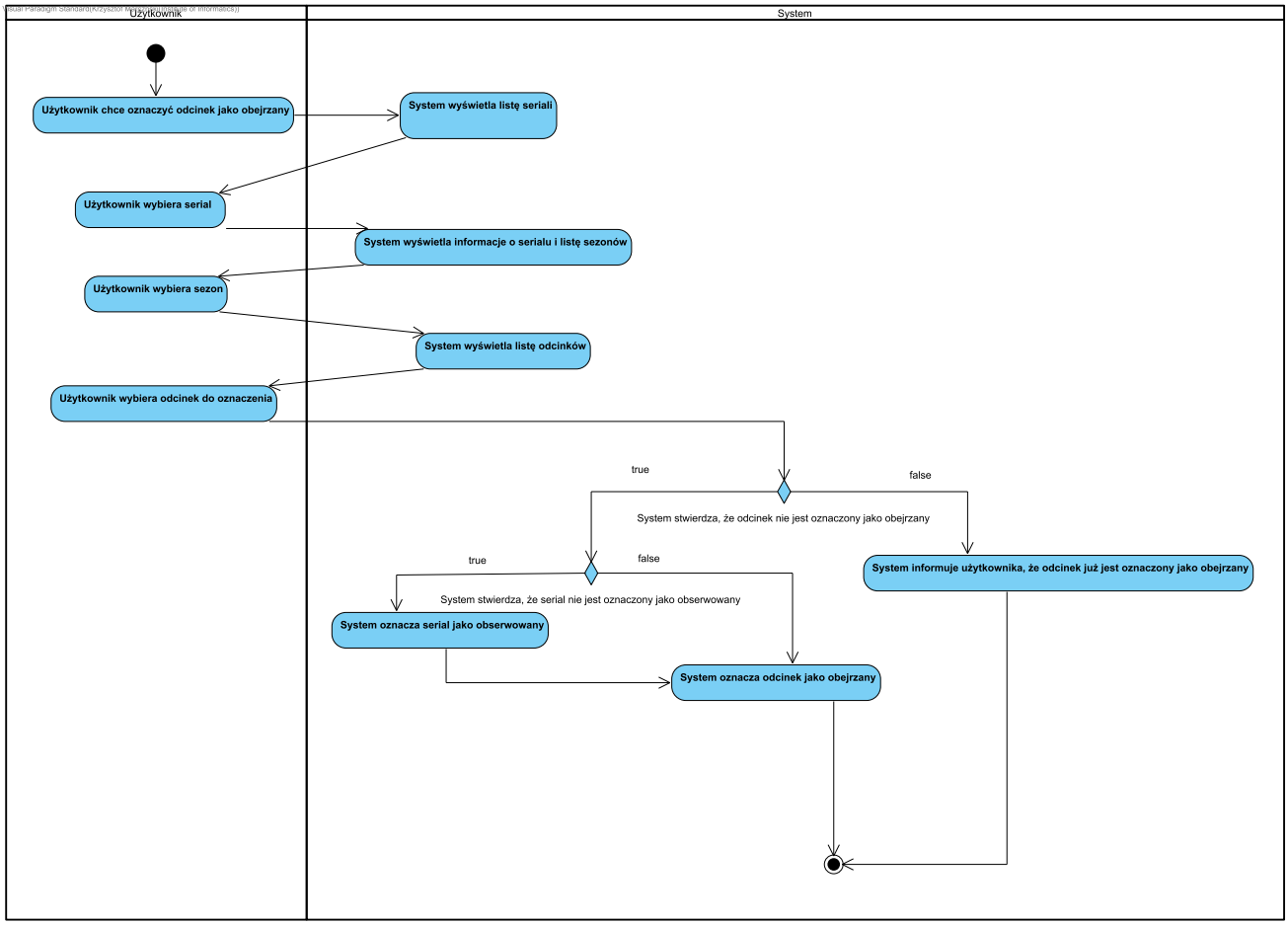
ID: UC01

U#ytkownik mo#e wyszuka# serial po nazwie i roku.

(5)

Specyfikacja przypadków użycia. Model informacyjny

1. Oznaczenie odcinka jako obejrzanego v2



***** Oznaczanie odcinka jako obejrzanego ver 2*****
1. U#ytkownik chce oznaczy# odcinek jako obejrzany
2. System wy#wietla list# seriali
3. U#ytkownik wybiera serial
4. System wy#wietla informacje o serialu i list# sezonów
5. U#ytkownik wybiera sezon
6. System wy#wietla list# odcinków
7. U#ytkownik wybiera odcinek do oznaczenia
8. if (System stwierdza, #e odcinek nie jest oznaczony jako obejrzany)
    8.1. if (System stwierdza, #e serial nie jest oznaczony jako obserwowany)
        8.1.1. System oznacza serial jako obserwowany
    8.2 System oznacza odcinek jako obejrzany
9. else
    9.1 System informuje u#ytkownika, #e odcinek ju# jest oznaczony jako obejrzany

1.1. Swimlane

1.2. System informuje u#ytkownika, #e odcinek ju# jest oznaczony jako obejrzany

1.3. System oznacza odcinek jako obejrzany

1.4. System oznacza serial jako obserwowany

1.5. System stwierdza, #e odcinek nie jest oznaczony jako obejrzany

1.6. System stwierdza, #e serial nie jest oznaczony jako obserwowany

1.7. System wy#wietla informacje o serialu i list# sezonów

1.8. System wy#wietla list# odcinków

1.9. System wy#wietla list# seriali

1.10. unnamed

1.11. unnamed

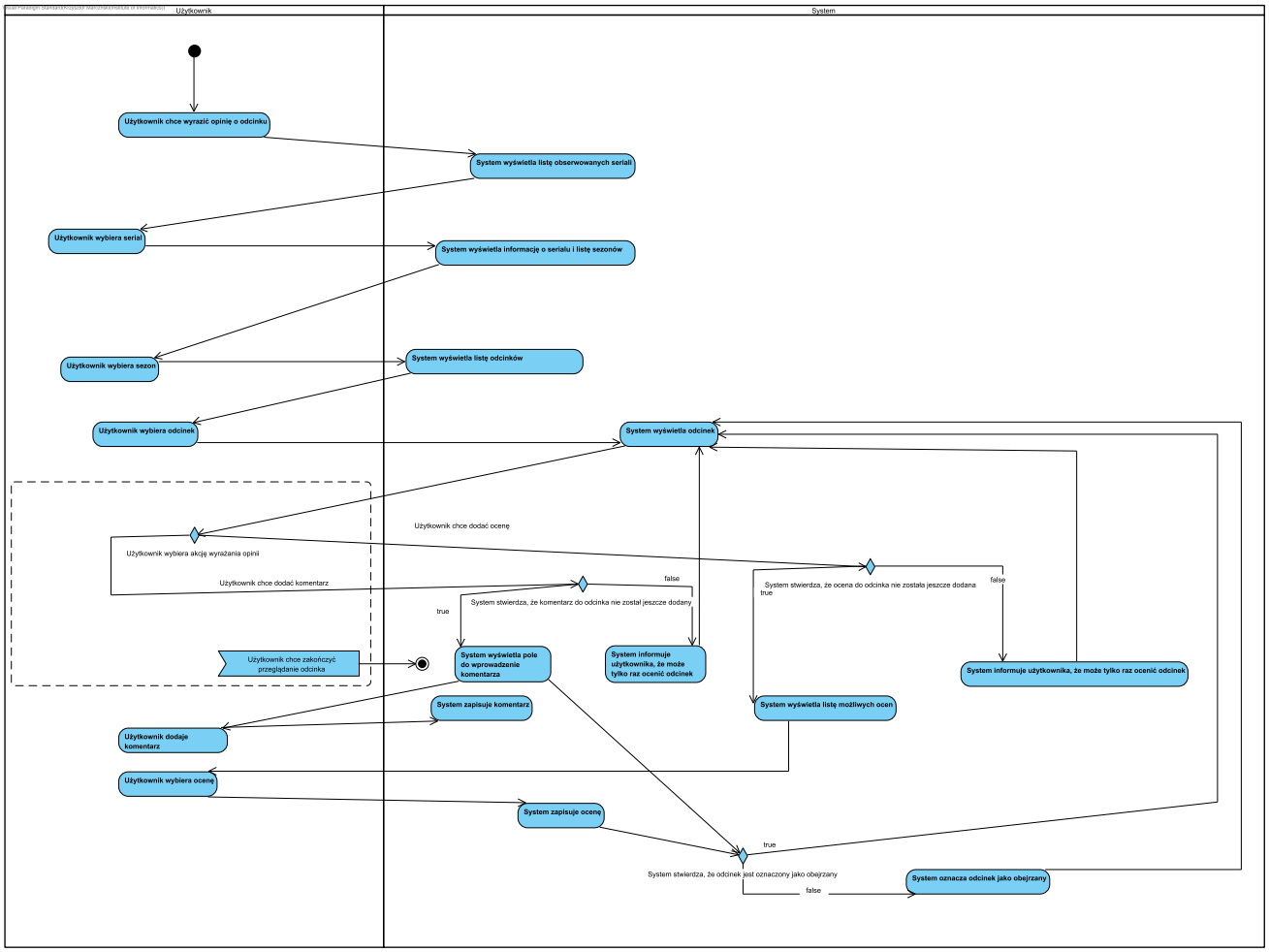
1.12. U#ytkownik chce oznaczy# odcinek jako obejrzany

1.13. U#ytkownik wybiera odcinek do oznaczenia

1.14. U#ytkownik wybiera serial

1.15. U#ytkownik wybiera sezon

1. Wyra#anie opinii o odcinku v2



***** Wyra#anie opinii ver 2*****

1. U#ytkownik chce wyrazi# opini#
2. System wy#wietla list# obserwowanych seriali
3. U#ytkownik wybiera serial
4. System wy#wietla informacj# o serialu i list# sezonów
5. U#ytkownik wybiera sezon
6. System wy#wietla list# obserwowanych odcinków
7. U#ytkownik wybiera odcinek
8. System wy#wietla odcinek
9. switch (U#ytkownik wybiera akcj# wyra#ania opinii)
    - 9.1 case (U#ytkownik chce doda# opini#)
        - 9.1.1 if (System stwierdza, #e ocena do odcinka nie zosta#a jeszcze dodana)
            - 9.1.1.1 System wy#wietla list# mo#liwych ocen
            - 9.1.1.2 U#ytkownik wybiera ocen#
            - 9.1.1.3 System zapisuje ocen#
        - 9.1.2 else
            - 9.1.2.1 System informuje u#ytkownika, #e mo#e tylko raz oceni# odcinek
    - 9.2 case (U#ytkownik chce doda# komentarz)
        - 9.2.1 if (System stwierdza, #e komentarz do odcinka nie zosta# jeszcze dodany)
            - 9.2.1.1 System wy#wietla pole do wprowadzenia komentarza
            - 9.2.1.2 U#ytkownik wpisuje komentarz
            - 9.2.1.3 System zapisuje komentarz
        - 9.2.2 else

9.2.2.1 System informuje u#ytkownika, #e mo#e tylko raz skomentowa# odcinek
10. Przejd# do punktu 8)

Rozszerzenia:
9a) U#ytkownik chce zako#czy# przegl#danie odcinka
KONIEC SCENARIUSZA

1.1. Region

1.2. Swimlane2

1.3. System informuje u#ytkownika, #e mo#e tylko raz oceni# odcinek

1.4. System oznacza odcinek jako obejrzany

1.5. System stwierdza, #e komentarz do odcinka nie zosta# jeszcze dodany

1.6. System stwierdza, #e ocena do odcinka nie zosta#a jeszcze dodana

1.7. System stwierdza, #e odcinek jest oznaczony jako obejrzany

1.8. System wy#wietla informacj# o serialu i list# sezonów

1.9. System wy#wietla list# mo#liwych ocen

1.10. System wy#wietla list# obserwowanych seriali

1.11. System wy#wietla list# odcinków

1.12. System wy#wietla odcinek

1.13. System wy#wietla pole do wprowadzenie komentarza

1.14. System zapisuje komentarz

1.15. System zapisuje ocen#

1.16. unnamed

1.17. unnamed

1.18. U#ytkownik chce wyrazi# opini# o odcinku

1.19. U#ytkownik chce zako#czy# przegl#danie odcinka

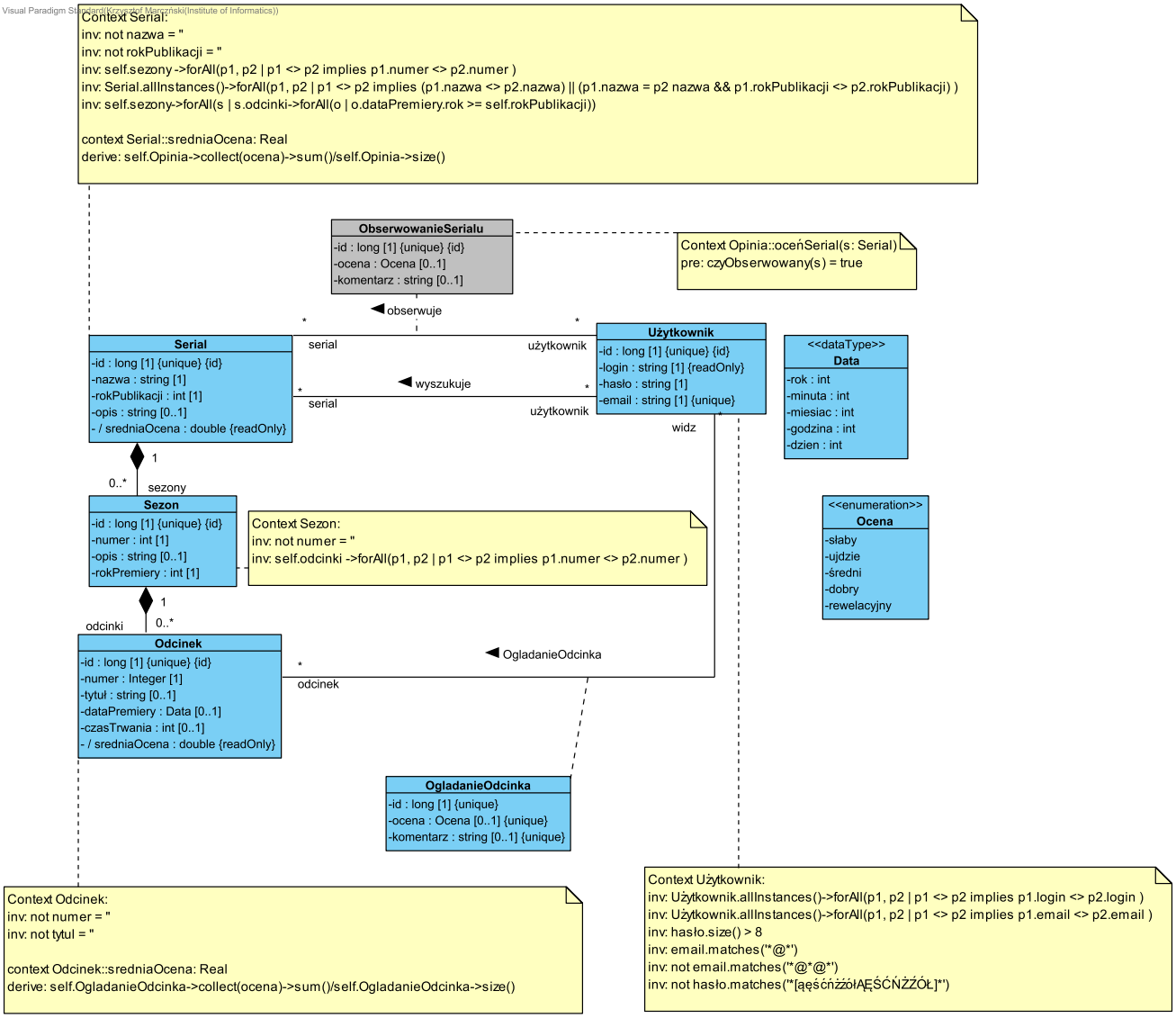
1.20. U#ytkownik dodaje komentarz

1.21. U#ytkownik wybiera akcj# wyra#ania opinii

1.22. U#ytkownik wybiera ocen#

- 1.23. U#ytkownik wybiera odcinek
- 1.24. U#ytkownik wybiera serial
- 1.25. U#ytkownik wybiera sezon

1. Model informacyjny



1.1. Data

1.2. ObserwowanieSerialu

1.3. Ocena

1.4. Odcinek

1.5. OgladanieOdcinka

1.6. Serial

1.7. Sezon

1.8. U#ytkownik

OCL

Context Użytkownik:

inv: Użytkownik.allInstances()->forAll(p1, p2 | p1 <> p2 implies p1.login <> p2.login)

inv: Użytkownik.allInstances()->forAll(p1, p2 | p1 <> p2 implies p1.email <> p2.email)

inv: size(hasło) > 8

inv: email.matches('*@*')

inv: not email.matches('*@*@*')

inv: not hasło.matches('*[ąęśćńżźółĄĘŚĆŃŻŹÓŁ]*')

Context Serial:

inv: not nazwa = ''

inv: not rokPublikacji = ''

inv: self.sezony ->forAll(p1, p2 | p1 <> p2 implies p1.numer <> p2.numer)

inv: Serial.allInstances()->forAll(p1, p2 | p1 <> p2 implies (p1.nazwa <> p2.nazwa) || (p1.nazwa = p2 nazwa && p1.rokPublikacji <> p2.rokPublikacji))

inv: self.sezony->forAll(s | s.odcinki->forAll(o | o.dataPremiery.rok >= rokPublikacji))

Context Serial::sredniaOcena: Real

derive: Opinia.allInstances()->collect(ocena)->sum()/size()

Context Sezon:

inv: not numer = ''

inv: self.odcinki ->forAll(p1, p2 | p1 <> p2 implies p1.numer <> p2.numer)

Context Odcinek:

inv: not numer = ''

inv: not tytul = ''

Context Odcinek::sredniaOcena: Real

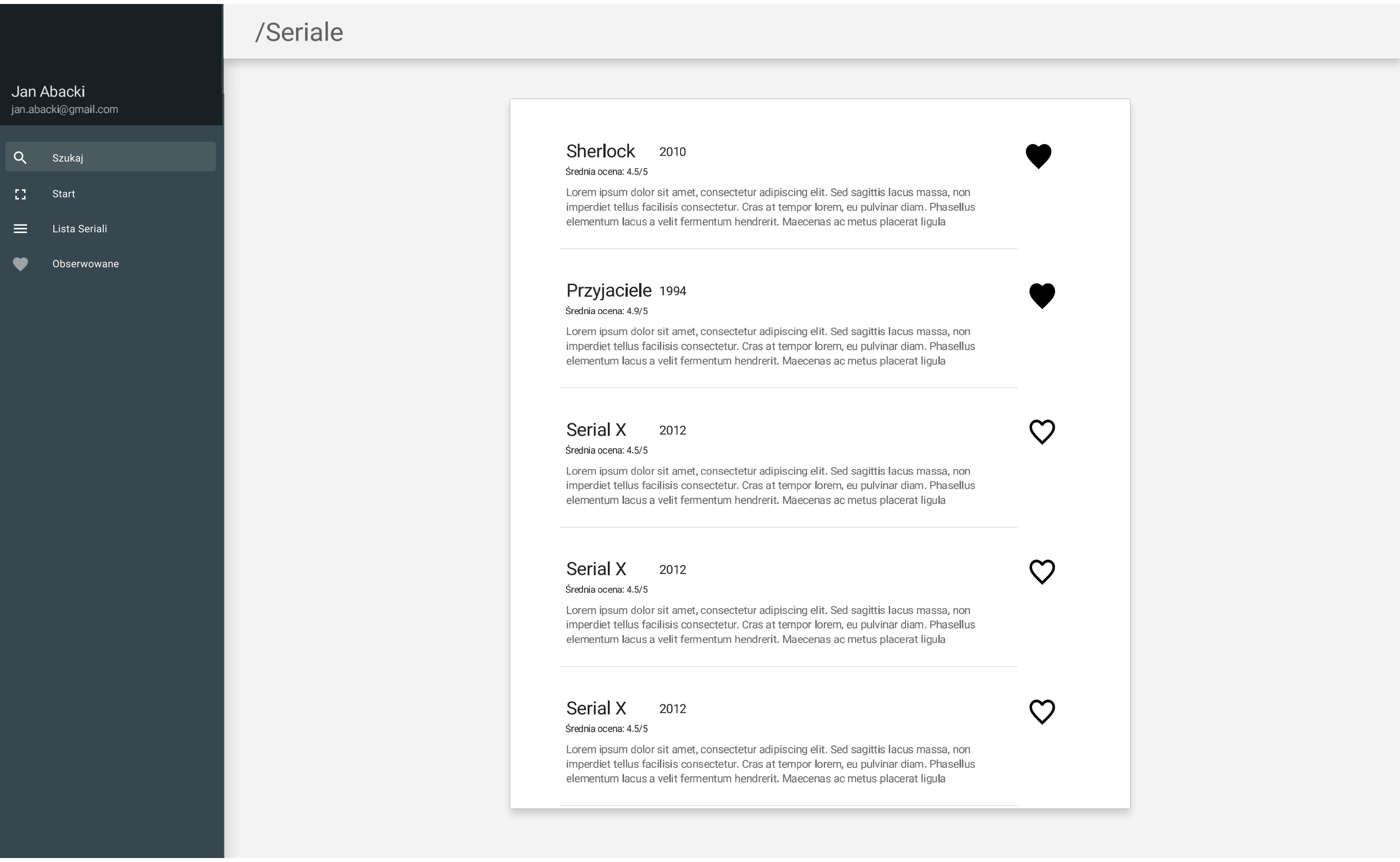
derive: OgladanieOdcinka.allInstances()->collect(ocena)->sum()/size()

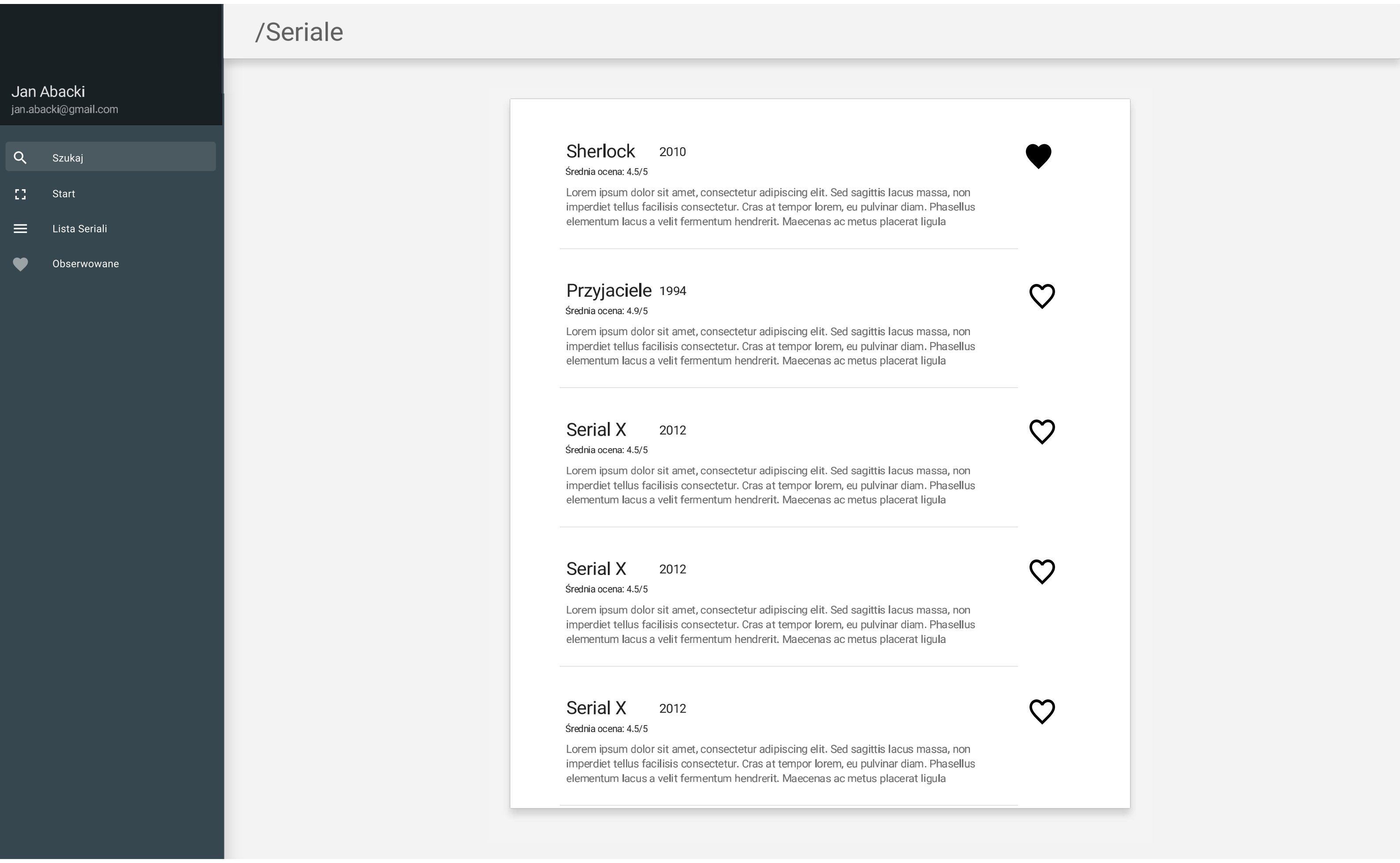
Context Opinia::oceńSerial(s: Serial)

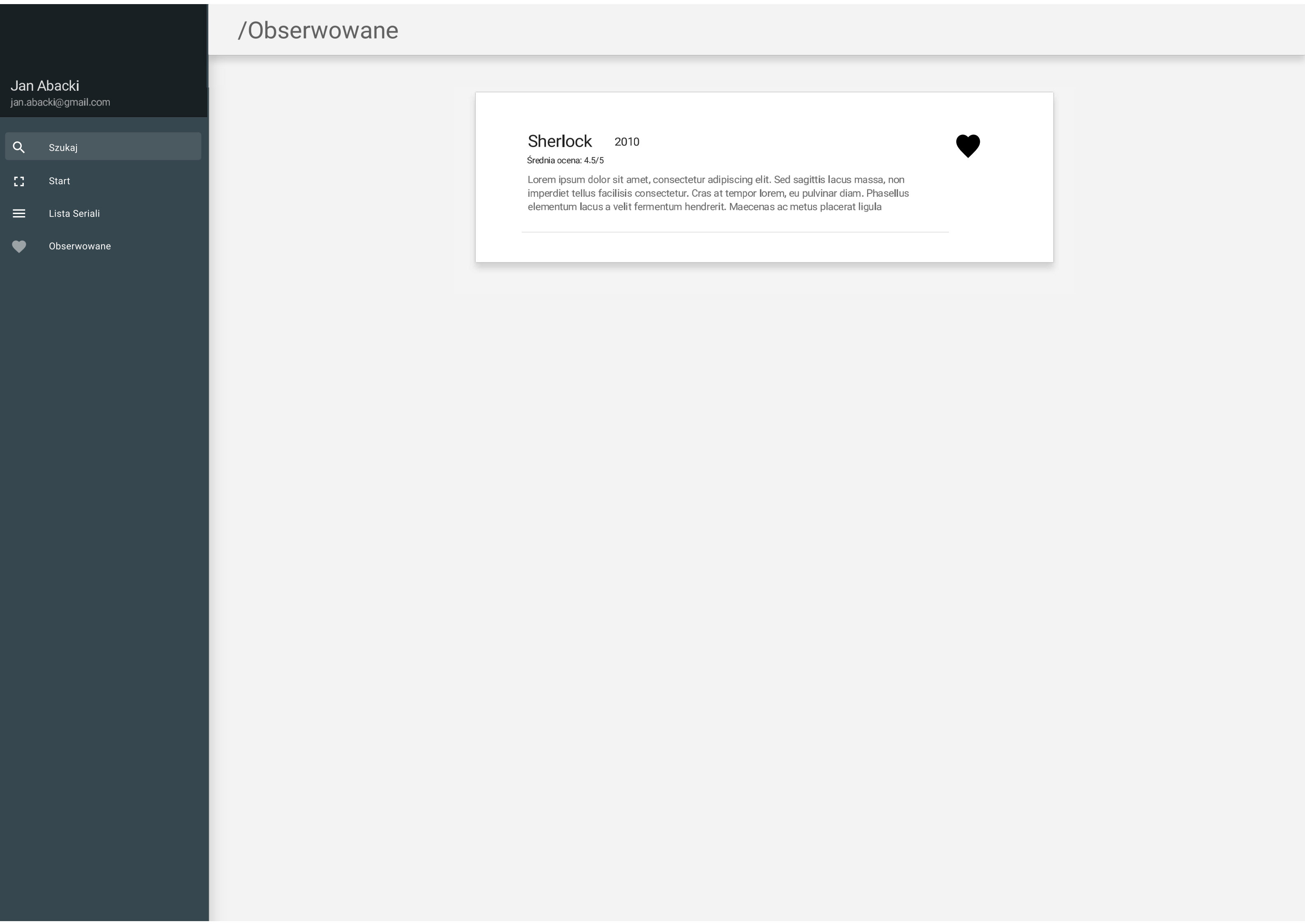
pre: czyObserwowany(s) = true

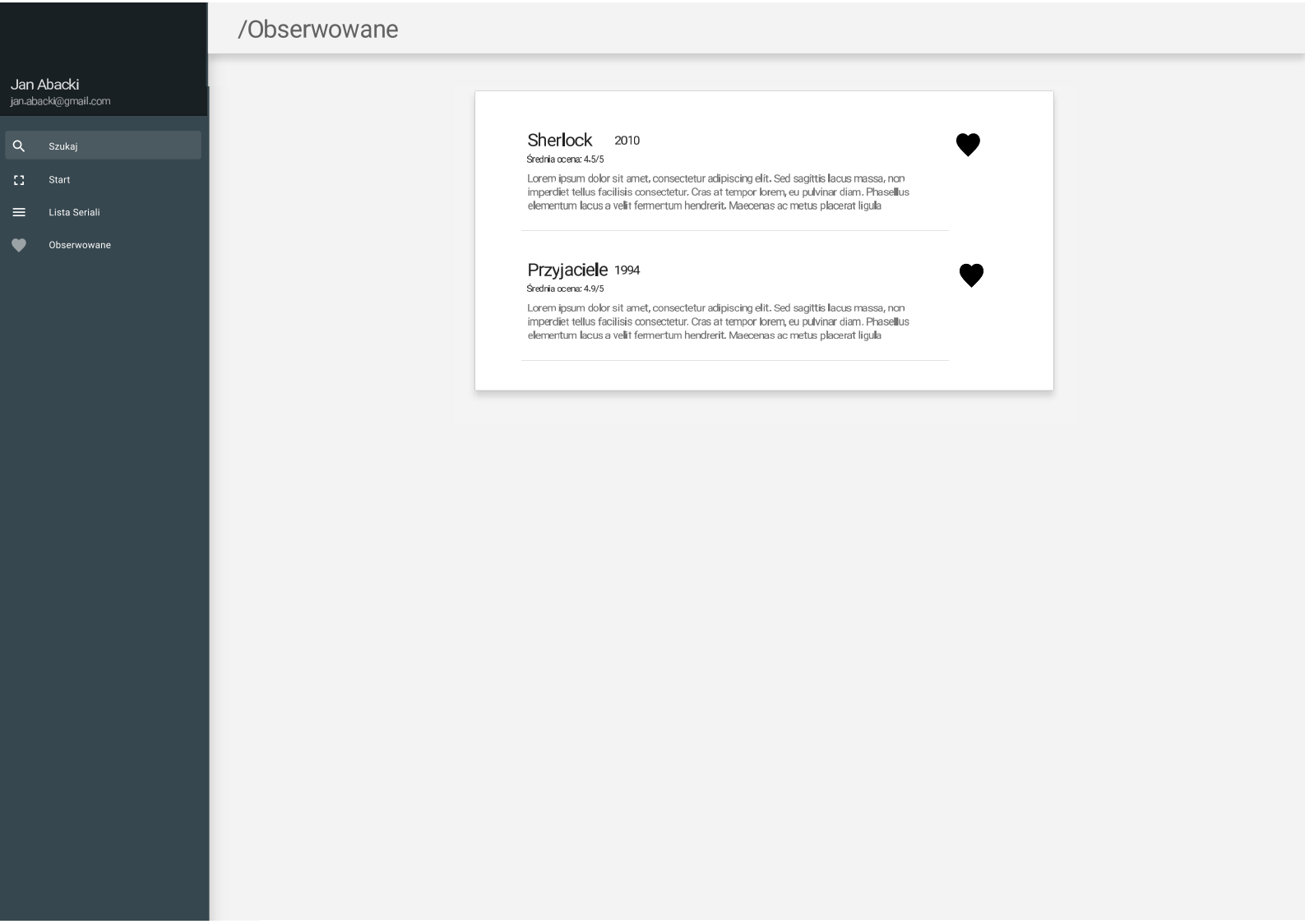
(6)
Prototyp interfejsu

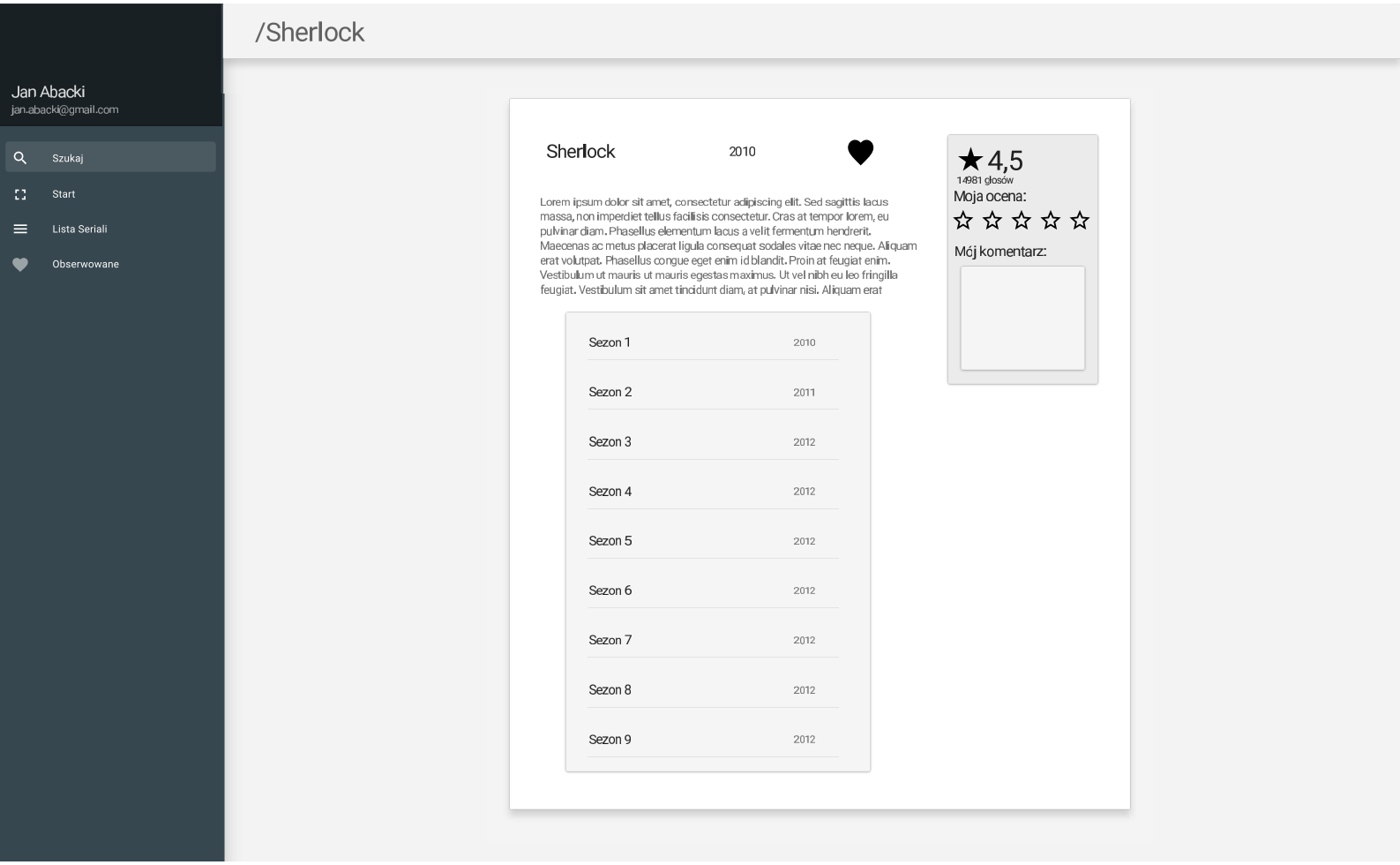

Start
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Witaj, Jan!
Co ciekawego
ostatnio oglądałeś???

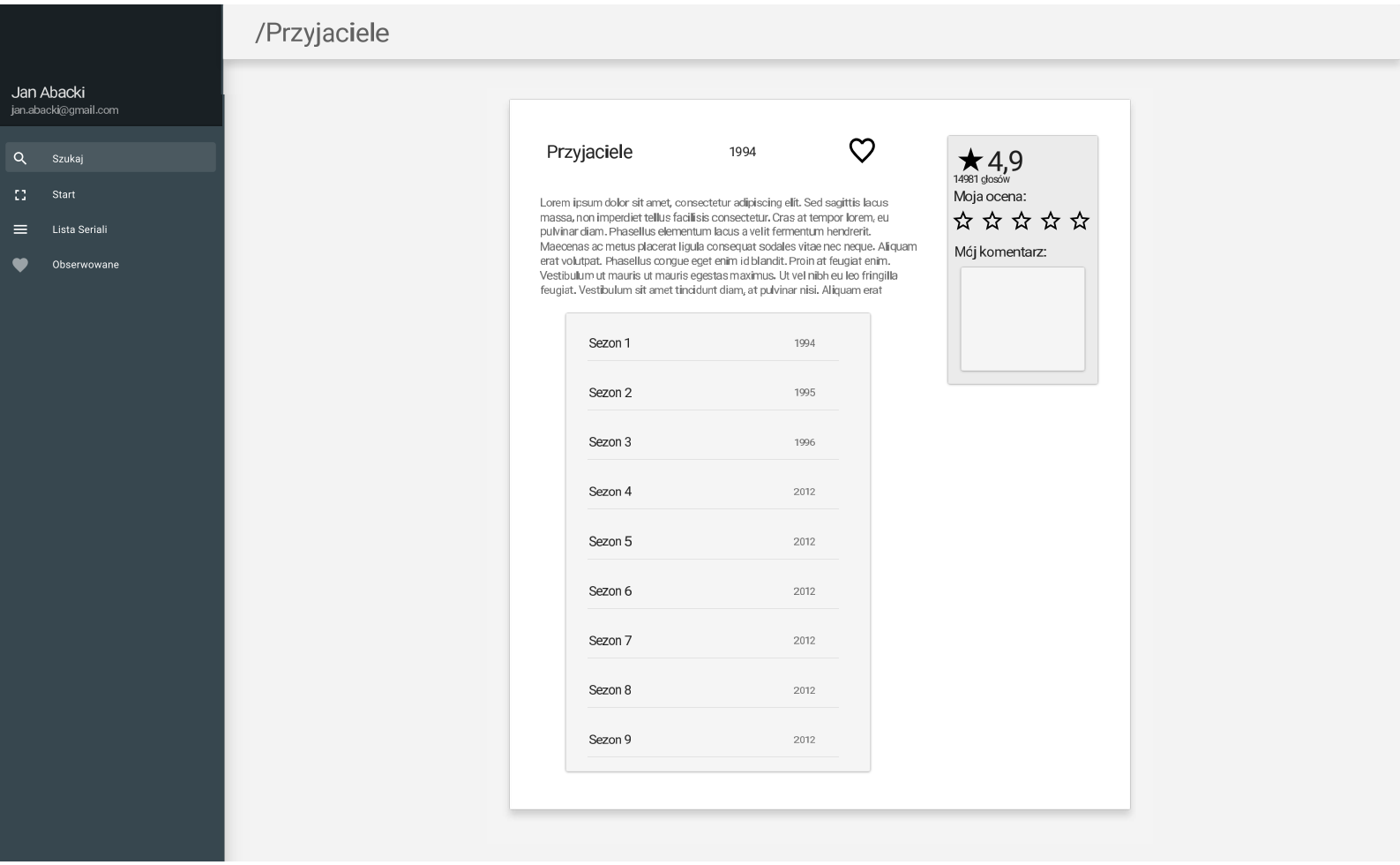

/Seriale
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock 2010
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Przyjaciele 1994
Średnia ocena: 4.9/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula

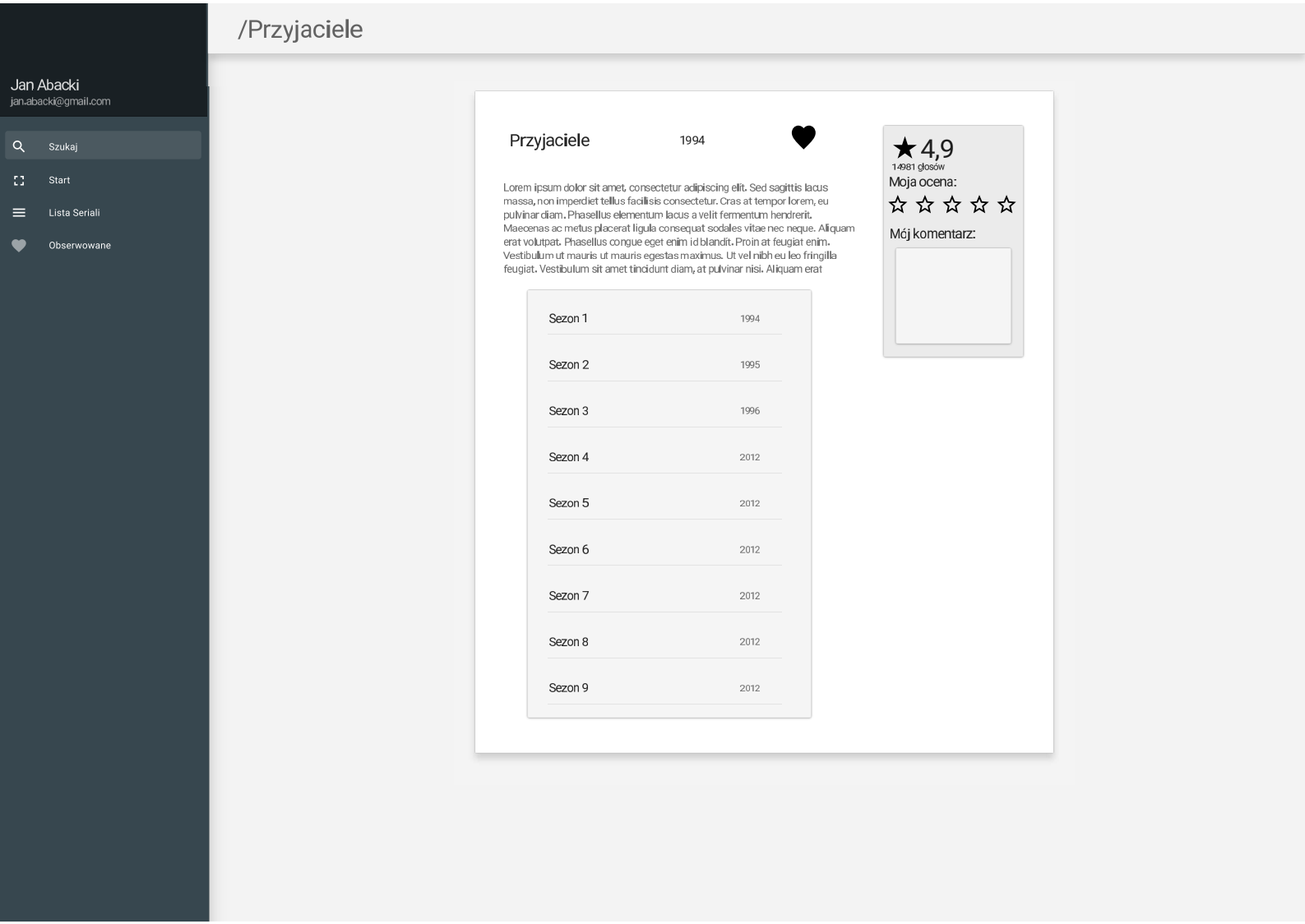

/Seriale
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock 2010
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Przyjaciele 1994
Średnia ocena: 4.9/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Serial X 2012
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula

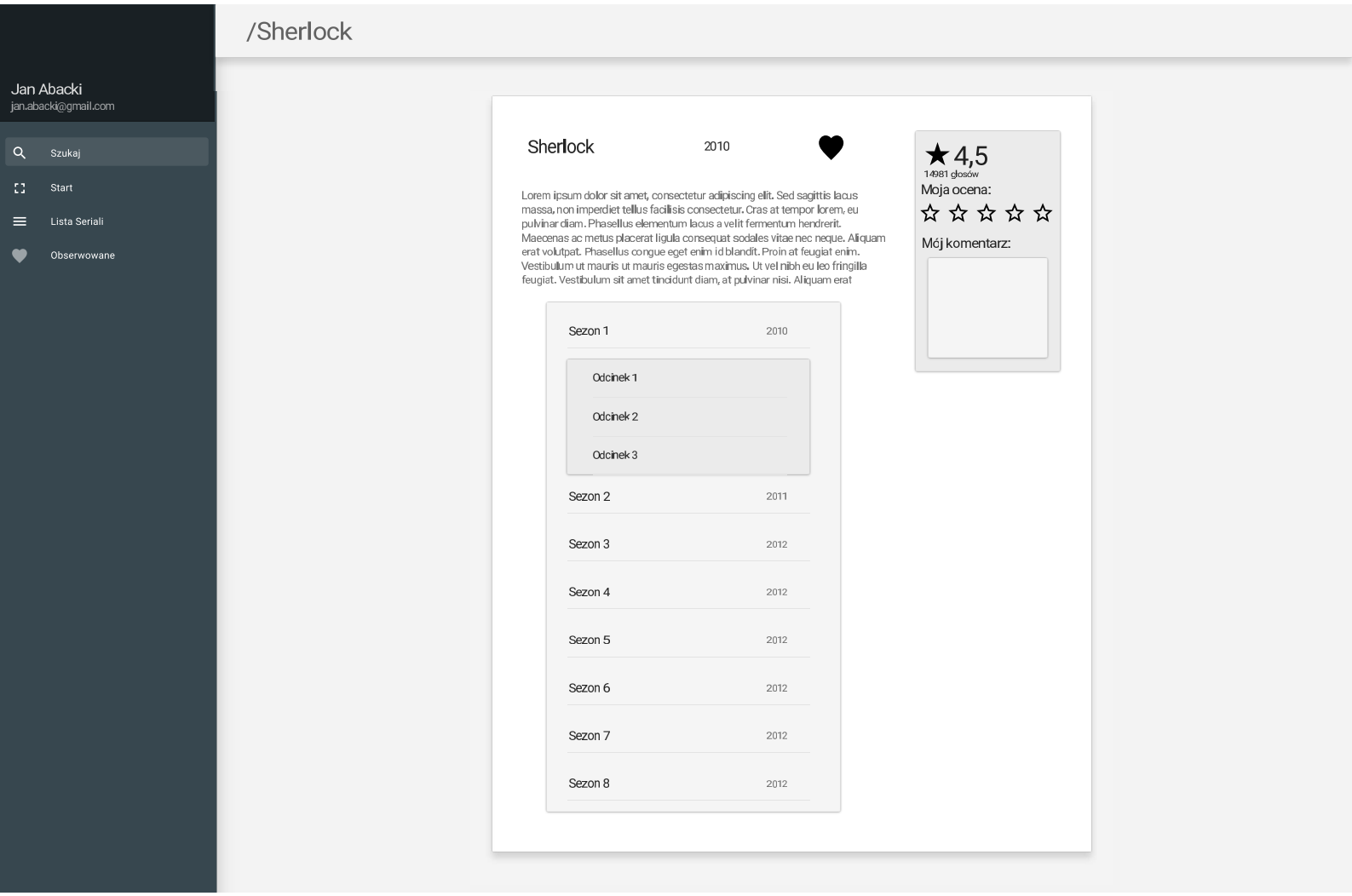

/Obserwowane
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock 2010
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula

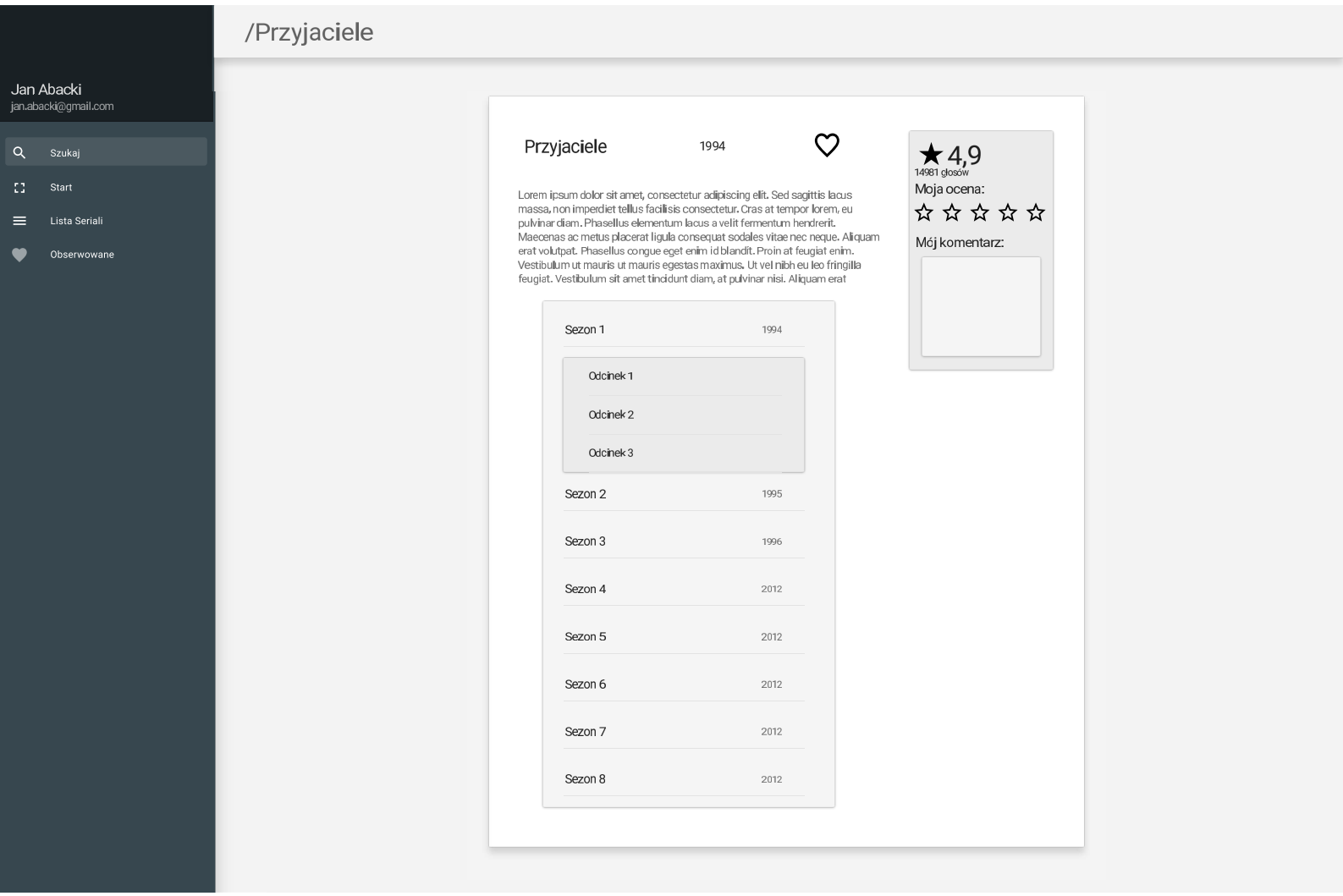

/Obserwowane
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock 2010
Średnia ocena: 4.5/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula
Przyjaciele 1994
Średnia ocena: 4.9/5
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula

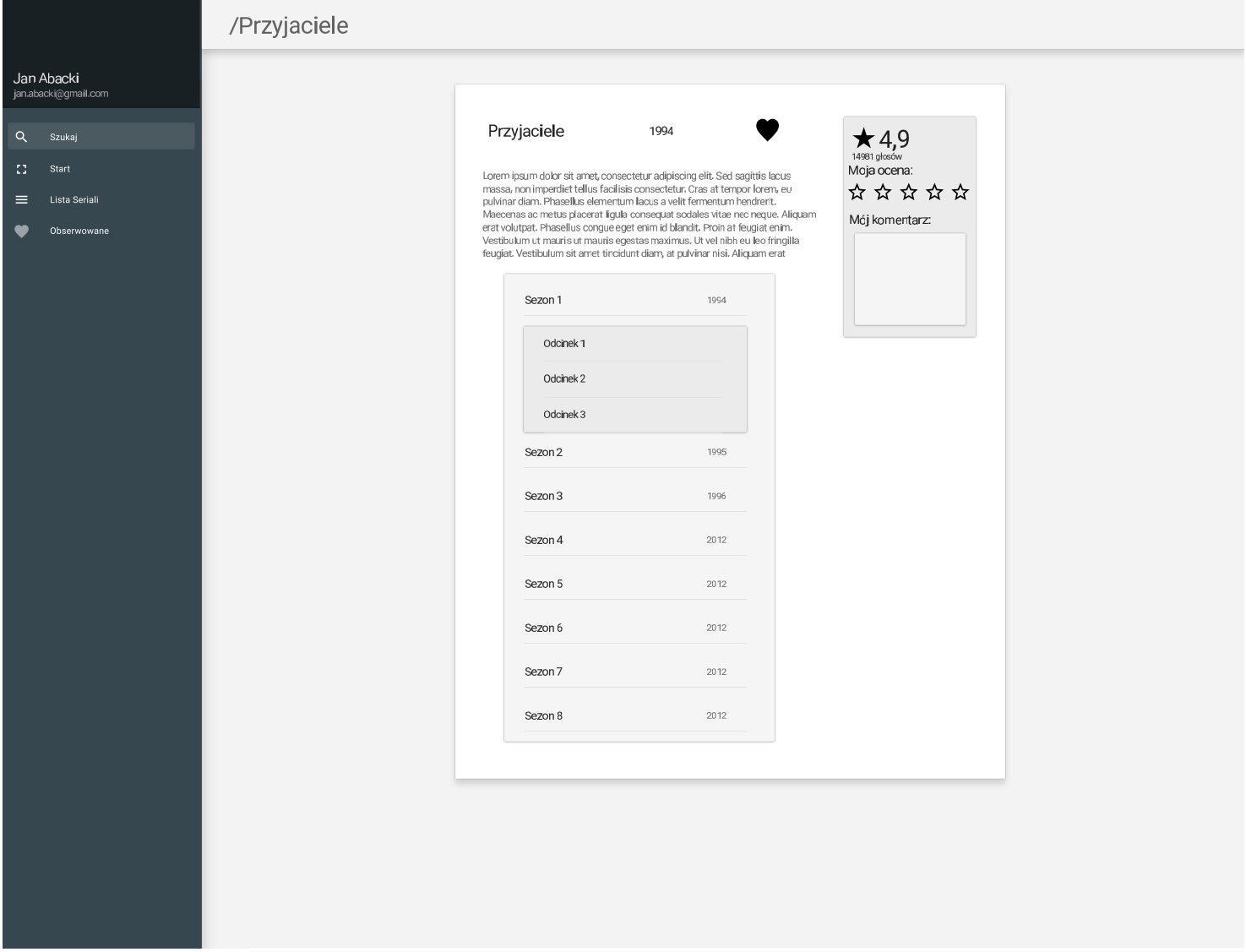

/Sherlock
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock 2010
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 2010
Sezon 2 2011
Sezon 3 2012
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
Sezon 9 2012
★4,5
14981 głosów
Moja ocena:
Mój komentarz:

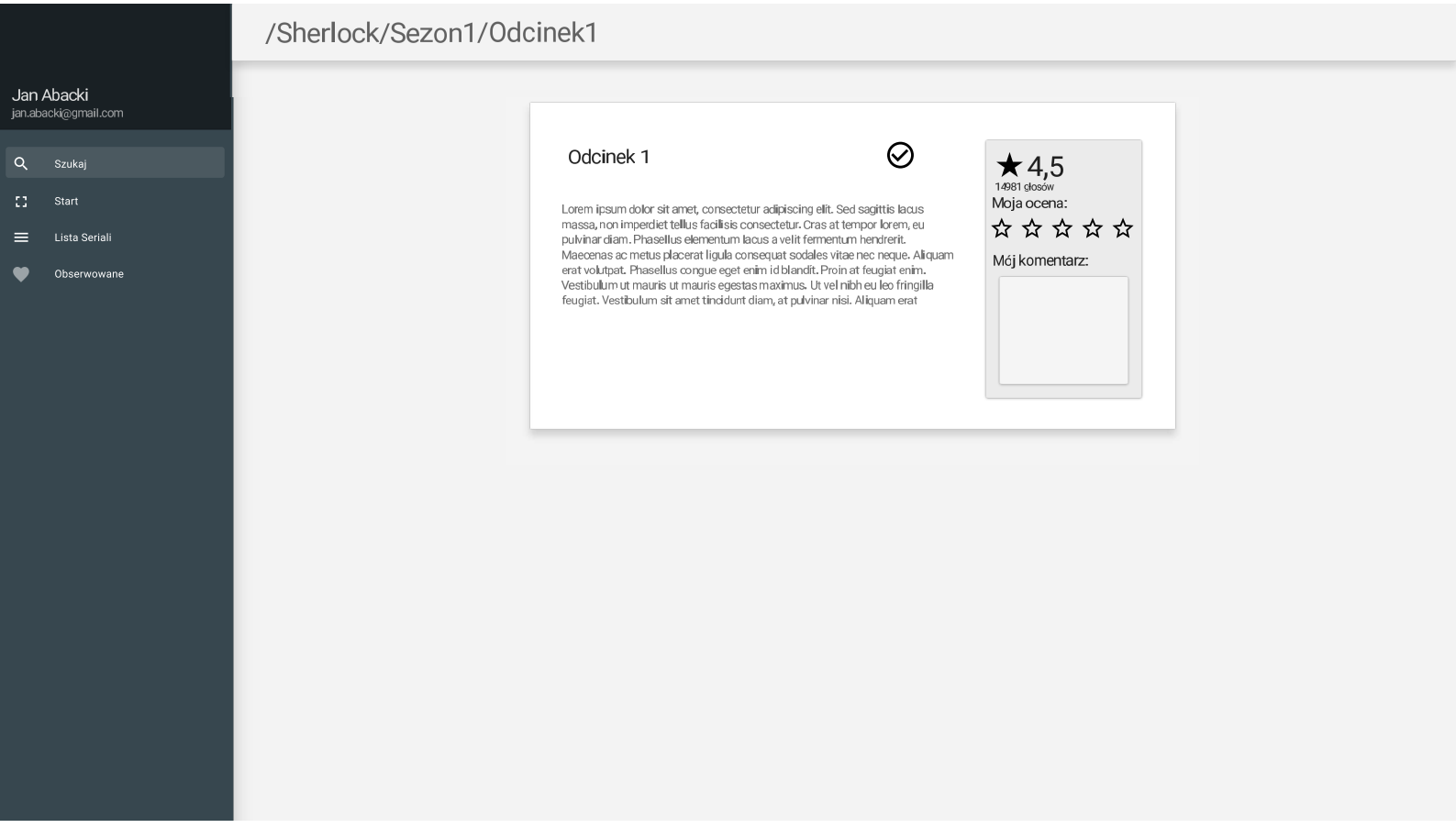

/Przyjaciele
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Przyjaciele
1994
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 1994
Sezon 2 1995
Sezon 3 1996
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
Sezon 9 2012
★4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Przyjaciele
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Przyjaciele
1994
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 1994
Sezon 2 1995
Sezon 3 1996
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
Sezon 9 2012
★4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Sherlock
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Sherlock
2010
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 2010
Odcinek 1
Odcinek 2
Odcinek 3
Sezon 2 2011
Sezon 3 2012
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
★4,5
14981 głosów
Moja ocena:
Mój komentarz:

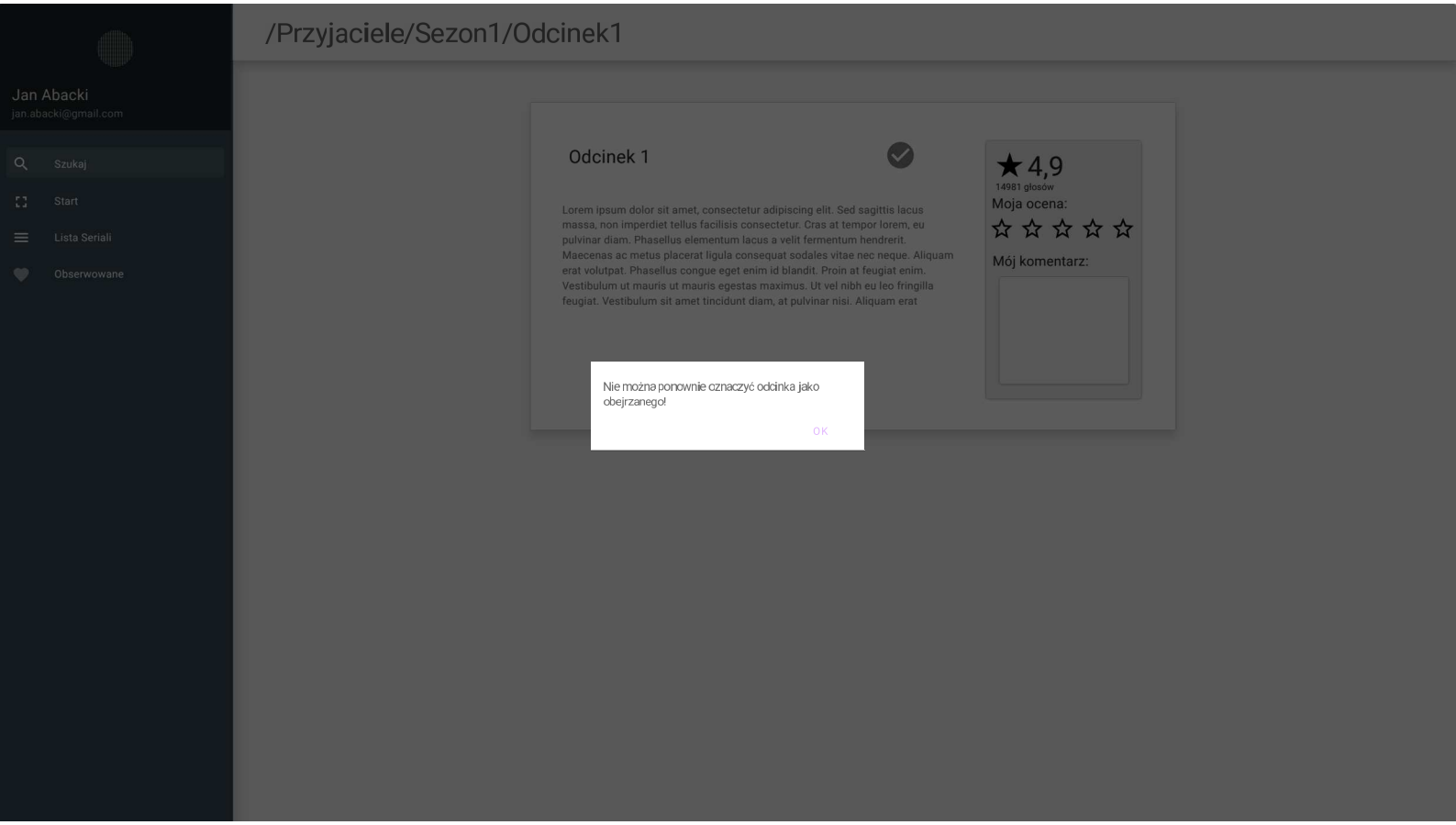

/Przyjaciele
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Przyjaciele
1994
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 1994
Odcinek 1
Odcinek 2
Odcinek 3
Sezon 2 1995
Sezon 3 1996
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
★4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Przyjaciele
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Przyjaciele
1994
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
Sezon 1 1994
Odcinek 1
Odcinek 2
Odcinek 3
Sezon 2 1995
Sezon 3 1996
Sezon 4 2012
Sezon 5 2012
Sezon 6 2012
Sezon 7 2012
Sezon 8 2012
★4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Sherlock/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
★4,5
14981 głosów
Moja ocena:
Mój komentarz:

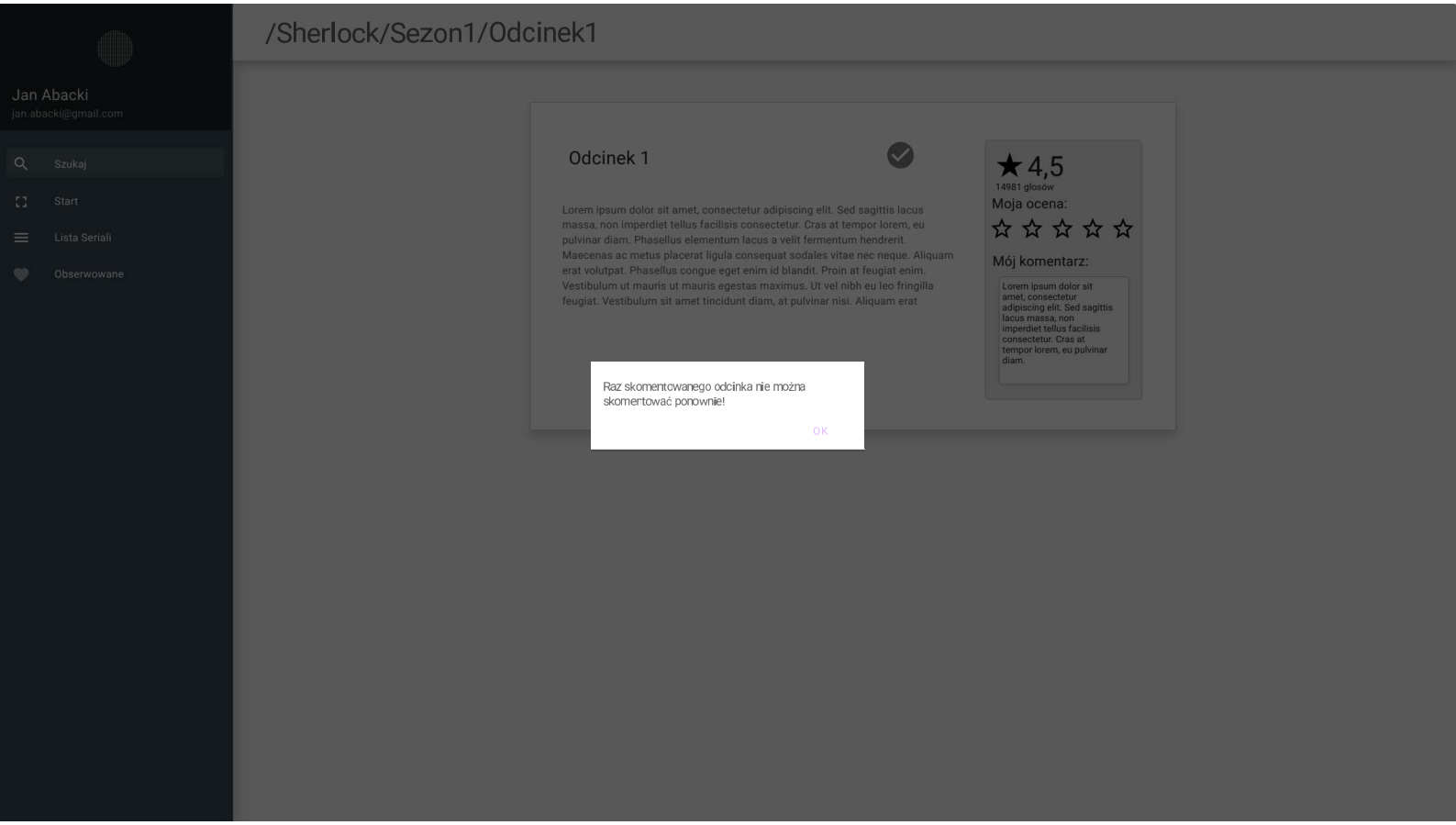

/Przyjaciele/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
★4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Przyjaciele/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
★4,9
14981 głosów
Moja ocena:
Mój komentarz:

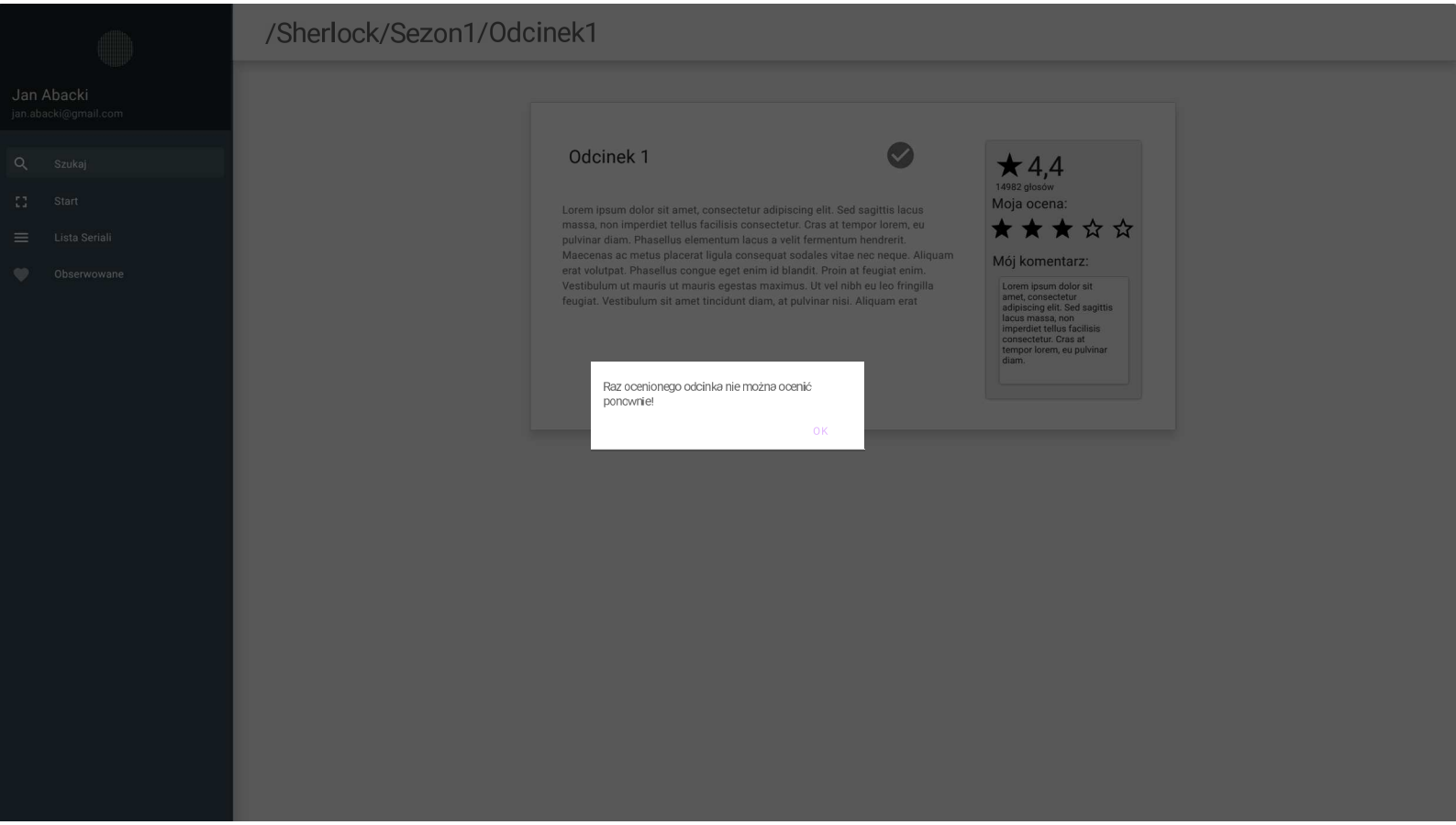

/Przyjaciele/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
4,9
14981 głosów
Moja ocena:
Mój komentarz:
Nie można ponownie oznaczyć odcinka jako obejrzanego!
OK


/Przyjaciele/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
4,9
14981 głosów
Moja ocena:
Mój komentarz:


/Sherlock/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
★4,5
14981 głosów
Moja ocena:
Mój komentarz:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam.


/Sherlock/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
★4,5
14981 głosów
Moja ocena:
Mój komentarz:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam.
Raz skomentowanego odcinka nie można skomertować ponownie!
OK


/Sherlock/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
4,4
14982 głosów
Moja ocena:
Mój komentarz:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam.

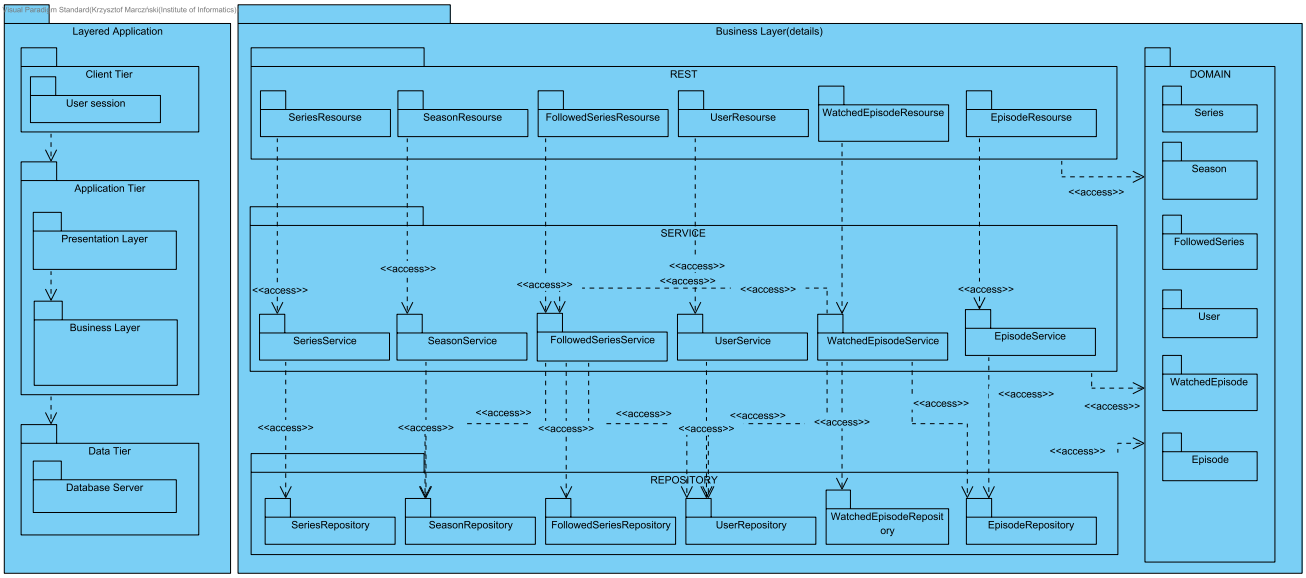

/Sherlock/Sezon1/Odcinek1
Jan Abacki
jan.abacki@gmail.com
Szukaj
Start
Lista Seriali
Obserwowane
Odcinek 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam. Phasellus elementum lacus a velit fermentum hendrerit. Maecenas ac metus placerat ligula consequat sodales vitae nec neque. Aliquam erat volutpat. Phasellus congue eget enim id blandit. Proin at feugiat enim. Vestibulum ut mauris ut mauris egestas maximus. Ut vel nibh eu leo fringilla feugiat. Vestibulum sit amet tincidunt diam, at pulvinar nisi. Aliquam erat
4,4
14982 głosów
Moja ocena:
Mój komentarz:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sagittis lacus massa, non imperdiet tellus facilisis consectetur. Cras at tempor lorem, eu pulvinar diam.
Raz ocenionego odcinka nie można ocenić ponownie!
OK

(7)

Architektura systemu.

Projekt bazy danych

1. Diagram pakietów - Layers



1.1. Application Tier

1.2. Business Layer

Logika biznesowa jest zrealizowana przez backend.

1.3. Business Layer(details)

1.4. Client Tier

1.5. Data Tier

1.6. Database Server

Serwer przechowuj#cy dane wprowadzano wcze#niej przez administratora i wysy#aj#cy dane do business layer na polecenie business layer.

1.7. DOMAIN

1.8. Episode

1.9. EpisodeRepository

1.10. EpisodeResourse

1.11. EpisodeService

1.12. FollowedSeries

1.13. FollowedSeriesRepository

1.14. FollowedSeriesResourse

1.15. FollowedSeriesService

1.16. Layered Application

1.17. Presentation Layer
Logika za rozmieszczeniem i wygl#dem elementów, przez taki "interfejs" users wykonuj# akcje application tier.

1.18. REPOSITORY

1.19. REST
Odpowiada

1.20. Season

1.21. SeasonRepository

1.22. SeasonResourse

1.23. SeasonService

1.24. Series

1.25. SeriesRepository

1.26. SeriesResourse

1.27. SeriesService

1.28. SERVICE

1.29. User

1.30. User session
U#ytkownicy to ludzie nale#acy do client tier. Przez internet maj# dost#p do presentation layer.

1.31. UserRepository

1.32. UserResourse

1.33. UserService

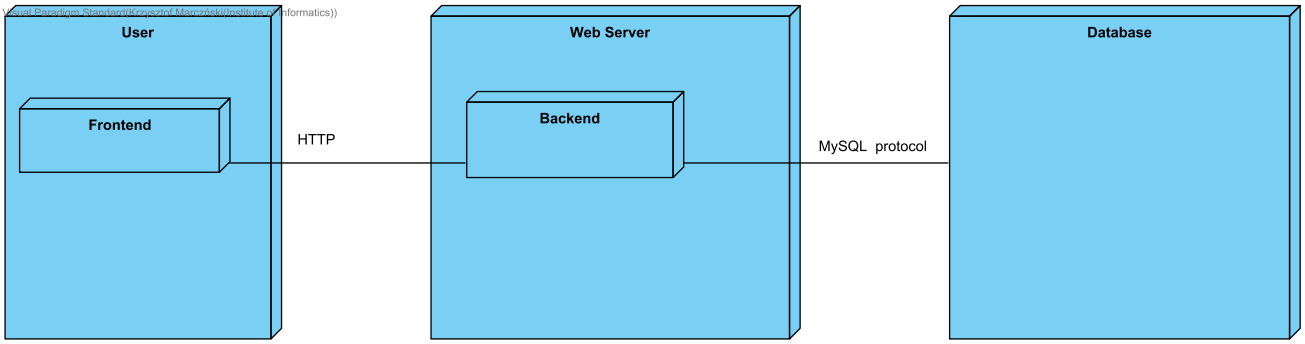
1.34. WatchedEpisode

1.35. WatchedEpisodeRepository

1.36. WatchedEpisodeResourse

1.37. WatchedEpisodeService

1. Architektura fizyczna



1.1. Backend

Logika aplikacji. Znajduje si# na web serwerze, w tej wersji na localhoscie w komputerze administratora.

1.2. Database

Dana bazy SQL wykonana w dialekcie mySQL, w tej wersji b#dzie znajdowa# si# na localhoscie na komputerze administratora.

1.3. Frontend

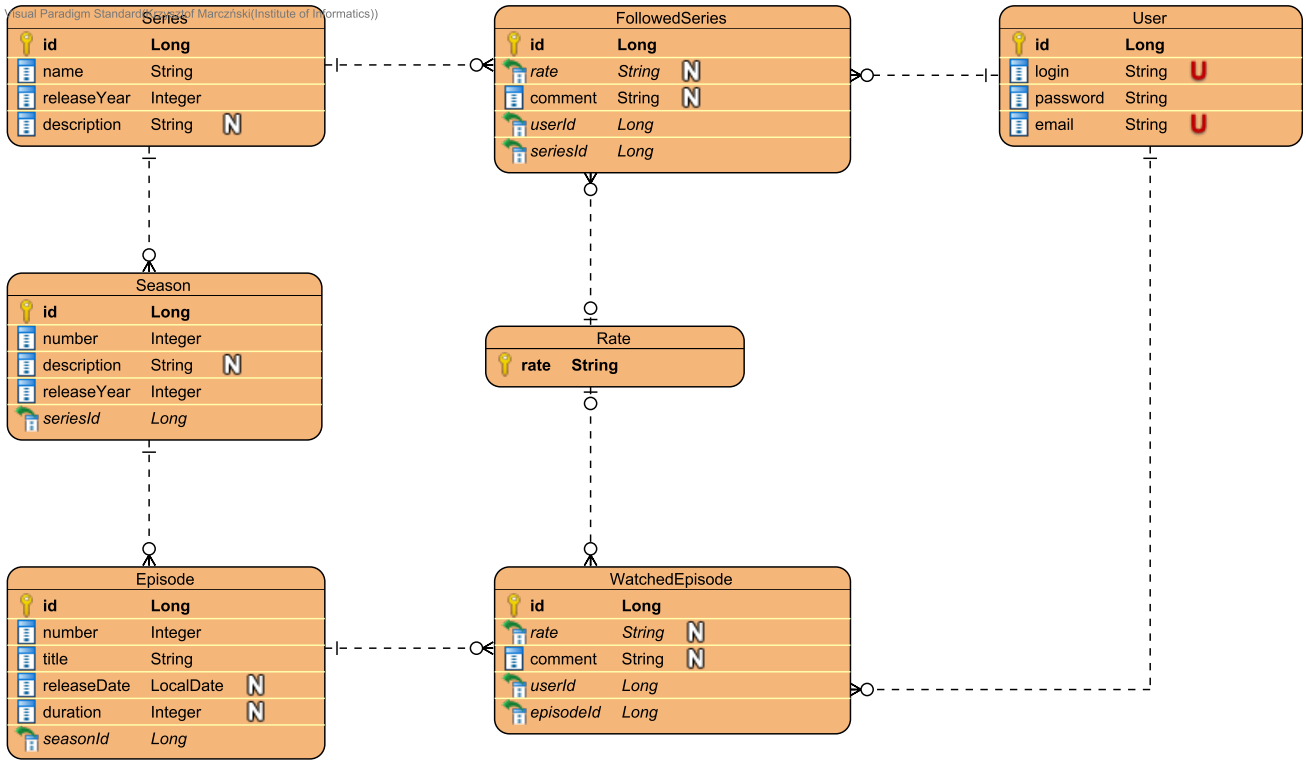
Cz### odpowiedzialna za wygl#d aplikacji. Znajduje si# na web serwerze, w tym wypadku w localho#cie na komputerze administratora.

1.4. User

U#ytkownik jest cz#owiekiem korzystaj#cym z aplikacji. Znajduje si# gdziekolwiek na #wiecie i korzysta z po##czenia internetowego przez HTPS.

1.5. Web Server

1. ERD from oracle



1.1. Episode

1.2. FollowedSeries

1.3. Rate

1.4. Season

1.5. Series

1.6. User

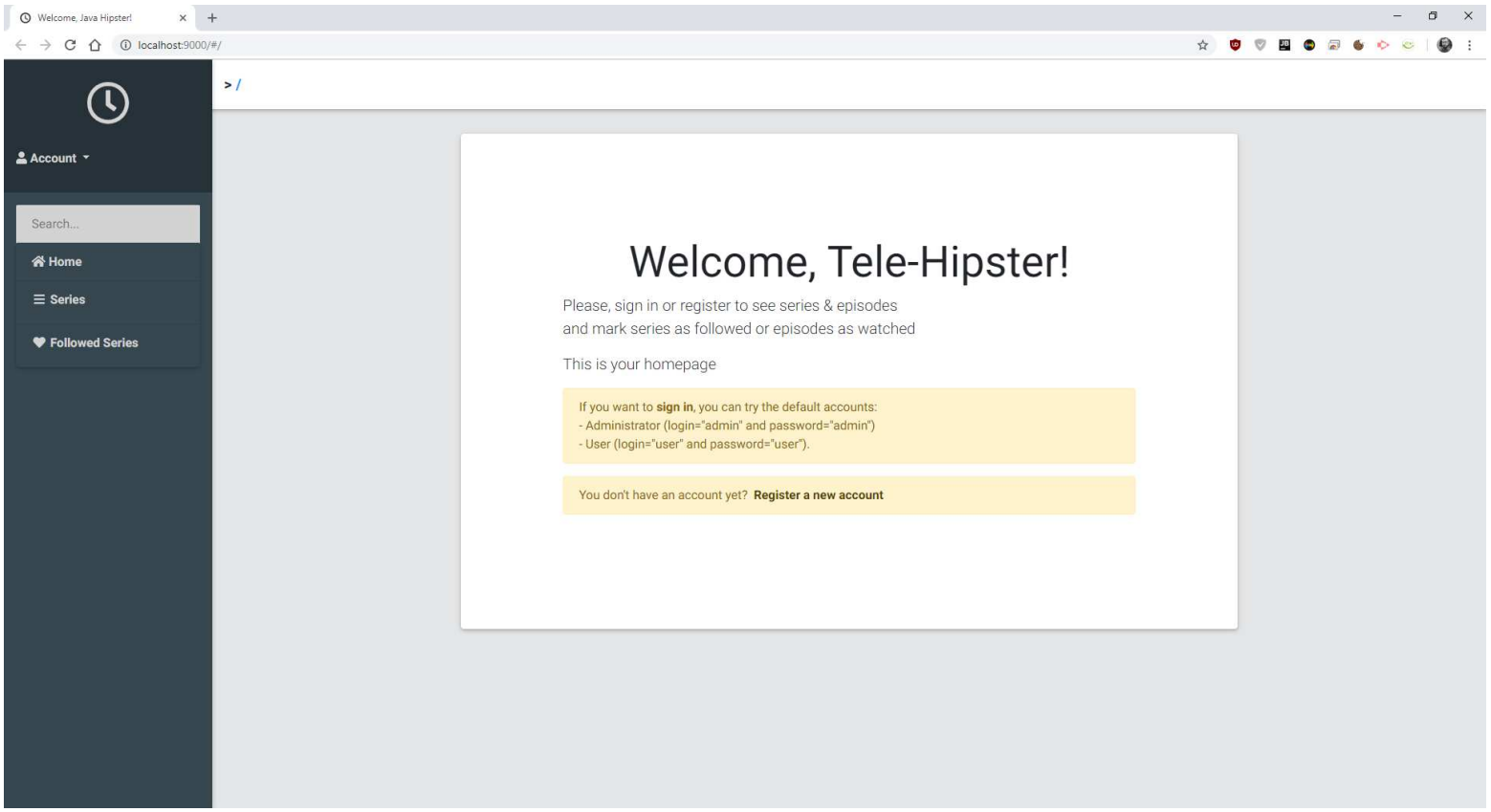
1.7. WatchedEpisode

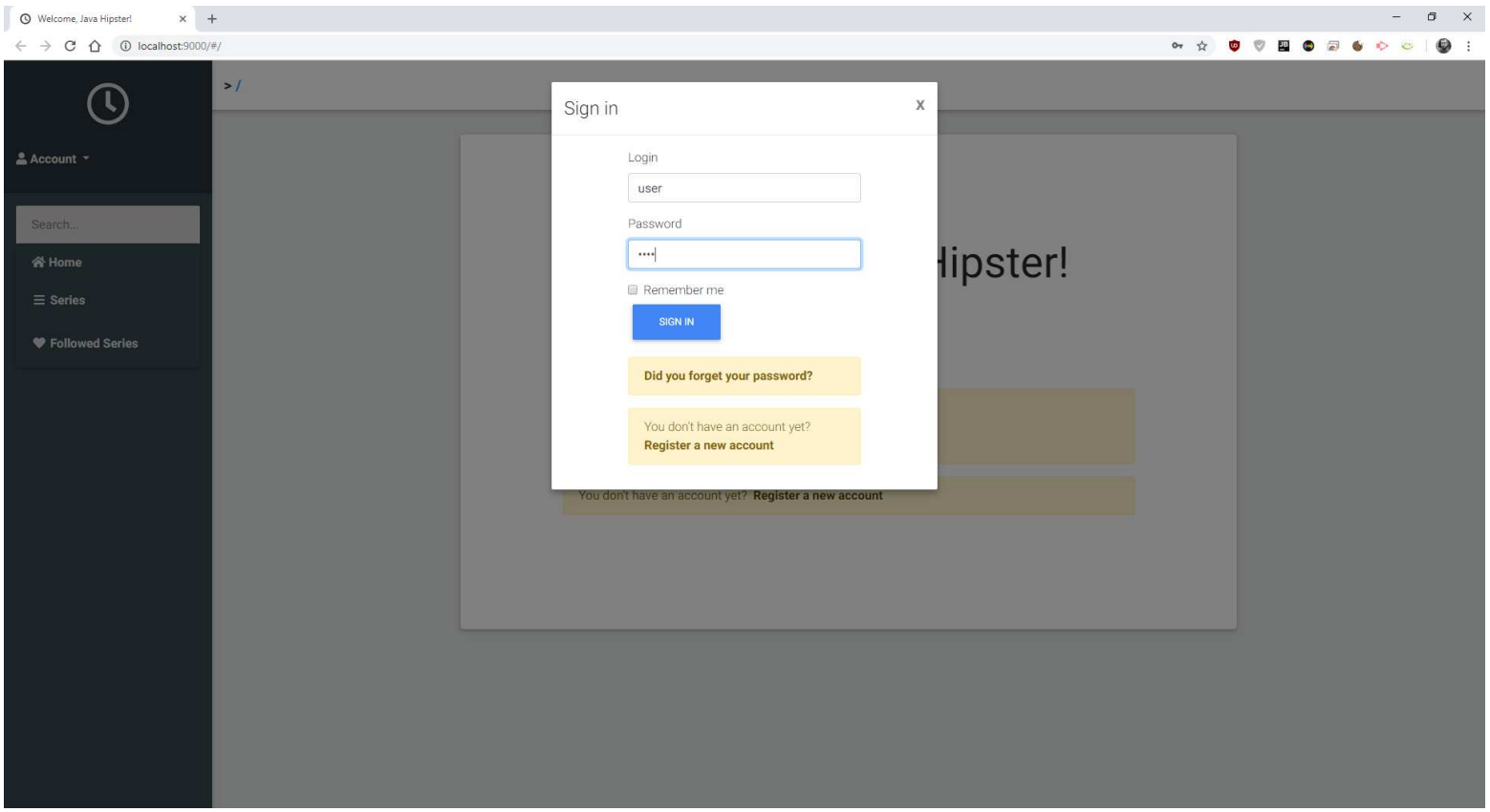
```sql
/* Schemat bazy danych wygenerowany z modelu informacyjnego */

CREATE TABLE Uzytkownicy
(
  login varchar2(100),
  haslo varchar2(100) NOT NULL,
  email varchar2(100) NOT NULL UNIQUE,
  PRIMARY KEY(login)
);

CREATE TABLE Seriale
(
  nazwa varchar2(100),
  rokProdukcji integer,
  opis varchar2(1000),
  PRIMARY KEY(nazwa, rokProdukcji)
);

CREATE TABLE Ocena
(
  rodzaj varchar2(100),
  PRIMARY KEY(rodzaj)
);

CREATE TABLE OpinieSeriale
(
  serialNazwa varchar2(100),
  serialRok integer,
  uzytkownikLogin varchar2(100),
  ocenaRodzaj varchar2(100),
  komentarz varchar2(1000),
  PRIMARY KEY(serialNazwa, serialRok, uzytkownikLogin),
  FOREIGN KEY(serialNazwa, serialRok) REFERENCES Seriale(nazwa, rokProdukcji),
  FOREIGN KEY(uzytkownikLogin) REFERENCES Uzytkownicy(login),
  FOREIGN KEY(ocenaRodzaj) REFERENCES Ocena(rodzaj)
);

CREATE TABLE Sezony
(
  serialNazwa varchar2(100),
  serialRok integer,
  numer integer,
  opis varchar2(1000),
  rokPremiery integer NOT NULL,
  PRIMARY KEY(serialNazwa, serialRok, numer),
  FOREIGN KEY(serialNazwa, serialRok) REFERENCES Seriale(nazwa, rokProdukcji)
);

CREATE TABLE Odcinki
(
  serialNazwa varchar2(100),
  serialRok integer,
  numerSezonu integer,
  numer integer,
  tytul varchar2(100),
  dataPremiery date,
  czasTrwania integer,
  PRIMARY KEY(serialNazwa, serialRok, numerSezonu, numer),
  FOREIGN KEY(serialNazwa, serialRok, numerSezonu) REFERENCES Sezony(serialNazwa,
  serialRok, numer)
);

CREATE TABLE OpinieOdcinki
(
  serialNazwa varchar2(100),
  serialRok integer,
  numerSezonu integer,
  numerOdcinka integer,
  uzytkownikLogin varchar2(100),
  ocenaRodzaj varchar2(100),
  komentarz varchar2(1000),
  PRIMARY KEY(serialNazwa, serialRok, numerSezonu, numerOdcinka, uzytkownikLogin),
  FOREIGN KEY(serialNazwa, serialRok, numerSezonu, numerOdcinka) REFERENCES
```

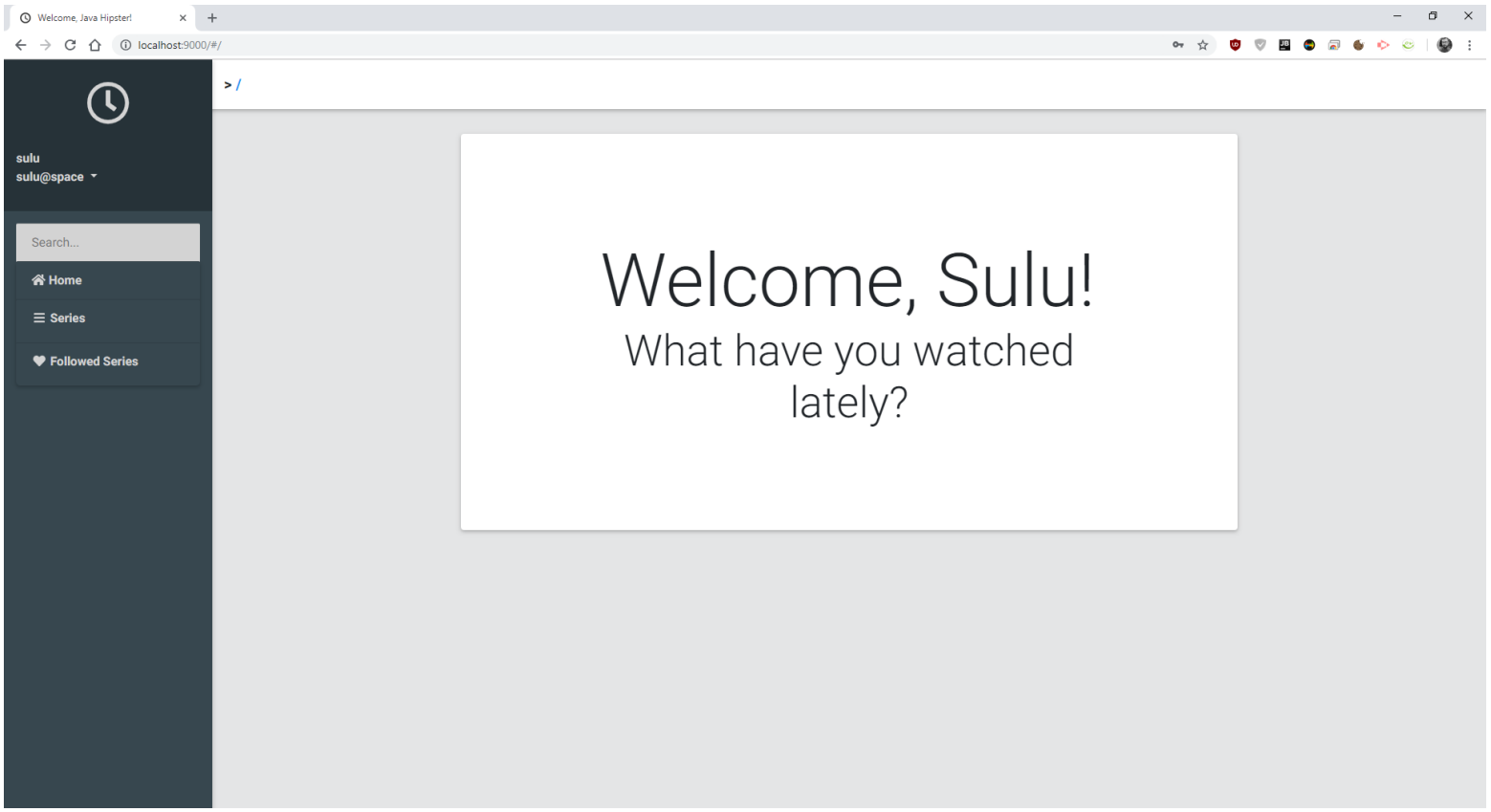
```
    Odcinki(serialNazwa, serialRok, numerSezonu, numer),
    FOREIGN KEY(uzytkownikLogin) REFERENCES Uzytkownicy(login),
    FOREIGN KEY(ocenaRodzaj) REFERENCES Ocena(rodzaj)
);


```

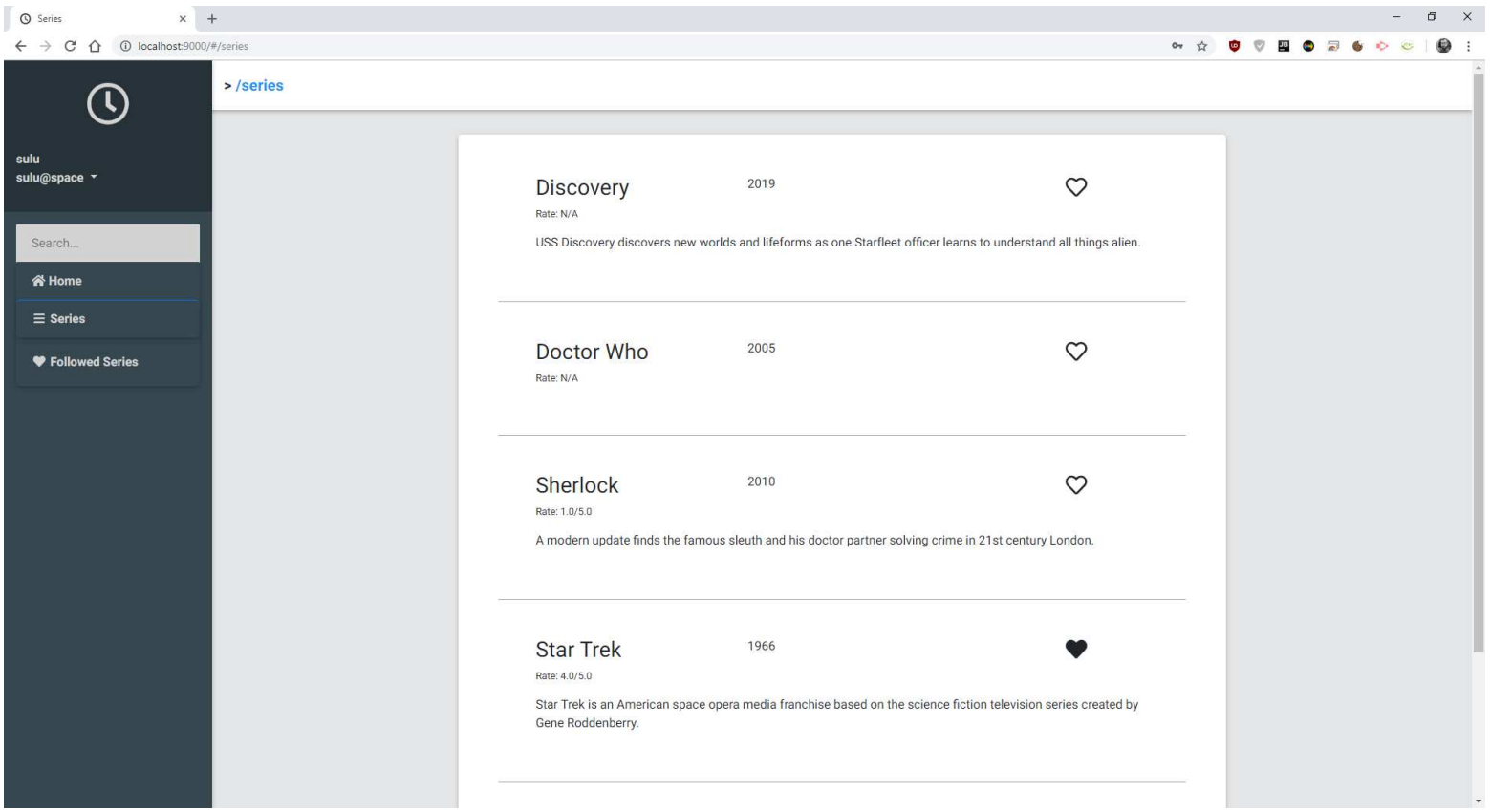
(8)

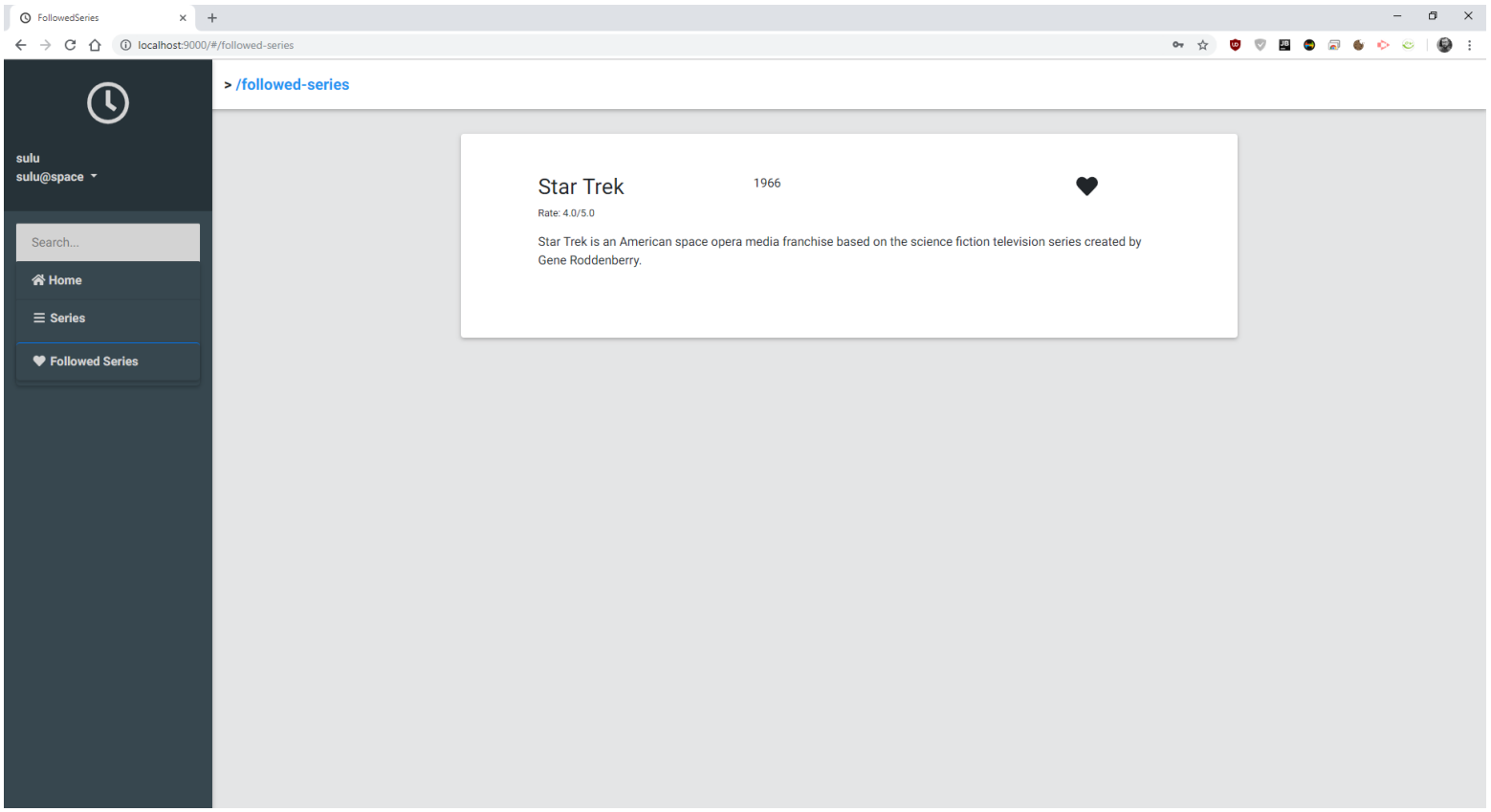
Implementacja interfejsu zgodnie z projektem

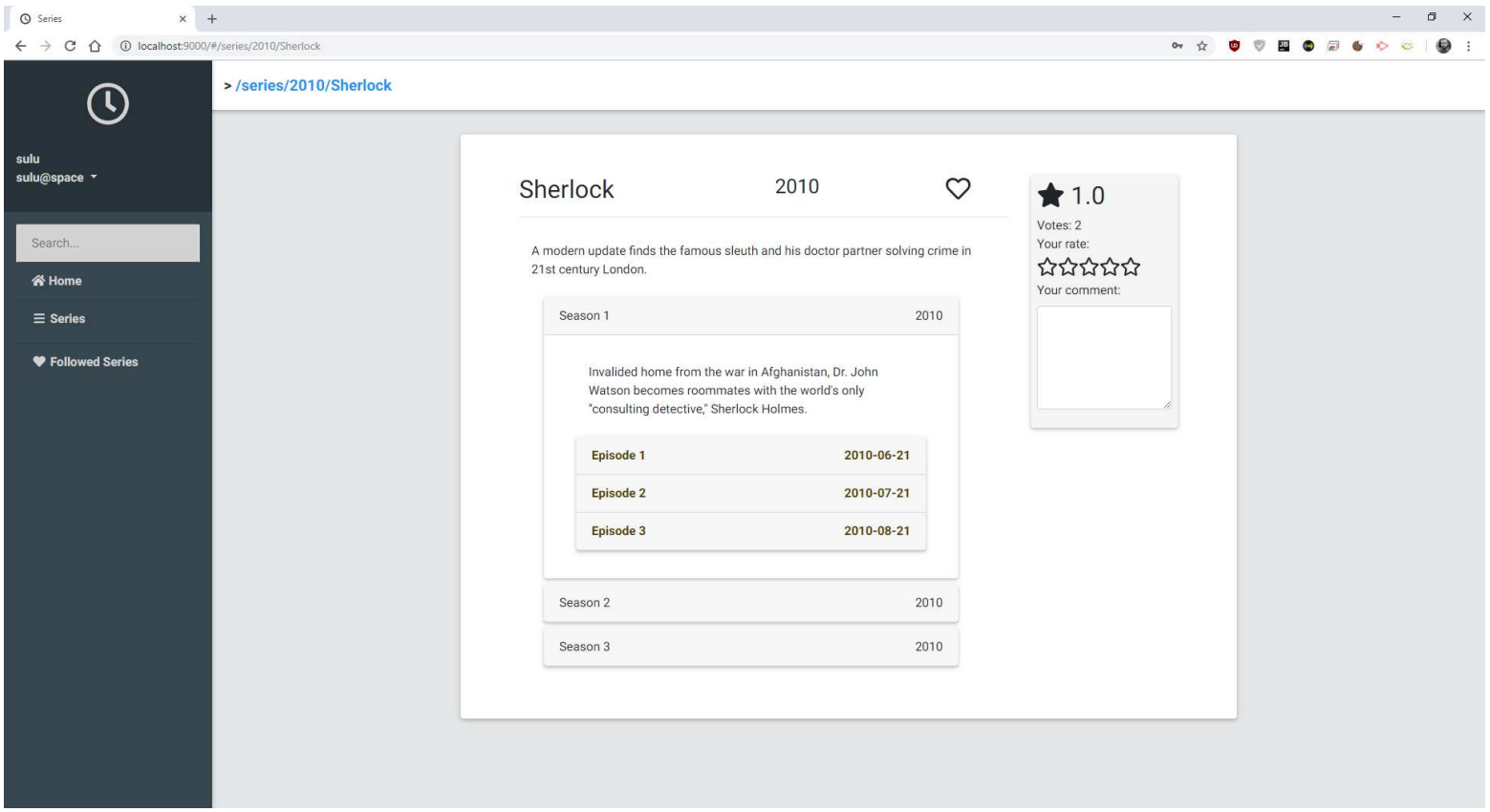

Welcome, Java Hipster!
localhost:9000/#/
> /
Account
Search...
Home
Series
Followed Series
Welcome, Tele-Hipster!
Please, sign in or register to see series & episodes
and mark series as followed or episodes as watched
This is your homepage
If you want to sign in, you can try the default accounts:
- Administrator (login="admin" and password="admin")
- User (login="user" and password="user").
You don't have an account yet? Register a new account

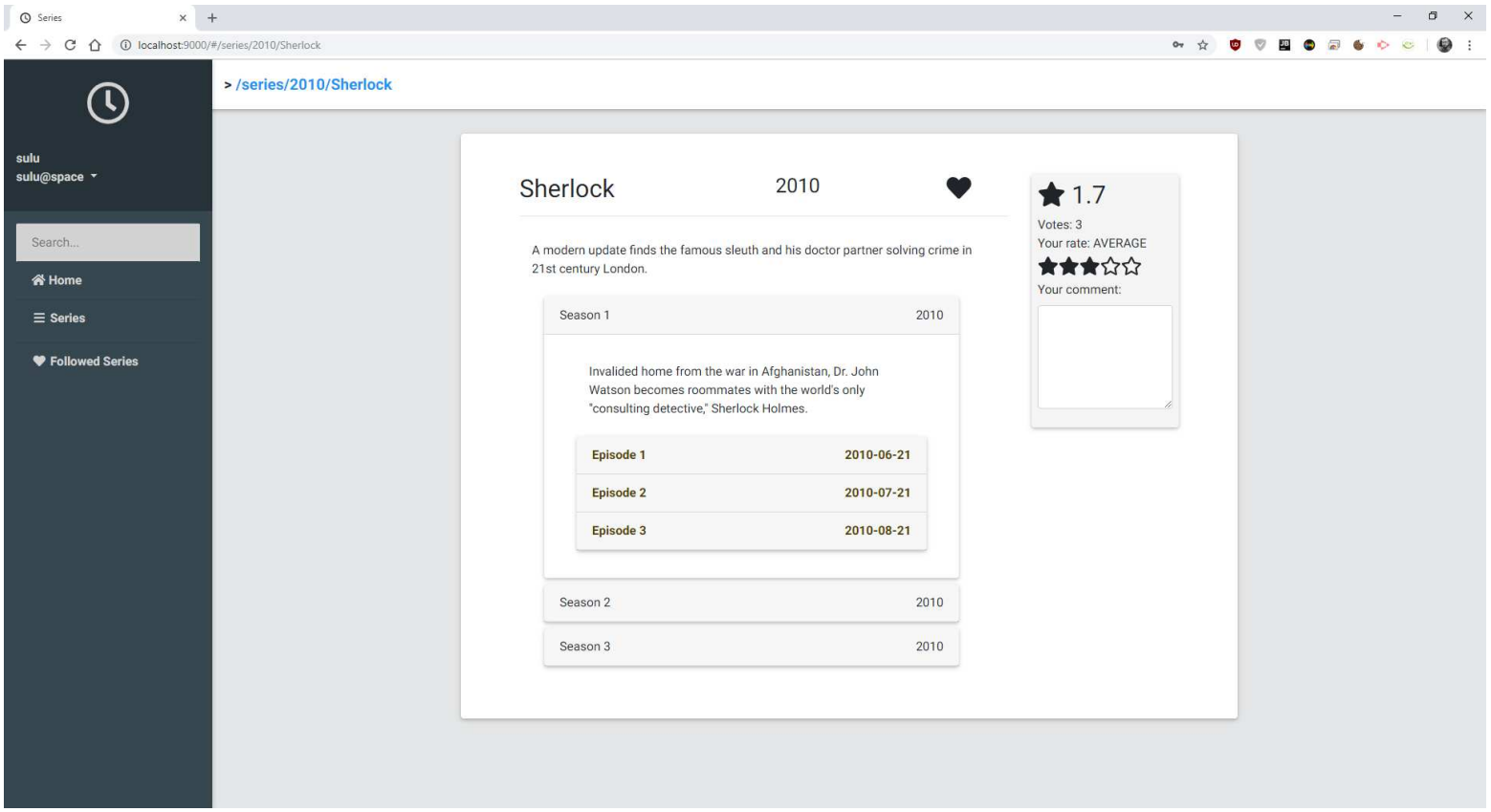

Welcome, Java Hipster!
localhost:9000/#/
> /
Account
Search...
Home
Series
Followed Series
Sign in
Login
user
Password
Remember me
SIGN IN
Did you forget your password?
You don't have an account yet?
Register a new account
You don't have an account yet? Register a new account

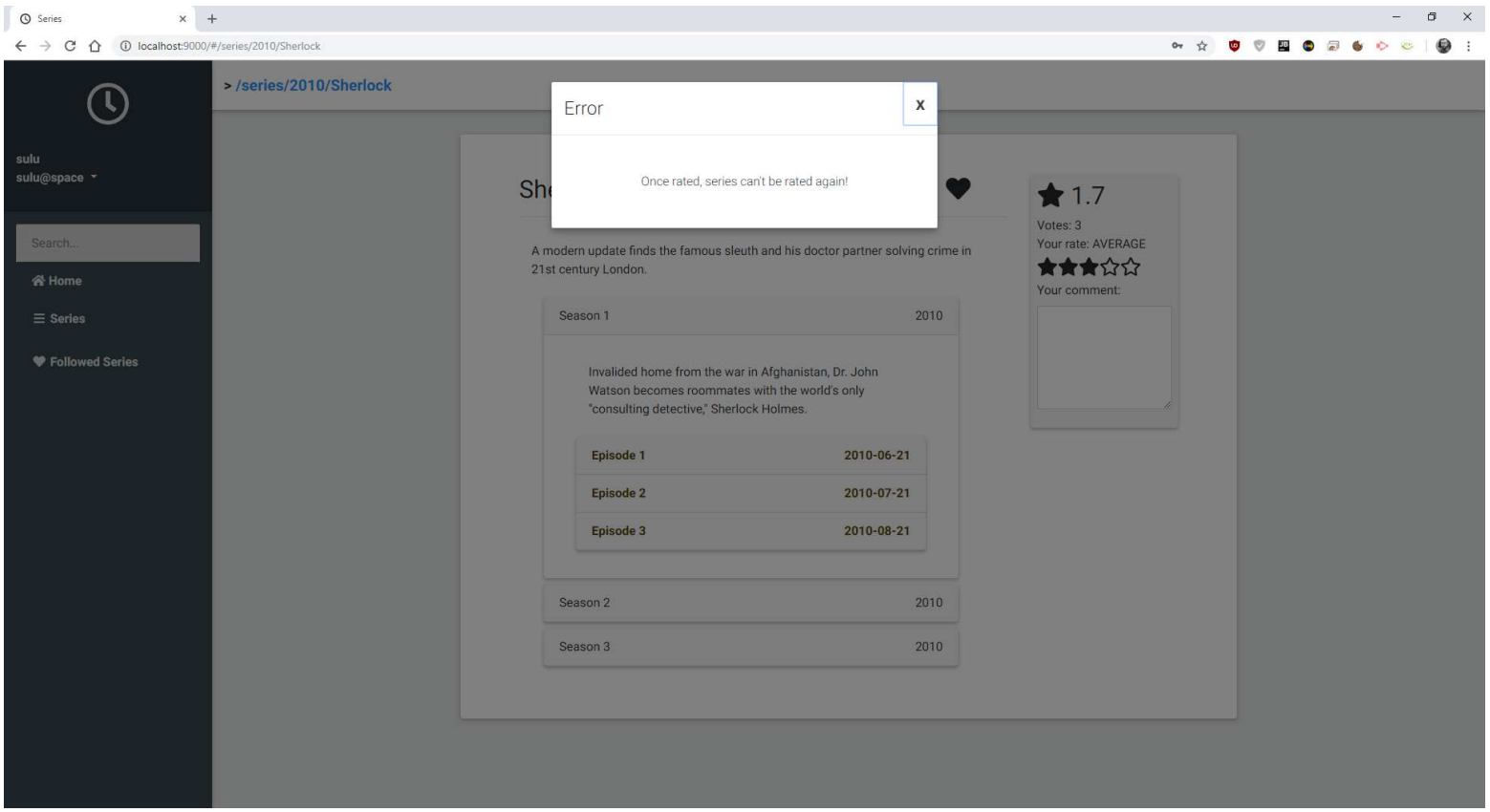

Welcome, Java Hipster!
localhost:9000/#/
sulu
sulu@space
Search...
Home
Series
Followed Series
> /
Welcome, Sulu!
What have you watched lately?

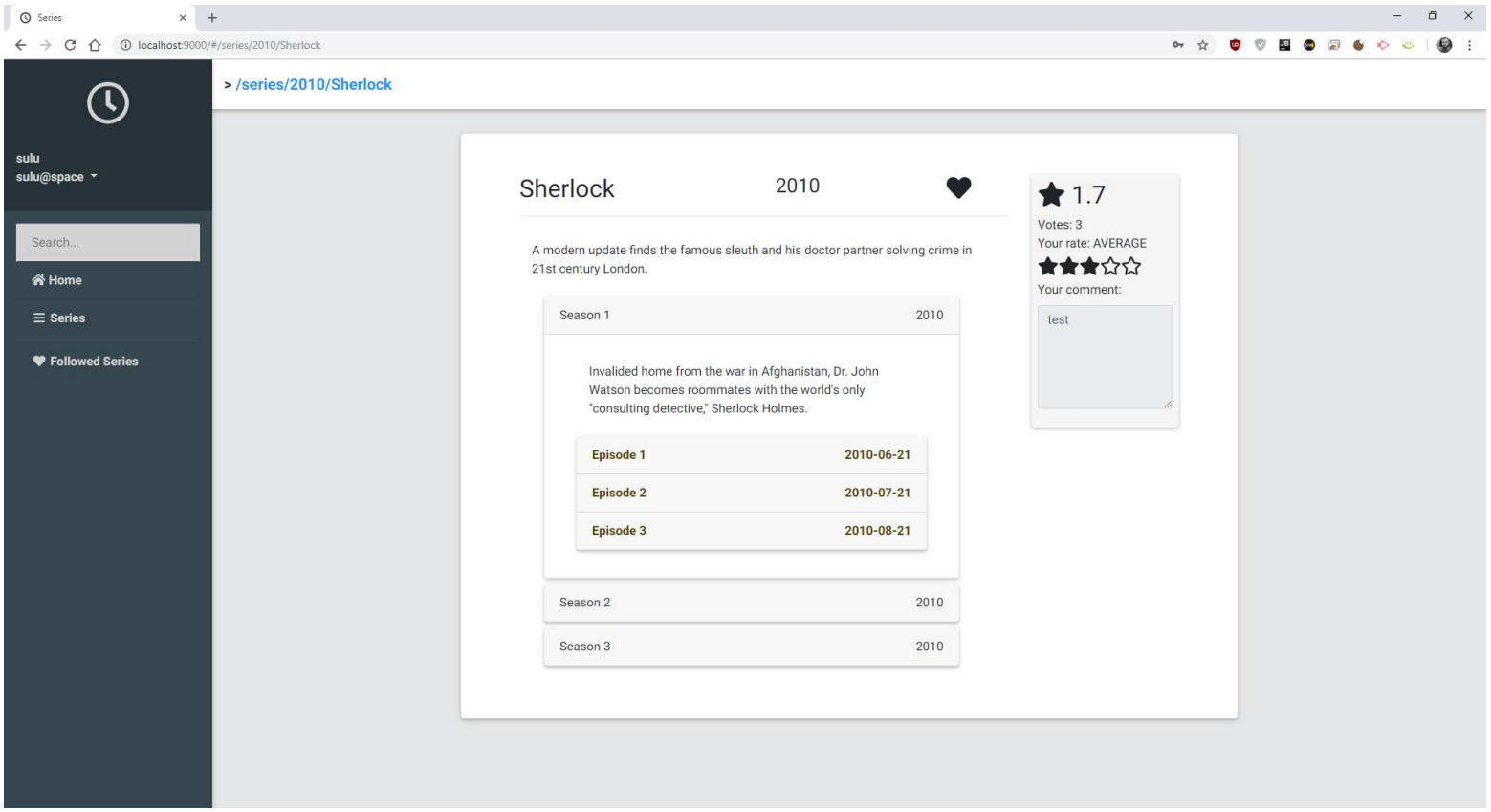

Series
localhost:9000/#/series
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series
Discovery
2019
Rate: N/A
USS Discovery discovers new worlds and lifeforms as one Starfleet officer learns to understand all things alien.
Doctor Who
2005
Rate: N/A
Sherlock
2010
Rate: 1.0/5.0
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Star Trek
1966
Rate: 4.0/5.0
Star Trek is an American space opera media franchise based on the science fiction television series created by Gene Roddenberry.

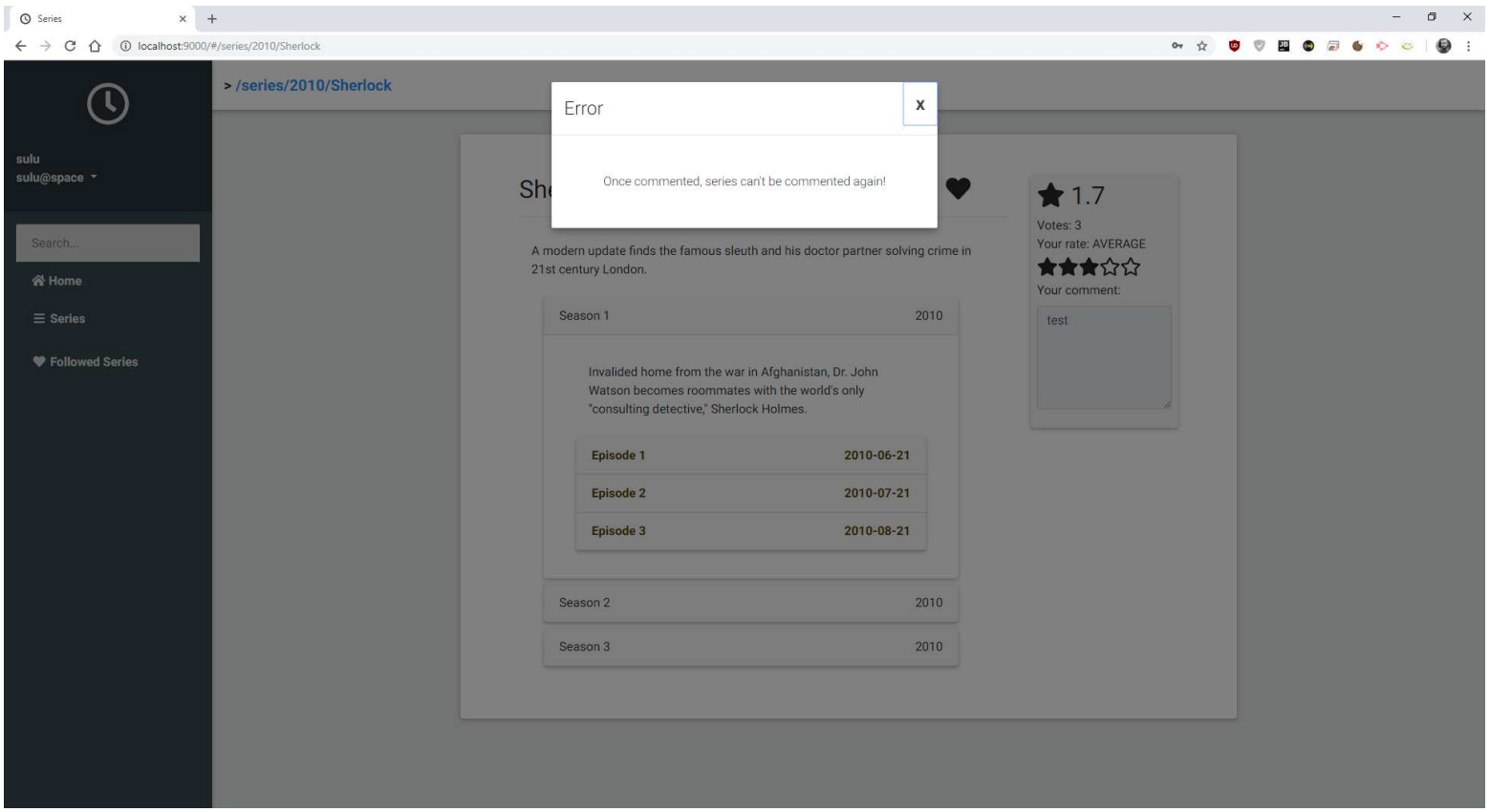

FollowedSeries
localhost:9000/#/followed-series
sulu
sulu@space
Search...
Home
Series
Followed Series
> /followed-series
Star Trek
1966
Rate: 4.0/5.0
Star Trek is an American space opera media franchise based on the science fiction television series created by Gene Roddenberry.

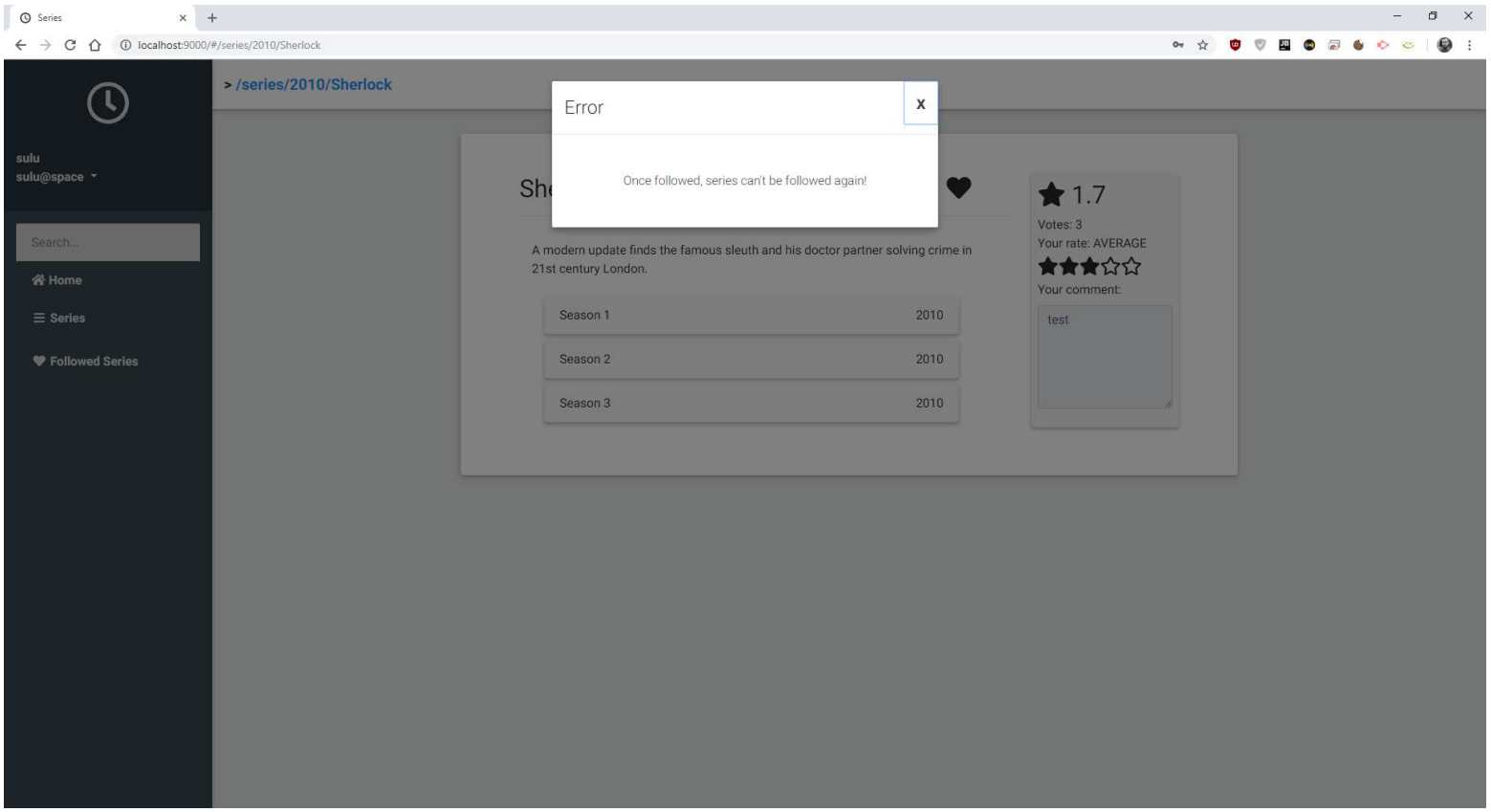

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Sherlock
2010
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1
2010
Invalided home from the war in Afghanistan, Dr. John Watson becomes roommates with the world's only "consulting detective," Sherlock Holmes.
Episode 1
2010-06-21
Episode 2
2010-07-21
Episode 3
2010-08-21
Season 2
2010
Season 3
2010
1.0
Votes: 2
Your rate:
Your comment:

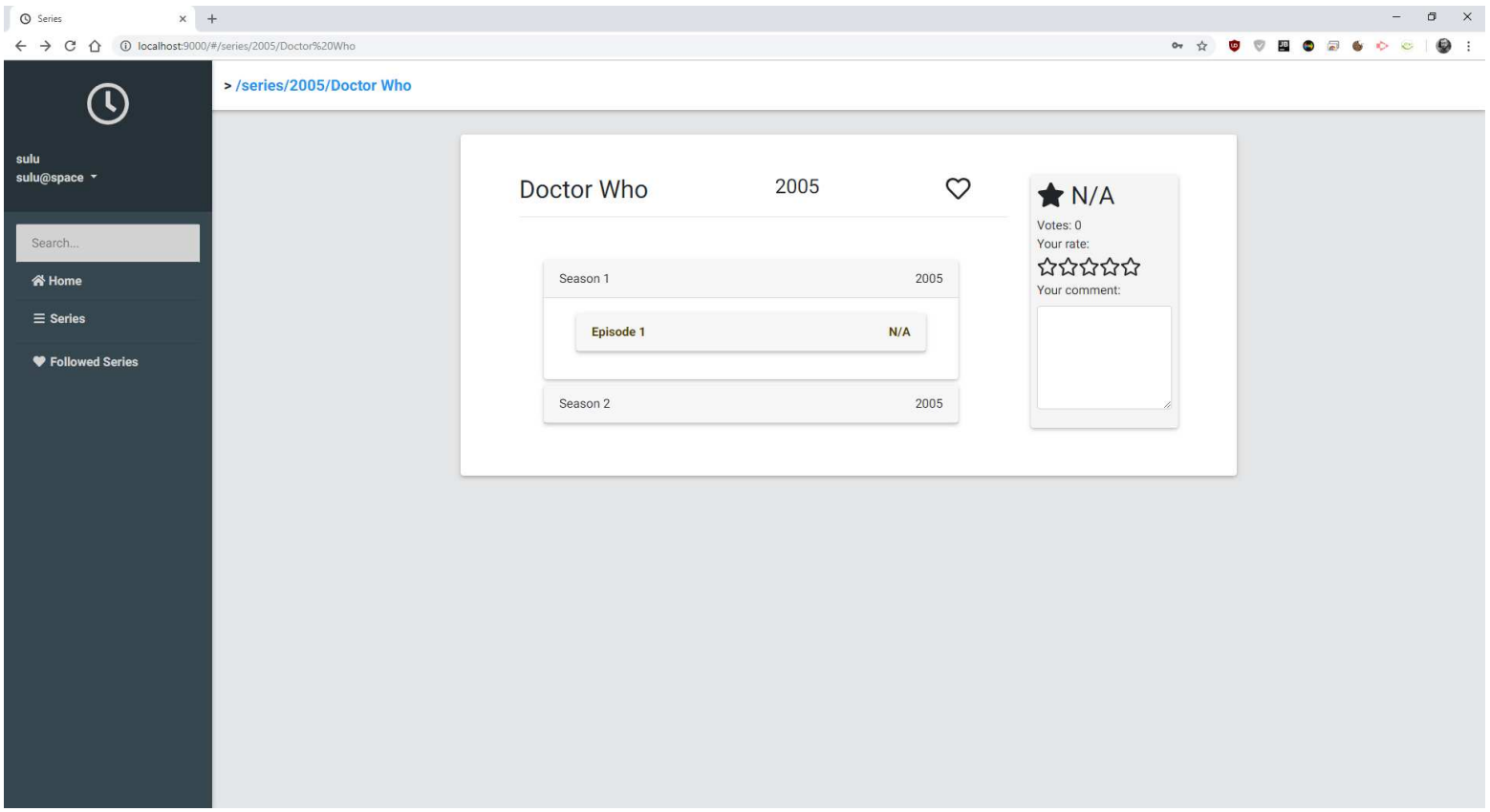

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Sherlock
2010
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1
2010
Invalided home from the war in Afghanistan, Dr. John Watson becomes roommates with the world's only "consulting detective," Sherlock Holmes.
Episode 1
2010-06-21
Episode 2
2010-07-21
Episode 3
2010-08-21
Season 2
2010
Season 3
2010
1.7
Votes: 3
Your rate: AVERAGE
Your comment:

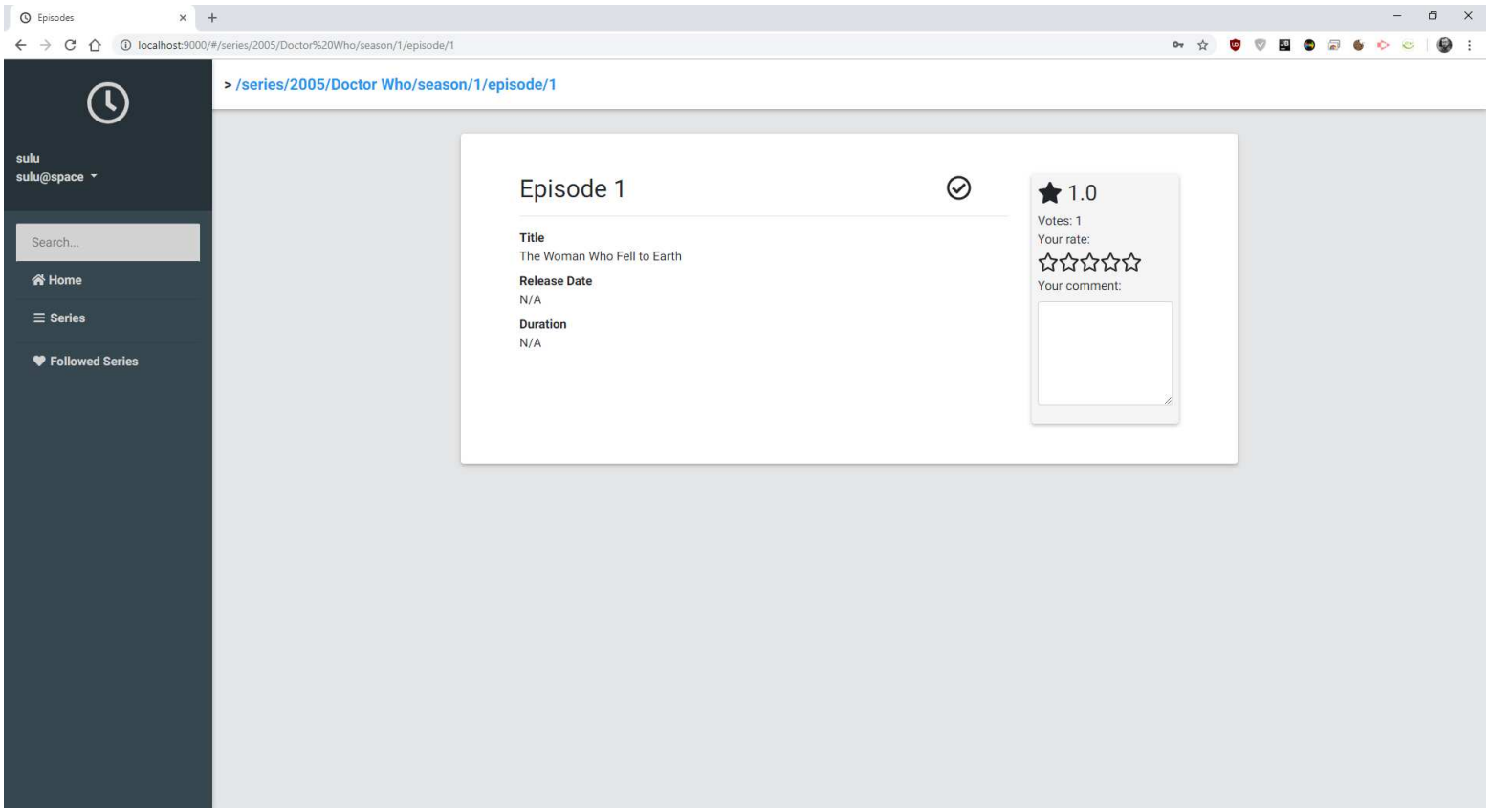

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Error
X
Once rated, series can't be rated again!
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1
2010
Invalided home from the war in Afghanistan, Dr. John Watson becomes roommates with the world's only "consulting detective," Sherlock Holmes.
Episode 1
2010-06-21
Episode 2
2010-07-21
Episode 3
2010-08-21
Season 2
2010
Season 3
2010
1.7
Votes: 3
Your rate: AVERAGE
Your comment:

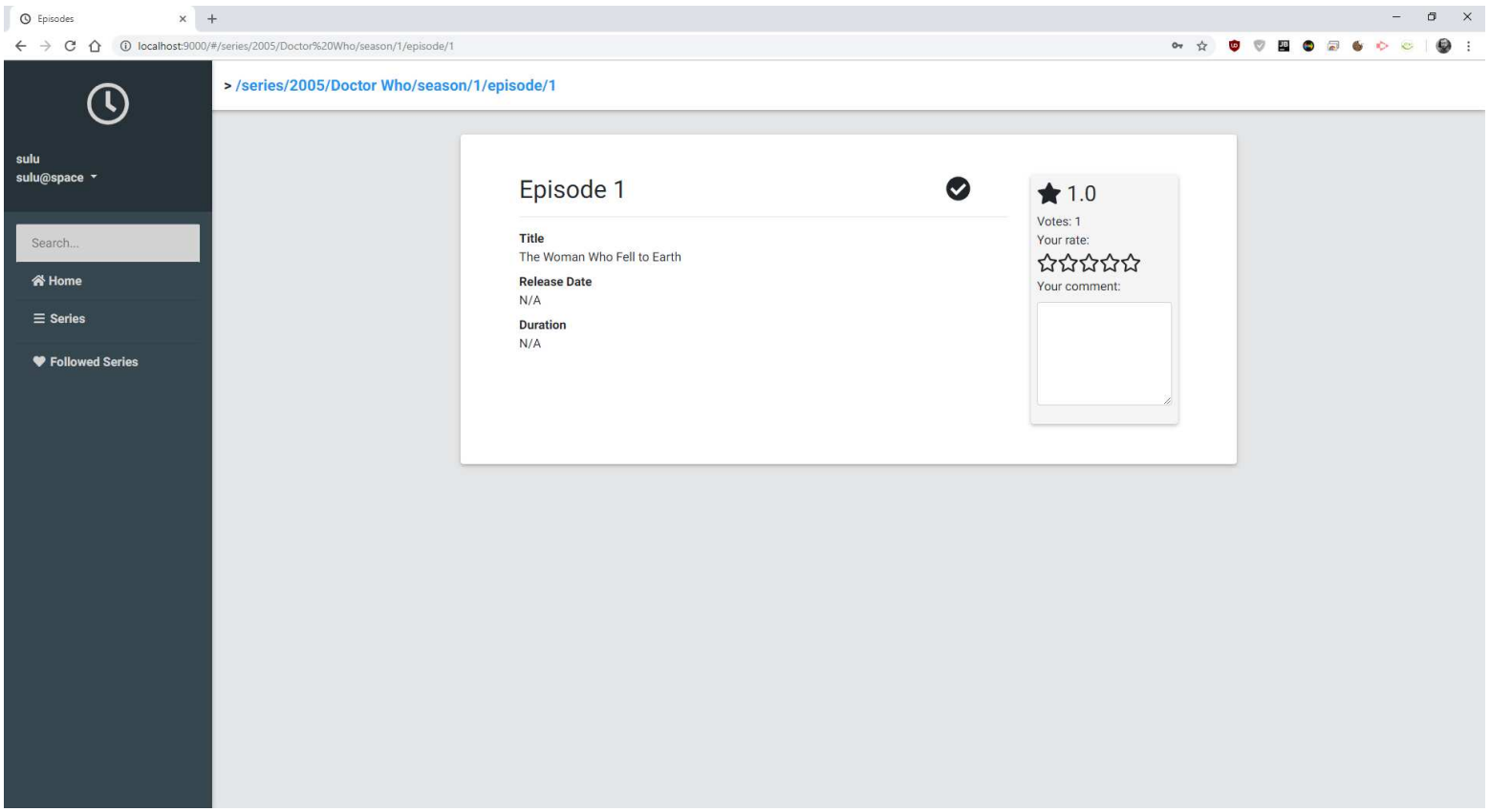

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Sherlock
2010
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1
2010
Invalided home from the war in Afghanistan, Dr. John Watson becomes roommates with the world's only "consulting detective," Sherlock Holmes.
Episode 1
2010-06-21
Episode 2
2010-07-21
Episode 3
2010-08-21
Season 2
2010
Season 3
2010
1.7
Votes: 3
Your rate: AVERAGE
Your comment:
test

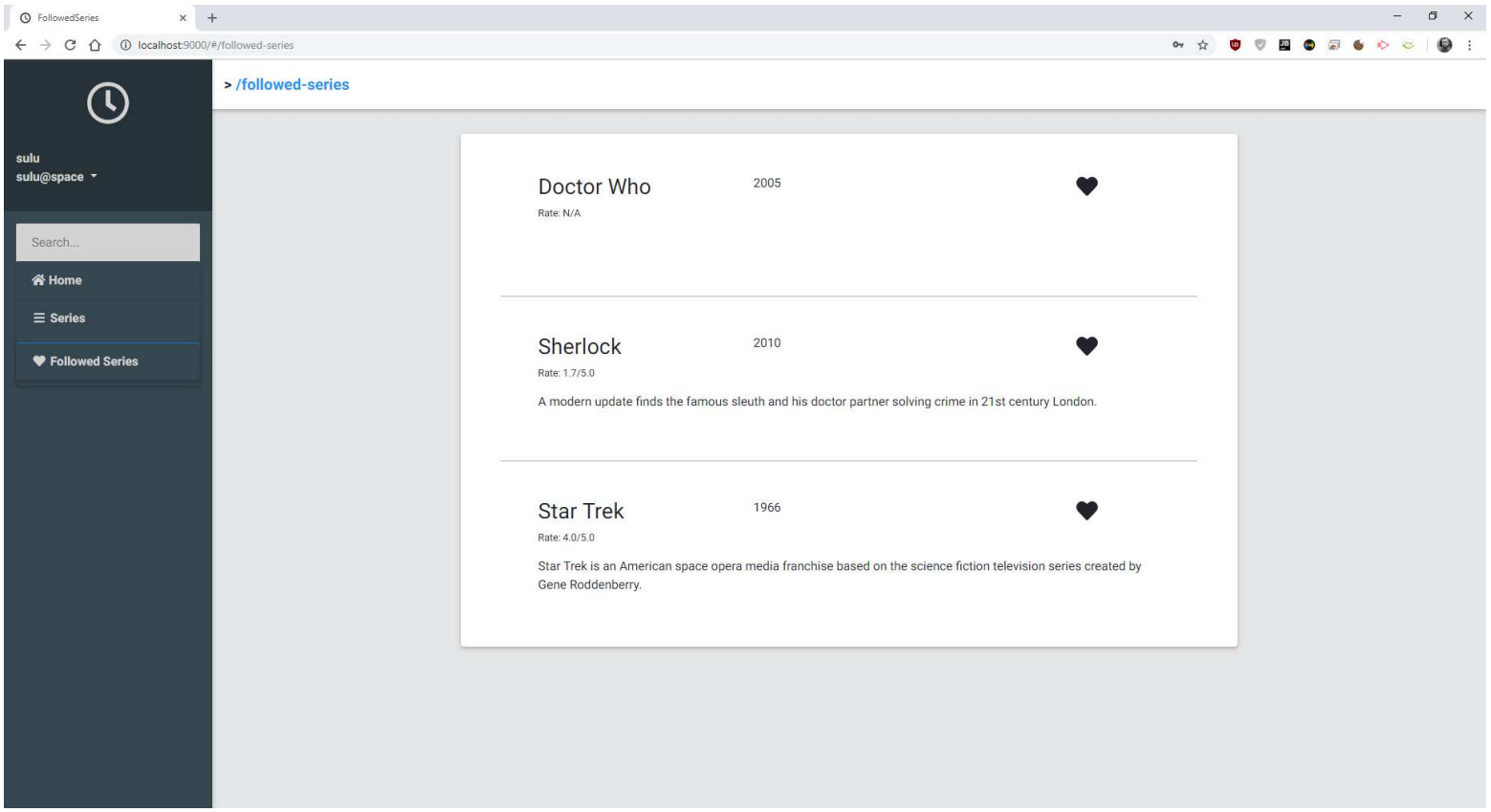

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Error
X
Once commented, series can't be commented again!
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1
2010
Invalided home from the war in Afghanistan, Dr. John Watson becomes roommates with the world's only "consulting detective," Sherlock Holmes.
Episode 1
2010-06-21
Episode 2
2010-07-21
Episode 3
2010-08-21
Season 2
2010
Season 3
2010
1.7
Votes: 3
Your rate: AVERAGE
Your comment:
test

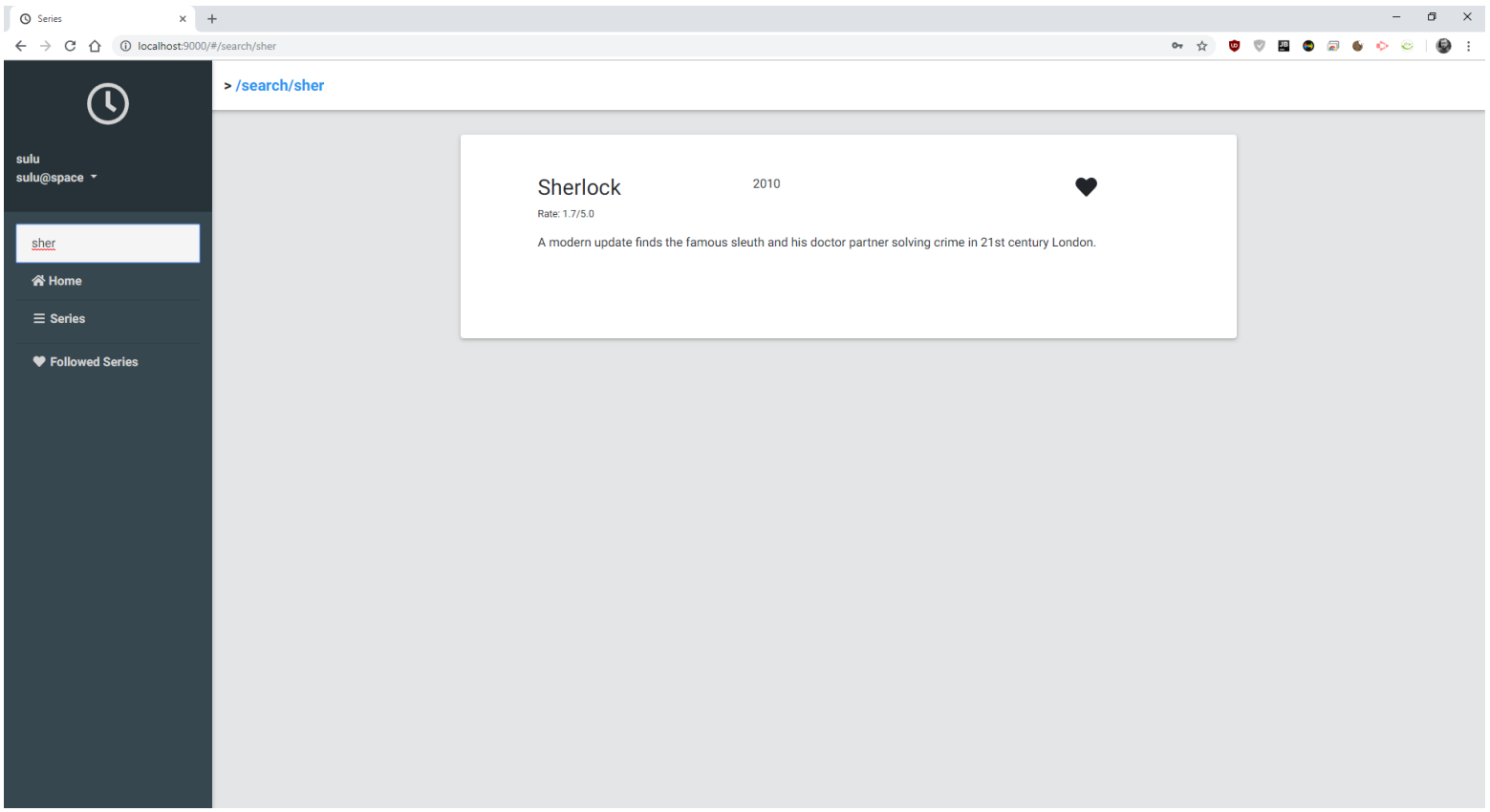

Series
localhost:9000/#/series/2010/Sherlock
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2010/Sherlock
Error
X
Once followed, series can't be followed again!
1.7
Votes: 3
Your rate: AVERAGE
Your comment:
test
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Season 1 2010
Season 2 2010
Season 3 2010


Series
localhost:9000/#/series/2005/Doctor%20Who
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2005/Doctor Who
Doctor Who
2005
N/A
Votes: 0
Your rate:
Your comment:
Season 1 2005
Episode 1 N/A
Season 2 2005


Episodes
localhost:9000/#/series/2005/Doctor%20Who/season/1/episode/1
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2005/Doctor Who/season/1/episode/1
Episode 1
Title
The Woman Who Fell to Earth
Release Date
N/A
Duration
N/A
1.0
Votes: 1
Your rate:
Your comment:


Episodes
localhost:9000/#/series/2005/Doctor%20Who/season/1/episode/1
sulu
sulu@space
Search...
Home
Series
Followed Series
> /series/2005/Doctor Who/season/1/episode/1
Episode 1
Title
The Woman Who Fell to Earth
Release Date
N/A
Duration
N/A
1.0
Votes: 1
Your rate:
Your comment:


FollowedSeries
localhost:9000/#/followed-series
sulu
sulu@space
Search...
Home
Series
Followed Series
> /followed-series
Doctor Who
2005
Rate: N/A
Sherlock
2010
Rate: 1.7/5.0
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.
Star Trek
1966
Rate: 4.0/5.0
Star Trek is an American space opera media franchise based on the science fiction television series created by Gene Roddenberry.


Series
localhost:9000/#/search/sher
sulu
sulu@space
sher
Home
Series
Followed Series
> /search/sher
Sherlock
2010
Rate: 1.7/5.0
A modern update finds the famous sleuth and his doctor partner solving crime in 21st century London.

(9)

Podłączenie logiki aplikacji do interfejsu

```
<!--episode-detail.component.html-->

<div class="row justify-content-center">
    <div class="col-11 no-padding">
        <div *ngIf="episode" class="row">
            <div class="col-md-9">
                <h2 class="d-block">
                    <div class="row">
                        <span class="col-10">Episode {{episode.number}}</span>
                        <fa-icon class="col-2" *ngIf="watchedEpisode"

                                 [icon]="watchedEpisode.id?['fas','check-circle']:['fa
                                 r','check-circle']"
                                 (click)="setFollowed()"></fa-icon>
                    </div>
                </h2>
                <hr>
                <jhi-alert-error></jhi-alert-error>
                <dl class="row-md jh-entity-details">
                    <dt><span>Title</span></dt>
                    <dd>
                        <span>{{episode.title}}</span>
                    </dd>
                    <dt><span>Release Date</span></dt>
                    <dd>
                        <span *ngIf="episode.releaseDate">{{episode.releaseDate |
                        date : 'fullDate'}}<br>{{episode.releaseDate | date :
                        'shortTime'}}</span>
                        <span *ngIf="!episode.releaseDate">N/A</span>
                    </dd>
                    <dt><span>Duration</span></dt>
                    <dd>
                        <span *ngIf="episode.duration">{{episode.duration}} min</span>
                        <span *ngIf="!episode.duration">N/A</span>
                    </dd>
                </dl>

                <button *jhiHasAnyAuthority="'ROLE_ADMIN'" type="submit"
                        (click)="previousState()"
                        class="btn btn-info">
                    <fa-icon [icon]="'arrow-left'"></fa-icon> <span> Back</span>
                </button>

                <button *jhiHasAnyAuthority="'ROLE_ADMIN'" type="button"
                        [routerLink]="['/episode', episode.id, 'edit']"
                        class="btn btn-primary">
                    <fa-icon [icon]="'pencil-alt'"></fa-icon> <span> Edit</span>
                </button>
            </div>
            <div class="col-md-3">
                <div class="card opinion" *ngIf="watchedEpisode">
                    <div>
                        <h3>
                            <fa-icon [icon]="['fas','star']"></fa-icon>
                            <span *ngIf="averageRate"> {{averageRate | number :
                            '1.1-1'}}</span>
                            <span *ngIf="!averageRate"> N/A</span>
                        </h3>
                        <div>
                            Votes: {{rateCount}}
                        </div>
                        <div>
                            Your rate: {{watchedEpisode.rate}}
                        </div>
                        <div class="stars">
                            <fa-icon *ngFor="let i of [1,2,3,4,5]"

                                     [icon]="shouldStarBeFilled(i)?['fas','star']:['fa
                                     r','star']"
                                     (click)="setRate(i)"></fa-icon>
                        </div>
                        <form>
```

```
                                <div class="form-group">
                                    <label for="comment">Your comment:</label>
                                    <div (click)="showErrorModal()">
                                    <textarea class="form-control" rows="5"
                                              id="comment"
                                              #comment
                                              (keydown.enter)="saveComment(comment.value)"

                                              [disabled]="watchedEpisode.comment">{{watche
                                              dEpisode.comment}}</textarea>
                                    </div>
                                </div>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```
// episode-detail.component.ts

import {Component, OnInit} from '@angular/core';
import {ActivatedRoute} from '@angular/router';

import {IEpisode} from 'app/shared/model/episode.model';
import {HttpErrorResponse, HttpResponse} from "@angular/common/http";
import {IWatchedEpisode, Rate} from "app/shared/model/watched-episode.model";
import {JhiAlertService} from "ng-jhipster";
import {EpisodeService} from "app/entities/episode/episode.service";
import {WatchedEpisodeService} from "app/entities/watched-episode";
import {ErrorModalService} from "app/core/error-modal/error-modal.service";

@Component({
    selector: 'jhi-episode-detail',
    templateUrl: './episode-detail.component.html'
})
export class EpisodeDetailComponent implements OnInit {
    episode: IEpisode;
    watchedEpisode: IWatchedEpisode;
    averageRate: number;
    private rateCount: number;

    constructor(private activatedRoute: ActivatedRoute, private episodeService:
    EpisodeService, private jhiAlertService: JhiAlertService, private
    watchedEpisodeService: WatchedEpisodeService, private errorModalService:
    ErrorModalService) {
    }

    ngOnInit() {
        this.activatedRoute.data.subscribe(({episode}) => {
            this.episode = episode;
            this.loadWatched();
            this.updateCommunityRate();
        });
    }

    shouldStarBeFilled(starNr: number): boolean {
        if (this.watchedEpisode.rate) {
            switch (this.watchedEpisode.rate) {
                case 'BAD':
                    return starNr <= 1;
                case 'MEDIOCRE':
                    return starNr <= 2;
                case 'AVERAGE':
                    return starNr <= 3;
                case 'GOOD':
                    return starNr <= 4;
                case 'AWESOME':
                    return starNr <= 5;
            }
        }
        else return false;
    }

    setRate(i: number) {
        if (this.watchedEpisode.id && this.watchedEpisode.rate) {
            this.errorModalService.open("Once rated, episode can't be rated again!");
        }
        else {
            switch (i) {
                case 1:
                    this.watchedEpisode.rate = Rate.BAD;
                    break;
                case 2:
                    this.watchedEpisode.rate = Rate.MEDIOCRE;
                    break;
                case 3:
                    this.watchedEpisode.rate = Rate.AVERAGE;
                    break;
                case 4:
                    this.watchedEpisode.rate = Rate.GOOD;
                    break;
```

```
                case 5:
                    this.watchedEpisode.rate = Rate.AWESOME;
                    break;
            }
            this.saveWatchedEpisode();
        }
    }

    saveWatchedEpisode() {
        if (this.watchedEpisode.id) {
            this.watchedEpisodeService.update(this.watchedEpisode).subscribe(
                (res: HttpResponse<IWatchedEpisode>) => {
                    this.watchedEpisode = res.body;
                    this.updateCommunityRate();
                }
            );
        } else {
            this.watchedEpisodeService.create(this.watchedEpisode).subscribe((res:
            HttpResponse<IWatchedEpisode>) => {
                this.watchedEpisode = res.body;
                this.updateCommunityRate();
            });
        }
    }

    saveComment(value: string) {
        if (this.watchedEpisode.id && this.watchedEpisode.comment) {
            this.errorModalService.open("Once commented, episode can't be commented
            again!");
        }
        else {
            this.watchedEpisode.comment = value;
            this.saveWatchedEpisode();
        }
    }

    setFollowed() {
        if (this.watchedEpisode && this.watchedEpisode.id) {
            this.errorModalService.open("Once watched, episode can't be watched
            again!");
        }
        else {
            this.saveWatchedEpisode();
        }
    }

    updateCommunityRate() {
        this.getAverageRate();
        this.getRateCount();
    }

    getAverageRate() {
        if (this.episode && this.episode.id) {
            this.watchedEpisodeService.getAverageRate(this.episode.id).subscribe(
                (res: HttpResponse<any>) => {
                    this.averageRate = res.body;
                }
            )
        }
    }

    getRateCount() {
        if (this.episode && this.episode.id) {
            this.watchedEpisodeService.getRateCount(this.episode.id).subscribe(
                (res: HttpResponse<any>) => {
                    this.rateCount = res.body;
                }
            )
        }
    }

    loadWatched() {
        this.watchedEpisodeService.findByEpisodeId(this.episode.id).subscribe(
```

```
141                 (res: HttpResponse<IWatchedEpisode>) => {
142                     this.watchedEpisode = res.body;
143                 },
144                 (res: HttpErrorResponse) => this.onError(res.message)
145             );
146         }
147
148         private onError(errorMessage: string) {
149             this.jhiAlertService.error(errorMessage, null, null);
150         }
151
152         previousState() {
153             window.history.back();
154         }
155
156         showErrorModal() {
157             if (this.watchedEpisode.id && this.watchedEpisode.comment) {
158                 this.errorModalService.open("Once commented, episode can't be commented
                    again!");
159             }
160         }
161     }
162
```

```typescript
//watched-episode.service.ts

import { Injectable } from '@angular/core';
import { HttpClient, HttpResponse } from '@angular/common/http';
import { Observable } from 'rxjs';

import { SERVER_API_URL } from 'app/app.constants';
import { createRequestOption } from 'app/shared';
import { IWatchedEpisode } from 'app/shared/model/watched-episode.model';
import {map} from "rxjs/operators";
import * as moment from "moment";

type EntityResponseType = HttpResponse<IWatchedEpisode>;
type EntityArrayResponseType = HttpResponse<IWatchedEpisode[]>;

@Injectable({ providedIn: 'root' })
export class WatchedEpisodeService {
    public resourceUrl = SERVER_API_URL + 'api/watched-episodes';

    constructor(private http: HttpClient) {}

    create(watchedEpisode: IWatchedEpisode): Observable<EntityResponseType> {
        return this.http.post<IWatchedEpisode>(this.resourceUrl, watchedEpisode, {
        observe: 'response' });
    }

    update(watchedEpisode: IWatchedEpisode): Observable<EntityResponseType> {
        return this.http.put<IWatchedEpisode>(this.resourceUrl, watchedEpisode, {
        observe: 'response' });
    }

    find(id: number): Observable<EntityResponseType> {
        return this.http.get<IWatchedEpisode>(`${this.resourceUrl}/${id}`, {
        observe: 'response' });
    }

    findByEpisodeId(id: number): Observable<EntityResponseType>{
        return this.http
            .get<IWatchedEpisode>(`${this.resourceUrl}/episode/${id}`, { observe:
            'response' })
            .pipe(map((res: EntityResponseType) =>
            WatchedEpisodeService.convertDateFromServer(res)));
    }

    getAverageRate(episodeId: number): Observable<EntityResponseType> {
        return
        this.http.get<IWatchedEpisode>(`${this.resourceUrl}/${episodeId}/average-rate`
        , { observe: 'response' });
    }

    getRateCount(episodeId: number): Observable<EntityResponseType> {
        return
        this.http.get<IWatchedEpisode>(`${this.resourceUrl}/${episodeId}/rate-count`,
        { observe: 'response' });
    }

    query(req?: any): Observable<EntityArrayResponseType> {
        const options = createRequestOption(req);
        return this.http.get<IWatchedEpisode[]>(this.resourceUrl, { params: options,
        observe: 'response' });
    }

    delete(id: number): Observable<HttpResponse<any>> {
        return this.http.delete<any>(`${this.resourceUrl}/${id}`, { observe:
        'response' });
    }

    protected static convertDateFromServer(res: EntityResponseType):
    EntityResponseType {
        if (res.body) {
            res.body.episode.releaseDate = res.body.episode.releaseDate != null ?
            moment(res.body.episode.releaseDate) : null;
        }
```

```
61              return res;
62          }
63      }
64
```

```java
//WatchedEpisodeResource.java

package pl.marczynski.seriesapp.web.rest;

import com.codahale.metrics.annotation.Timed;
import pl.marczynski.seriesapp.domain.WatchedEpisode;
import pl.marczynski.seriesapp.service.WatchedEpisodeService;
import pl.marczynski.seriesapp.web.rest.errors.BadRequestAlertException;
import pl.marczynski.seriesapp.web.rest.jhipster.util.HeaderUtil;
import io.github.jhipster.web.util.ResponseUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.net.URI;
import java.net.URISyntaxException;

import java.util.List;
import java.util.Optional;

/**
 * REST controller for managing WatchedEpisode.
 */
@RestController
@RequestMapping("/api")
public class WatchedEpisodeResource {

    private final Logger log = LoggerFactory.getLogger(WatchedEpisodeResource.class);

    private static final String ENTITY_NAME = "watchedEpisode";

    private final WatchedEpisodeService watchedEpisodeService;

    public WatchedEpisodeResource(WatchedEpisodeService watchedEpisodeService) {
        this.watchedEpisodeService = watchedEpisodeService;
    }

    /**
     * POST  /watched-episodes : Create a new watchedEpisode.
     *
     * @param watchedEpisode the watchedEpisode to create
     * @return the ResponseEntity with status 201 (Created) and with body the new
     watchedEpisode, or with status 400 (Bad Request) if the watchedEpisode has
     already an ID
     * @throws URISyntaxException if the Location URI syntax is incorrect
     */
    @PostMapping("/watched-episodes")
    @Timed
    public ResponseEntity<WatchedEpisode> createWatchedEpisode(@Valid @RequestBody
    WatchedEpisode watchedEpisode) throws URISyntaxException {
        log.debug("REST request to save WatchedEpisode : {}", watchedEpisode);
        if (watchedEpisode.getId() != null) {
            throw new BadRequestAlertException("A new watchedEpisode cannot already
            have an ID", ENTITY_NAME, "idexists");
        }
        WatchedEpisode result = watchedEpisodeService.save(watchedEpisode);
        return ResponseEntity.created(new URI("/api/watched-episodes/" +
        result.getId()))
            .headers(HeaderUtil.createEntityCreationAlert(ENTITY_NAME,
            result.getId().toString()))
            .body(result);
    }

    /**
     * PUT  /watched-episodes : Updates an existing watchedEpisode.
     *
     * @param watchedEpisode the watchedEpisode to update
     * @return the ResponseEntity with status 200 (OK) and with body the updated
     watchedEpisode,
     * or with status 400 (Bad Request) if the watchedEpisode is not valid,
     * or with status 500 (Internal Server Error) if the watchedEpisode couldn't be
```

```
         updated
67       */
68      @PutMapping("/watched-episodes")
69      @Timed
70      public ResponseEntity<WatchedEpisode> updateWatchedEpisode(@Valid @RequestBody
        WatchedEpisode watchedEpisode) {
71          log.debug("REST request to update WatchedEpisode : {}", watchedEpisode);
72          if (watchedEpisode.getId() == null) {
73              throw new BadRequestAlertException("Invalid id", ENTITY_NAME, "idnull");
74          }
75          WatchedEpisode result = watchedEpisodeService.update(watchedEpisode);
76          return ResponseEntity.ok()
77              .headers(HeaderUtil.createEntityUpdateAlert(ENTITY_NAME,
                watchedEpisode.getId().toString()))
78              .body(result);
79      }
80
81      /**
82       * GET  /watched-episodes : get all the watchedEpisodes.
83       *
84       * @return the ResponseEntity with status 200 (OK) and the list of
         watchedEpisodes in body
85       */
86      @GetMapping("/watched-episodes")
87      @Timed
88      public List<WatchedEpisode> getAllWatchedEpisodes() {
89          log.debug("REST request to get all WatchedEpisodes");
90          return watchedEpisodeService.findAll();
91      }
92
93      /**
94       * GET  /watched-episodes/:id : get the watchedEpisode by "id".
95       *
96       * @param id the id of the watchedEpisode to retrieve
97       * @return the ResponseEntity with status 200 (OK) and with body the
         watchedEpisode, or with status 404 (Not Found)
98       */
99      @GetMapping("/watched-episodes/{id}")
100     @Timed
101     public ResponseEntity<WatchedEpisode> getWatchedEpisode(@PathVariable Long id) {
102         log.debug("REST request to get WatchedEpisode : {}", id);
103         Optional<WatchedEpisode> watchedEpisode = watchedEpisodeService.findById(id);
104         return ResponseUtil.wrapOrNotFound(watchedEpisode);
105     }
106
107     /**
108      * GET  /watched-episodes/episode/:id : get the watchedEpisode by "id".
109      *
110      * @param id the id of the episode in the watchedEpisode to retrieve
111      * @return the ResponseEntity with status 200 (OK) and with body the
         watchedEpisode, or with status 404 (Not Found)
112      */
113     @GetMapping("/watched-episodes/episode/{id}")
114     @Timed
115     public ResponseEntity<WatchedEpisode>
        getWatchedEpisodeForEpisodeId(@PathVariable Long id) {
116         log.debug("REST request to get WatchedEpisode by Episode: {}", id);
117         Optional<WatchedEpisode> watchedEpisode =
            watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(id);
118         return ResponseUtil.wrapOrNotFound(watchedEpisode);
119     }
120
121     /**
122      * GET  /watched-episodes/:id/average-rate : Get an average rate for episodes,
         which id is episodeId
123      *
124      * @param id the id of the watchedEpisode to check
125      * @return the ResponseEntity with status 200 (OK) and with body the
         watchedEpisode, or with status 404 (Not Found)
126      */
127     @GetMapping("/watched-episodes/{id}/average-rate")
128     @Timed
129     public Float getAverageRate(@PathVariable Long id) {
```

```
130             log.debug("REST request to get WatchedEpisode : {}", id);
131             return watchedEpisodeService.getAverageRate(id);
132         }
133 
134         /**
135          * GET  /watched-episodes/:id/rate-count : Get a count of how many users watched episode, which id is episodeId
136          *
137          * @param id the id of the watchedEpisode to retrieve
138          * @return the ResponseEntity with status 200 (OK) and with body the watchedEpisode, or with status 404 (Not Found)
139          */
140         @GetMapping("/watched-episodes/{id}/rate-count")
141         @Timed
142         public Integer getRateCount(@PathVariable Long id) {
143             log.debug("REST request to get WatchedEpisode : {}", id);
144             return watchedEpisodeService.getRateCount(id);
145         }
146 
147 
148         /**
149          * DELETE  /watched-episodes/:id : delete the watchedEpisode by "id".
150          *
151          * @param id the id of the watchedEpisode to delete
152          * @return the ResponseEntity with status 200 (OK)
153          */
154         @DeleteMapping("/watched-episodes/{id}")
155         @Timed
156         public ResponseEntity<Void> deleteWatchedEpisode(@PathVariable Long id) {
157             log.debug("REST request to delete WatchedEpisode : {}", id);
158 
159             watchedEpisodeService.deleteById(id);
160             return
                ResponseEntity.ok().headers(HeaderUtil.createEntityDeletionAlert(ENTITY_NAME,
                id.toString())).build();
161         }
162     }
163 
```

```java
//WatchedEpisodeService.java

package pl.marczynski.seriesapp.service;

import pl.marczynski.seriesapp.domain.WatchedEpisode;

import java.util.List;
import java.util.Optional;
/**
 * Service interface for WatchedEpisode.
 */
public interface WatchedEpisodeService {

    /**
     * Save WatchedEpisode
     * @param watchedEpisode WatchedEpisode to save
     * @return WatchedEpisode
     */
    WatchedEpisode save(WatchedEpisode watchedEpisode);

    /**
     * Find all watchedEpisode
     * @return List of watchedEpisode
     */
    List<WatchedEpisode> findAll();

    /**
     * Find WatchedEpisode by id
     * @param id the id of the WatchedEpisode to find
     * @return Optional of WatchedEpisode
     */
    Optional<WatchedEpisode> findById(Long id);

    /**
     * Delete WatchedEpisode by id
     * @param id the id of the WatchedEpisode to delete
     */
    void deleteById(Long id);

    /**
     * Update an existing WatchedEpisode
     *
     * @param watchedEpisode WatchedEpisode to update
     * @return WatchedEpisode
     */
    WatchedEpisode update(WatchedEpisode watchedEpisode);

    /**
     * Get an average rate for episodes, which id is episodeId
     * @param episodeId the id of the episode to check
     * @return Float
     */
    Float getAverageRate(Long episodeId);

    /**
     * Get a count of how many users watched episode, which id is episodeId
     * @param episodeId the id of the episode to check
     * @return Integer
     */
    Integer getRateCount(Long episodeId);

    /**
     * Find if current user watched episode, which id is id
     * @param id the id of the episode to check
     * @return Optional of WatchedEpisode
     */
    Optional<WatchedEpisode> getWatchedEpisodeForCurrentUserByEpisodeId(Long id);
}

```

```java
//WatchedEpisodeServiceImpl.java

package pl.marczynski.seriesapp.service.impl;

import org.springframework.stereotype.Service;
import pl.marczynski.seriesapp.domain.*;
import pl.marczynski.seriesapp.domain.builder.FollowedSeriesBuilder;
import pl.marczynski.seriesapp.domain.builder.WatchedEpisodeBuilder;
import pl.marczynski.seriesapp.repository.WatchedEpisodeRepository;
import pl.marczynski.seriesapp.security.AuthoritiesConstants;
import pl.marczynski.seriesapp.security.SecurityUtils;
import pl.marczynski.seriesapp.service.EpisodeService;
import pl.marczynski.seriesapp.service.FollowedSeriesService;
import pl.marczynski.seriesapp.service.UserService;
import pl.marczynski.seriesapp.service.WatchedEpisodeService;

import java.util.List;
import java.util.Optional;

/**
 * Service class for managing WatchedEpisode.
 */
@Service
public class WatchedEpisodeServiceImpl implements WatchedEpisodeService {
    private WatchedEpisodeRepository watchedEpisodeRepository;

    private FollowedSeriesService followedSeriesService;
    private UserService userService;
    private EpisodeService episodeService;

    public WatchedEpisodeServiceImpl(WatchedEpisodeRepository
    watchedEpisodeRepository, UserService userService, FollowedSeriesService
    followedSeriesService, EpisodeService episodeService) {
        this.watchedEpisodeRepository = watchedEpisodeRepository;
        this.userService = userService;
        this.followedSeriesService = followedSeriesService;
        this.episodeService = episodeService;
    }

    /**
     * Save WatchedEpisode
     *
     * @param watchedEpisode WatchedEpisode to save
     * @return WatchedEpisode
     */
    @Override
    public WatchedEpisode save(WatchedEpisode watchedEpisode) {
        if (watchedEpisode == null) return null;

        WatchedEpisode result = null;
        Optional<User> user = userService.findCurrentUser();
        Optional<WatchedEpisode> watchedEpisodeOptional =
        watchedEpisodeRepository.findByEpisodeIdAndUserIsCurrentUser(watchedEpisode.ge
        tEpisode().getId());
        watchedEpisodeOptional.ifPresent(watchedEpisode1 ->
        watchedEpisode.setId(watchedEpisode1.getId()));

        if (user.isPresent()) {

            watchedEpisode.setUser(user.get());
            result = watchedEpisodeRepository.save(watchedEpisode);
            Series series = watchedEpisode.getEpisode().getSeason().getSeries();
            FollowedSeries followedSeries = new
            FollowedSeriesBuilder().series(series).user(user.get()).build();
            followedSeriesService.save(followedSeries);
        }
        return result;
    }

    /**
     * Find all watchedEpisode
     *
     * @return List of watchedEpisode
```

```
68          */
69         @Override
70         public List<WatchedEpisode> findAll() {
71             if (SecurityUtils.isCurrentUserInRole(AuthoritiesConstants.ADMIN)) {
72                 return watchedEpisodeRepository.findAll();
73             } else {
74                 return watchedEpisodeRepository.findByUserIsCurrentUser();
75             }
76         }
77 
78         /**
79          * Find WatchedEpisode by id
80          *
81          * @param id the id of the WatchedEpisode to find
82          * @return Optional of WatchedEpisode
83          */
84         @Override
85         public Optional<WatchedEpisode> findById(Long id) {
86             return watchedEpisodeRepository.findById(id);
87         }
88 
89         /**
90          * Delete WatchedEpisode by id
91          *
92          * @param id the id of the WatchedEpisode to delete
93          */
94         @Override
95         public void deleteById(Long id) {
96             watchedEpisodeRepository.deleteById(id);
97         }
98 
99         /**
100          * Update an existing WatchedEpisode
101          *
102          * @param watchedEpisode WatchedEpisode to update
103          * @return WatchedEpisode
104          */
105         @Override
106         public WatchedEpisode update(WatchedEpisode watchedEpisode) {
107             return save(watchedEpisode);
108         }
109 
110         /**
111          * Get an average rate for episodes, which id is episodeId
112          *
113          * @param episodeId the id of the episode to check
114          * @return Float
115          */
116         @Override
117         public Float getAverageRate(Long episodeId) {
118             return watchedEpisodeRepository.getAverageRateByEpisodeId(episodeId);
119         }
120 
121         /**
122          * Get a count of how many users watched episode, which id is episodeId
123          *
124          * @param episodeId the id of the episode to check
125          * @return Integer
126          */
127         @Override
128         public Integer getRateCount(Long episodeId) {
129             return watchedEpisodeRepository.getRateCountByEpisodeId(episodeId);
130         }
131 
132         /**
133          * Find or create watchedEpisode for current user by episode id
134          *
135          * @param id the id of the episode to check
136          * @return Optional of WatchedEpisode
137          */
138         @Override
139         public Optional<WatchedEpisode> getWatchedEpisodeForCurrentUserByEpisodeId(Long
            id) {
```

```
140             if(id == null) return Optional.empty();
141
142             Optional<User> user = userService.findCurrentUser();
143             Optional<WatchedEpisode> result = Optional.empty();
144
145             if (user.isPresent()) {
146                 result = watchedEpisodeRepository.findByEpisodeIdAndUserIsCurrentUser(id);
147
148                 if (!result.isPresent()) {
149                     Optional<Episode> episode = episodeService.findById(id);
150
151                     if (episode.isPresent()) {
152                         result = Optional.of(new
                            WatchedEpisodeBuilder().user(user.get()).episode(episode.get()).bu
                            ild());
153                     }
154                 }
155             }
156             return result;
157         }
158     }
159
```

```java
//WatchedEpisodeRepository.java

package pl.marczynski.seriesapp.repository;

import org.springframework.data.repository.query.Param;
import pl.marczynski.seriesapp.config.Constants;
import pl.marczynski.seriesapp.domain.WatchedEpisode;
import org.springframework.data.jpa.repository.*;
import org.springframework.stereotype.Repository;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;
import java.util.List;
import java.util.Optional;

/**
 * Spring Data  repository for the WatchedEpisode entity.
 */
@SuppressWarnings("unused")
@Repository
public interface WatchedEpisodeRepository extends JpaRepository<WatchedEpisode,
Long> {

    /**
     * Find list of WatchedEpisode for current user
     * @return List of WatchedEpisode
     */
    @Query("select watched_episode from WatchedEpisode watched_episode where
    watched_episode.user.login = ?#{principal.username}")
    List<WatchedEpisode> findByUserIsCurrentUser();

    /**
     * Find if current user watched episode, which id is episodeId
     * @param episodeId the id of the Episode to check
     * @return Optional of WatchedEpisode
     */
    @Query("select watched_episode from WatchedEpisode watched_episode where
    watched_episode.user.login = ?#{principal.username} and
    watched_episode.episode.id = :episodeId")
    Optional<WatchedEpisode> findByEpisodeIdAndUserIsCurrentUser(@Param("episodeId")
    Long episodeId);

    /**
     * Get an average rate for episode, which id is episodeId
     * @param episodeId the id of the episode to check
     * @return Float
     */
    @Query("select avg(case watched_episode.rate " +
        "when 'BAD' then 1 " +
        "when 'MEDIOCRE' then 2 " +
        "when 'AVERAGE' then 3 " +
        "when 'GOOD' then 4 " +
        "when 'AWESOME' then 5 " +
        "else 0 END) from WatchedEpisode as watched_episode where
        watched_episode.episode.id = :episodeId and watched_episode.rate is not null")
    Float getAverageRateByEpisodeId(@Param("episodeId") Long episodeId);

    /**
     * Get a count of how many users watched episode, which id is episodeId
     * @param episodeId the id of the episode to check
     * @return Integer
     */
    @Query("select count(watched_episode) from WatchedEpisode as watched_episode
    where watched_episode.episode.id = :episodeId and watched_episode.rate is not
    null")
    Integer getRateCountByEpisodeId(@Param("episodeId") Long episodeId);
}

```

```java
//WatchedEpisode.java

package pl.marczynski.seriesapp.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;
import javax.validation.constraints.*;

import java.io.Serializable;
import java.util.Objects;

import pl.marczynski.seriesapp.domain.enumeration.Rate;

/**
 * A WatchedEpisode.
 */
@Entity
@Table(name = "watched_episode")
public class WatchedEpisode implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Enumerated(EnumType.STRING)
    @Column(name = "rate")
    private Rate rate;

    @Size(max = 500)
    @Column(name = "jhi_comment", length = 500)
    private String comment;

    @ManyToOne(optional = false)
    @NotNull
    @JsonIgnoreProperties("")
    private User user;

    @ManyToOne(optional = false)
    @NotNull
    @JsonIgnoreProperties("")
    private Episode episode;

    public WatchedEpisode() {
    }

    public WatchedEpisode(Rate rate, String comment, User user, Episode episode) {
        this.rate = rate;
        this.comment = comment;
        this.user = user;
        this.episode = episode;
    }

    // jhipster-needle-entity-add-field - JHipster will add fields here, do not remove
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Rate getRate() {
        return rate;
    }

    public void setRate(Rate rate) {
        this.rate = rate;
    }

    public String getComment() {
```

```
74          return comment;
75      }
76
77      public void setComment(String comment) {
78          this.comment = comment;
79      }
80
81      public User getUser() {
82          return user;
83      }
84
85      public void setUser(User user) {
86          this.user = user;
87      }
88
89      public Episode getEpisode() {
90          return episode;
91      }
92
93      public void setEpisode(Episode episode) {
94          this.episode = episode;
95      }
96      // jhipster-needle-entity-add-getters-setters - JHipster will add getters and
        setters here, do not remove
97
98      @Override
99      public boolean equals(Object o) {
100         if (this == o) {
101             return true;
102         }
103         if (o == null || getClass() != o.getClass()) {
104             return false;
105         }
106         WatchedEpisode watchedEpisode = (WatchedEpisode) o;
107         if (watchedEpisode.getId() == null || getId() == null) {
108             return false;
109         }
110         return Objects.equals(getId(), watchedEpisode.getId());
111     }
112
113     @Override
114     public int hashCode() {
115         return Objects.hashCode(getId());
116     }
117
118     @Override
119     public String toString() {
120         return "WatchedEpisode{" +
121             "id=" + getId() +
122             ", rate='" + getRate() + "'" +
123             ", comment='" + getComment() + "'" +
124             "}";
125     }
126 }
127
```

All Classes

Packages

pl.marczynski.seriesapp
pl.marczynski.seriesapp.aop.lo
pl.marczynski.seriesapp.config
pl.marczynski.seriesapp.config
pl.marczynski.seriesapp.config
pl.marczynski.seriesapp.doma
pl.marczynski.seriesapp.doma
pl.marczynski.seriesapp.doma
pl.marczynski.seriesapp.doma
pl.marczynski.seriesapp.repos
pl.marczynski.seriesapp.repos
pl.marczynski.seriesapp.repos
pl.marczynski.seriesapp.secur
pl.marczynski.seriesapp.secur

All Classes

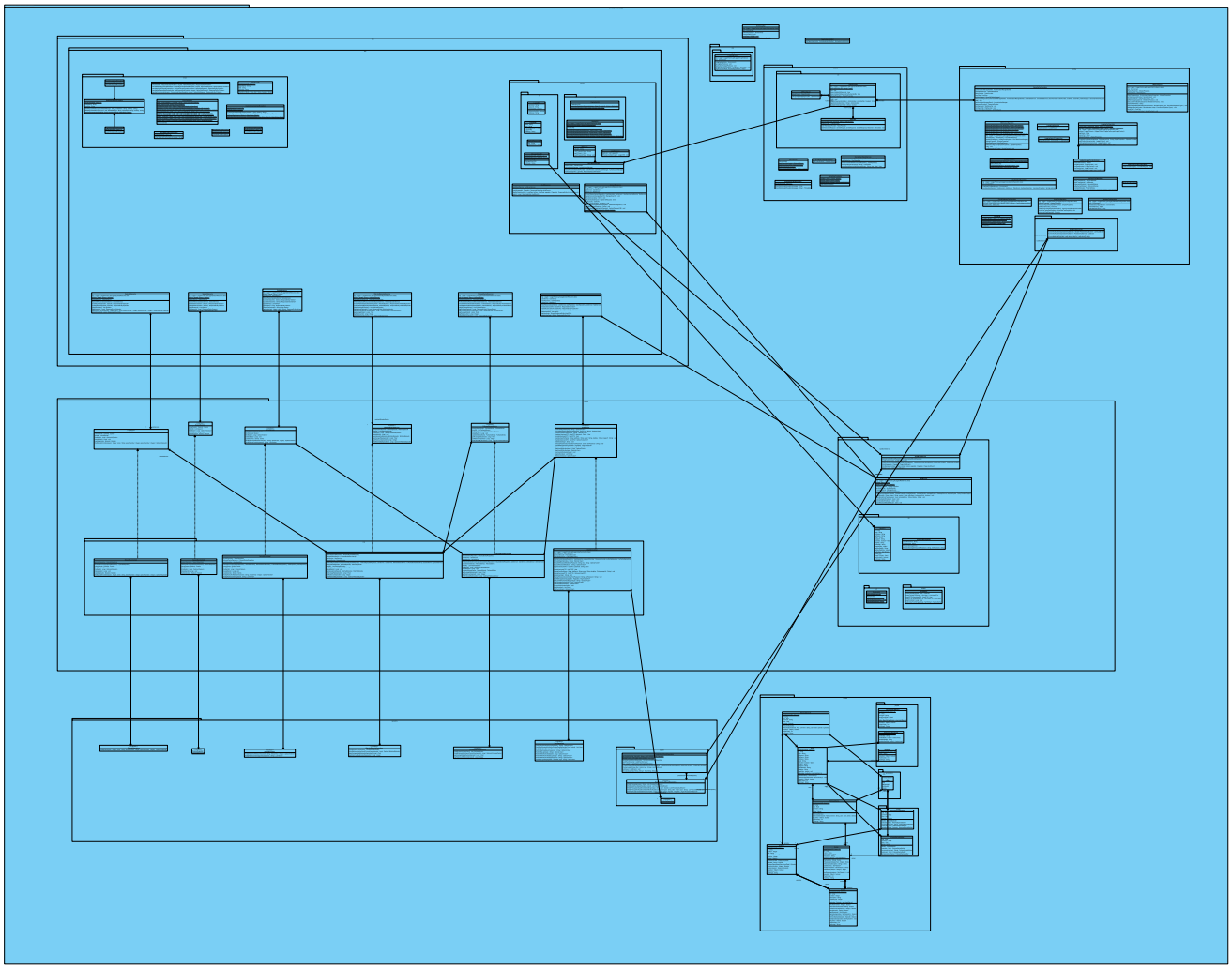
AbstractAuditingEntity
AccountResource
AccountResourceIntTest
ApplicationProperties
ApplicationWebXml
AsyncConfiguration
AuditEventConverter
AuditEventService
AuditResource
AuditResourceIntTest
AuthoritiesConstants
Authority
AuthorityRepository
BadRequestAlertException
CloudDatabaseConfiguration
Constants
CustomAuditEventRepository
CustomAuditEventRepositoryI
CustomParameterizedExceptic
DatabaseConfiguration
DateTimeFormatConfiguration
DateTimeWrapper
DateTimeWrapperRepository
DefaultProfileUtil
DomainUserDetailsService
DomainUserDetailsServiceIntT
EmailAlreadyUsedException
EmailNotFoundException
Episode
EpisodeRepository
EpisodeResource
EpisodeResourceIntTest
EpisodeService
EpisodeServiceImpl
ErrorConstants
ExceptionTranslator
ExceptionTranslatorIntTest
ExceptionTranslatorTestContro
ExceptionTranslatorTestContro
ExceptionTranslatorTestContro
FieldErrorVM
FollowedSeries

Packages

Package	Description
pl.marczynski.seriesapp	Package that stores all files for series app.
pl.marczynski.seriesapp.aop.logging	Aspect for logging execution of service and repository Spring components.
pl.marczynski.seriesapp.config	Spring Framework configuration files.
pl.marczynski.seriesapp.config.audit	Audit specific code.
pl.marczynski.seriesapp.config.timezone	Unit tests for the UTC Hibernate configuration.
pl.marczynski.seriesapp.domain	JPA domain objects.
pl.marczynski.seriesapp.domain.builder	Builder pattern for JPA domain objects.
pl.marczynski.seriesapp.domain.enumeration	Enumeration for domain objects.
pl.marczynski.seriesapp.domain.jhipster	Automatically generated domain objects.
pl.marczynski.seriesapp.repository	Spring Data JPA repositories.
pl.marczynski.seriesapp.repository.jhipster	Automatically generated Spring Data JPA repositories.
pl.marczynski.seriesapp.repository.timezone	Spring Data JPA repository for the DateTime.
pl.marczynski.seriesapp.security	Spring Security configuration.
pl.marczynski.seriesapp.security.jwt	Spring Security configuration for JSON Web Token.
pl.marczynski.seriesapp.service	Service layer beans.
pl.marczynski.seriesapp.service.impl	Services implementation.
pl.marczynski.seriesapp.service.jhipster	Automatically generated services.
pl.marczynski.seriesapp.service.jhipster.dto	Data Transfer Objects.
pl.marczynski.seriesapp.service.jhipster.mapper	MapStruct mappers for mapping domain objects and Data Transfer Objects.
pl.marczynski.seriesapp.service.jhipster.util	Automatically generated utility classes.
pl.marczynski.seriesapp.web.rest	Automatically generated Spring MVC REST controllers.
pl.marczynski.seriesapp.web.rest.errors	Specific errors used with Zalando's "problem-spring-web" library.
pl.marczynski.seriesapp.web.rest.jhipster	Spring MVC REST controllers.
pl.marczynski.seriesapp.web.rest.jhipster.util	Automatically generated utility classes for REST controllers.
pl.marczynski.seriesapp.web.rest.jhipster.vm	View Models used by Spring MVC REST controllers.

(10)
Realizacja PU

1. Diagram klas - realizacja



1.1. AbstractAuditingEntity

Base abstract class for entities which will hold definitions for created, last modified by and created,
last modified by date.

1.2. AccountResource

REST controller for managing the current user's account.

1.3. aop

1.4. ApplicationProperties

Properties specific to Seriesapp.
<p>
Properties are configured in the application.yml file.
See {@link io.github.jhipster.config.JHipsterProperties} for a good example.

1.5. ApplicationWebXml

This is a helper Java class that provides an alternative to creating a web.xml.
This will be invoked only when the application is deployed to a Servlet container like Tomcat, JBoss etc.

1.6. AsyncConfiguration

1.7. audit

1.8. AuditEventConverter

1.9. AuditEventService
Service for managing audit events.
<p>
This is the default implementation to support SpringBoot Actuator AuditEventRepository

1.10. AuditResource
REST controller for getting the audit events.

1.11. AuthoritiesConstants
Constants for Spring Security authorities.

1.12. Authority
An authority (a security role) used by Spring Security.

1.13. AuthorityRepository
Spring Data JPA repository for the Authority entity.

1.14. BadRequestAlertException

1.15. builder

1.16. CloudDatabaseConfiguration

1.17. config

1.18. Constants
Application constants.

1.19. CustomAuditEventRepository
An implementation of Spring Boot's AuditEventRepository.

1.20. CustomParameterizedException
Custom, parameterized exception, which can be translated on the client side.
For example:

<pre>
throw new CustomParameterizedException("myCustomError", "hello", "world");
</pre>

Can be translated with:

<pre>
"error.myCustomError" : "The server says {{param0}} to {{param1}}"
</pre>

1.21. DatabaseConfiguration

1.22. DateTimeFormatConfiguration
Configure the converters to use the ISO format for dates by default.

1.23. DefaultProfileUtil
Utility class to load a Spring profile to be used as default
when there is no <code>spring.profiles.active</code> set in the environment or as command line argument.
If the value is not available in <code>application.yml</code> then <code>dev</code> profile will be used as default.

1.24. domain

1.25. DomainUserDetailsService
Authenticate a user from the database.

1.26. dto

1.27. EmailAlreadyUsedException

1.28. EmailNotFoundException

1.29. enumeration

1.30. Episode
A Episode.

1.31. EpisodeRepository
Spring Data repository for the Episode entity.

1.32. EpisodeResource
REST controller for managing Episode.

1.33. EpisodeService
Service interface for episodes.

1.34. EpisodeServiceImpl
Service class for managing episodes.

1.35. ErrorConstants

1.36. errors

1.37. ExceptionTranslator
Controller advice to translate the server side exceptions to client-friendly json structures.
The error response follows RFC7807 - Problem Details for HTTP APIs (https://tools.ietf.org/html/rfc7807)

1.38. FieldErrorVM

1.39. FollowedSeries
A FollowedSeries.

1.40. FollowedSeriesBuilder

1.41. FollowedSeriesRepository
Spring Data repository for the FollowedSeries entity.

1.42. FollowedSeriesResource
REST controller for managing FollowedSeries.

1.43. FollowedSeriesService
Service interface for FollowedSeries.

1.44. FollowedSeriesServiceImpl

1.45. HeaderUtil
Utility class for HTTP headers creation.

1.46. impl

1.47. InternalServerErrorException
Simple exception with a message, that returns an Internal Server Error code.

1.48. InvalidPasswordException

1.49. JacksonConfiguration

1.50. jhipster

1.51. jhipster

1.52. jhipster

1.53. jhipster

1.54. jwt

1.55. JWTConfigurer

1.56. JWTFilter
Filters incoming requests and installs a Spring Security principal if a header corresponding to a valid user is
found.

1.57. JWTToken
Object to return as body in JWT Authentication.

1.58. KeyAndPasswordVM
View Model object for storing the user's key and password.

1.59. LiquibaseConfiguration

1.60. LocaleConfiguration

1.61. LogbackLoggerContextListener
Logback configuration is achieved by configuration file and API.
When configuration file change is detected, the configuration is reset.
This listener ensures that the programmatic configuration is also re-applied after reset.

1.62. LoggerVM
View Model object for storing a Logback logger.

1.63. logging

1.64. LoggingAspect
Aspect for logging execution of service and repository Spring components.

By default, it only runs with the "dev" profile.

1.65. LoggingAspectConfiguration

1.66. LoggingConfiguration

1.67. LoginAlreadyUsedException

1.68. LoginVM
View Model object for storing a user's credentials.

1.69. LogsResource
Controller for view and managing Log Level at runtime.

1.70. MailService
Service for sending emails.
<p>
We use the @Async annotation to send emails asynchronously.

1.71. ManagedUserVM
View Model extending the UserDTO, which is meant to be used in the user management UI.

1.72. mapper

1.73. MetricsConfiguration

1.74. PaginationUtil
Utility class for handling pagination.

<p>
Pagination uses the same principles as the GitHub API,
and follow RFC 5988 (Link header).

1.75. PasswordChangeDTO
A DTO representing a password change required data - current and new password.

1.76. PersistenceAuditEventRepository
Spring Data JPA repository for the PersistentAuditEvent entity.

1.77. PersistentAuditEvent
Persist AuditEvent managed by the Spring Boot actuator.

@see org.springframework.boot.actuate.audit.AuditEvent

1.78. pl.marczynski.seriesapp

1.79. RandomUtil
Utility class for generating random Strings.

1.80. Rate
The Rate enumeration.

1.81. repository

1.82. rest

1.83. Season
A Season.

1.84. SeasonRepository
Spring Data repository for the Season entity.

1.85. SeasonResource
REST controller for managing Season.

1.86. SeasonService
Service interface for Season.

1.87. SeasonServiceImpl

1.88. security

1.89. SecurityConfiguration

1.90. SecurityUtils
Utility class for Spring Security.

1.91. Series
A Series.

1.92. SeriesappApp

1.93. SeriesRepository
Spring Data repository for the Series entity.

1.94. SeriesResource
REST controller for managing Series.

1.95. SeriesService
Service interface for Series.

1.96. SeriesServiceImpl

1.97. service

1.98. SpringSecurityAuditorAware
Implementation of AuditorAware based on Spring Security.

1.99. TokenProvider

1.100. User
A user.

1.101. UserDTO
A DTO representing a user, with his authorities.

1.102. UserJWTController
Controller to authenticate users.

1.103. UserMapper
Mapper for the entity User and its DTO called UserDTO.

Normal mappers are generated using MapStruct, this one is hand-coded as MapStruct support is still in beta, and requires a manual step with an IDE.

1.104. UserNotActivatedException
This exception is thrown in case of a not activated user trying to authenticate.

1.105. UserRepository
Spring Data JPA repository for the User entity.

1.106. UserResource
REST controller for managing users.
<p>
This class accesses the User entity, and needs to fetch its collection of authorities.
<p>
For a normal use-case, it would be better to have an eager relationship between User and Authority,
and send everything to the client side: there would be no View Model and DTO, a lot less code, and an outer-join
which would be good for performance.
<p>
We use a View Model and a DTO for 3 reasons:

We want to keep a lazy association between the user and the authorities, because people will
quite often do relationships with the user, and we don't want them to get the authorities all the time for nothing (for performance reasons). This is the #1 goal: we should not impact our users'
application because of this use-case.
 Not having an outer join causes n+1 requests to the database. This is not a real issue as we have by default a second-level cache. This means on the first HTTP call we do the n+1 requests,

but then all authorities come from the cache, so in fact it's much better than doing an outer join
(which will get lots of data from the database, for each HTTP call).
 As this manages users, for security reasons, we'd rather have a DTO layer.

<p>
Another option would be to have a specific JPA entity graph to handle this case.

1.107. UserService
Service interface for User.

1.108. UserServiceImpl
Service class for managing users.

1.109. util

1.110. util

1.111. vm

1.112. WatchedEpisode
A WatchedEpisode.

1.113. WatchedEpisodeBuilder

1.114. WatchedEpisodeRepository
Spring Data repository for the WatchedEpisode entity.

1.115. WatchedEpisodeResource
REST controller for managing WatchedEpisode.

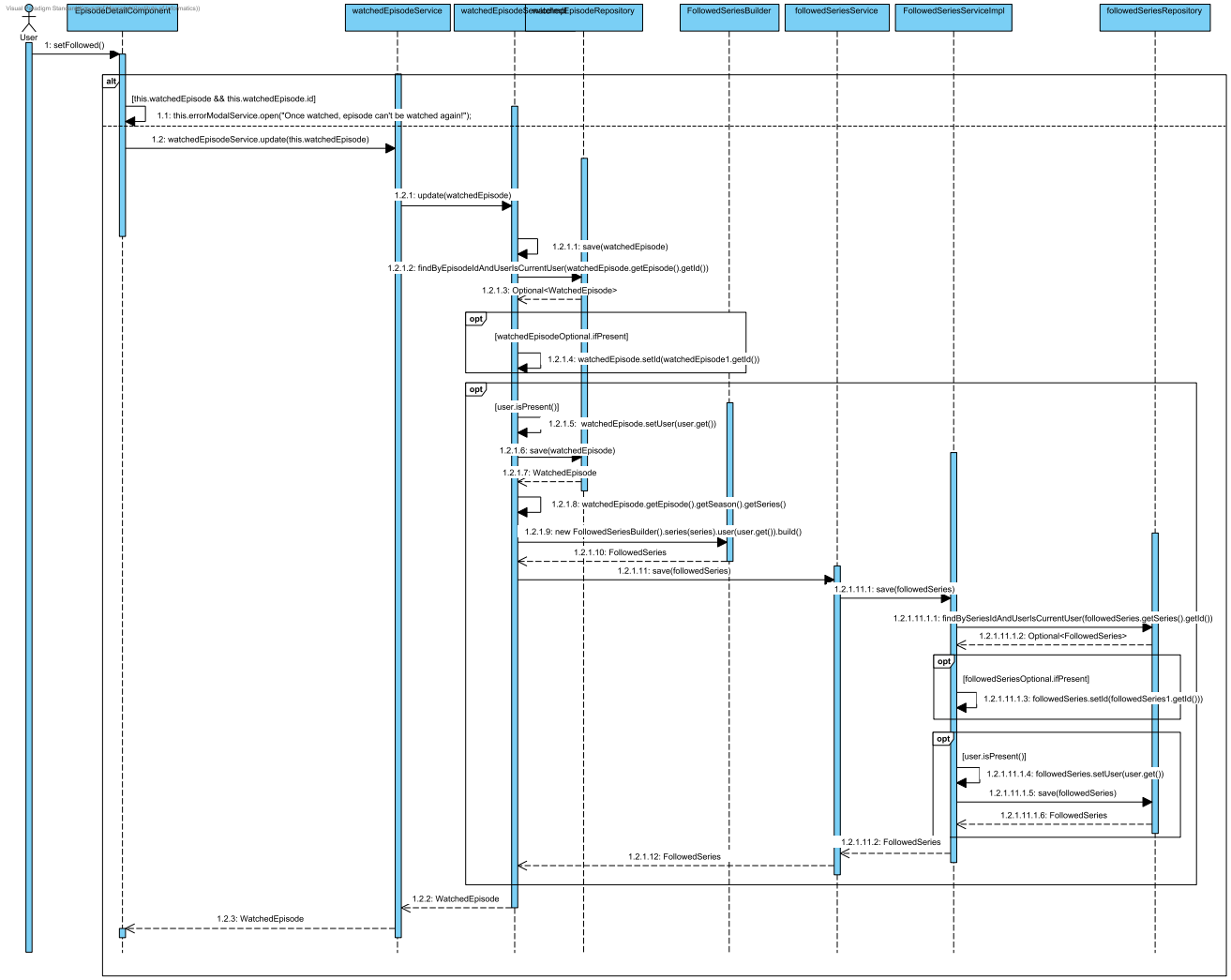
1.116. WatchedEpisodeService
Service interface for WatchedEpisode.

1.117. WatchedEpisodeServiceImpl
Service class for managing WatchedEpisode.

1.118. web

1.119. WebConfigurer
Configuration of web application with Servlet 3.0 APIs.

1. Realizacja - oznaczenie odcinka jako obejrzanego



1.1. CombinedFragment

1.2. CombinedFragment2

1.3. CombinedFragment3

1.4. CombinedFragment4

1.5. CombinedFragment5

1.6. EpisodeDetailComponent

1.7. FollowedSeriesBuilder

1.8. followedSeriesRepository

1.9. followedSeriesService

1.10. FollowedSeriesServiceImpl

1.11. User

1.12. watchedEpisodeRepository

1.13. watchedEpisodeService

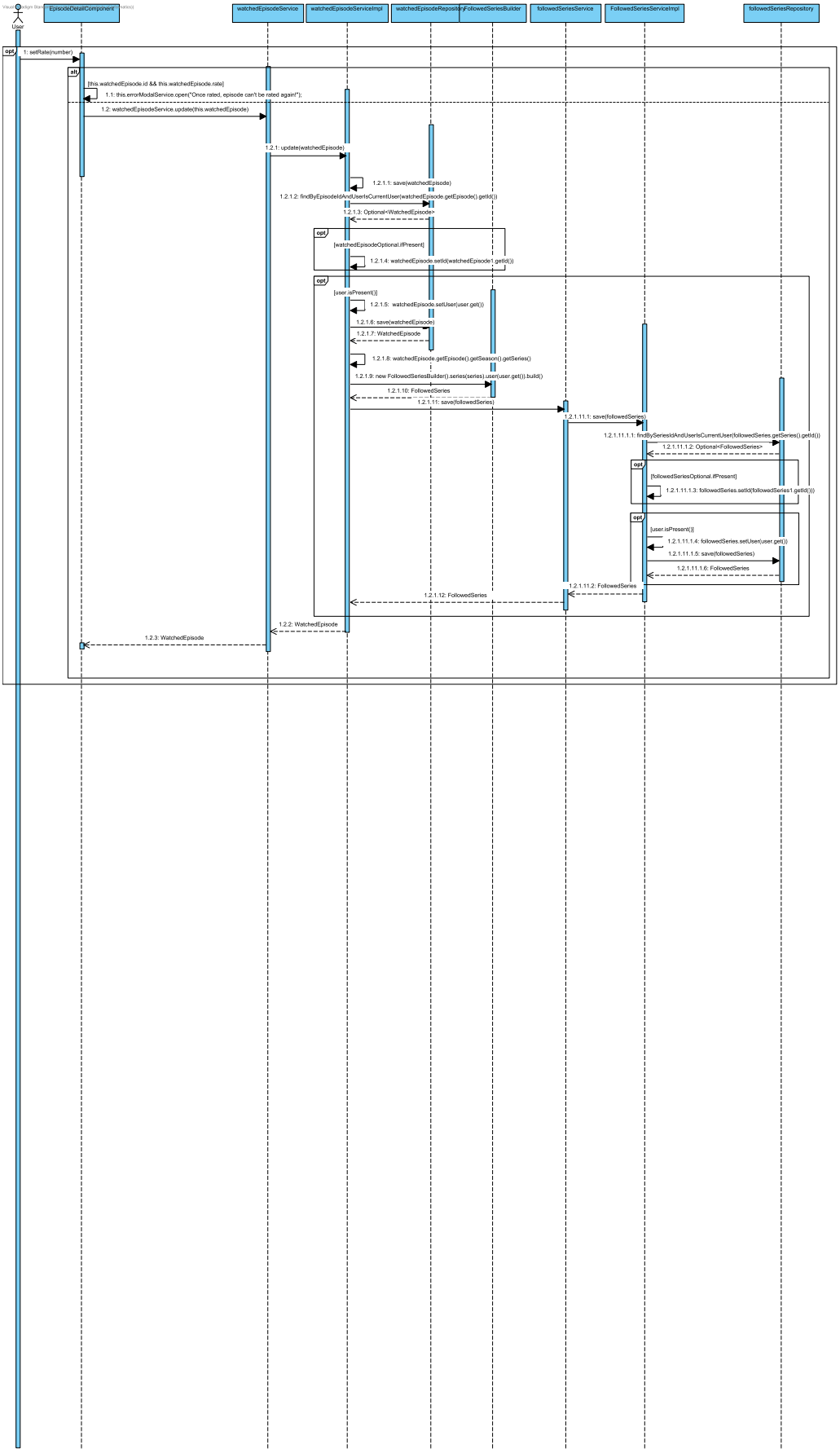
1.14. watchedEpisodeServiceImpl

1.15. Messages

From	No.	Name	Type	Action Type	To	Async.
User	1	setFollowed()	→	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.1	this.errorModalService.open("Once watched, episode can't be watched again!");	↩	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.2	watchedEpisodeService.update(this.watchedEpisode)	→	Unspecified	watchedEpisodeService	
watchedEpisodeService	1.2.1	update(watchedEpisode)	→	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.1	save(watchedEpisode)	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.2	findByEpisodeIdAndUserIsCurrentUser(watchedEpisode.getEpisode().getId())	→	Unspecified	watchedEpisodeRepository	
watchedEpisodeRepository	1.2.1.3	Optional<WatchedEpisode>	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.4	watchedEpisode.setId(watchedEpisode1.getId())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.5	watchedEpisode.setUser(user.get())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.6	save(watchedEpisode)	→	Unspecified	watchedEpisodeRepository	
watchedEpisodeServiceImpl	1.2.1.8	watchedEpisode.getEpisode().getSeason().getSeries()	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeRepository	1.2.1.7	WatchedEpisode	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.9	new FollowedSeriesBuilder().series(series).user(user.get()).build()	→	Unspecified	FollowedSeriesBuilder	
FollowedSeriesBuilder	1.2.1.10	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	

From	No.	Name	Type	Action Type	To	Async.
watchedEpisodeServiceImpl	1.2.1.11	save(followedSeries)	→	Unspecified	followedSeriesService	
followedSeriesService	1.2.1.11.1	save(followedSeries)	→	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.1	findBySeriesIdAndUserIsCurrentUser(followedSeries.getSeries().getId())	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.2	Optional<FollowedSeries>	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.3	followedSeries.setId(followedSeries1.getId()))		Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.4	followedSeries.setUser(user.get())		Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.5	save(followedSeries)	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.6	FollowedSeries	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.2	FollowedSeries	→	Reply	followedSeriesService	
followedSeriesService	1.2.1.12	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.2	WatchedEpisode	→	Reply	watchedEpisodeService	
watchedEpisodeService	1.2.3	WatchedEpisode	→	Reply	EpisodeDetailComponent	

1. Realizacja - wyrazenie opinii o odcinku (ocena)



1.1. CombinedFragment

1.2. CombinedFragment2

1.3. CombinedFragment3

1.4. CombinedFragment4

1.5. CombinedFragment5

1.6. CombinedFragment6

1.7. EpisodeDetailComponent

1.8. FollowedSeriesBuilder

1.9. followedSeriesRepository

1.10. followedSeriesService

1.11. FollowedSeriesServiceImpl

1.12. User

1.13. watchedEpisodeRepository

1.14. watchedEpisodeService

1.15. watchedEpisodeServiceImpl

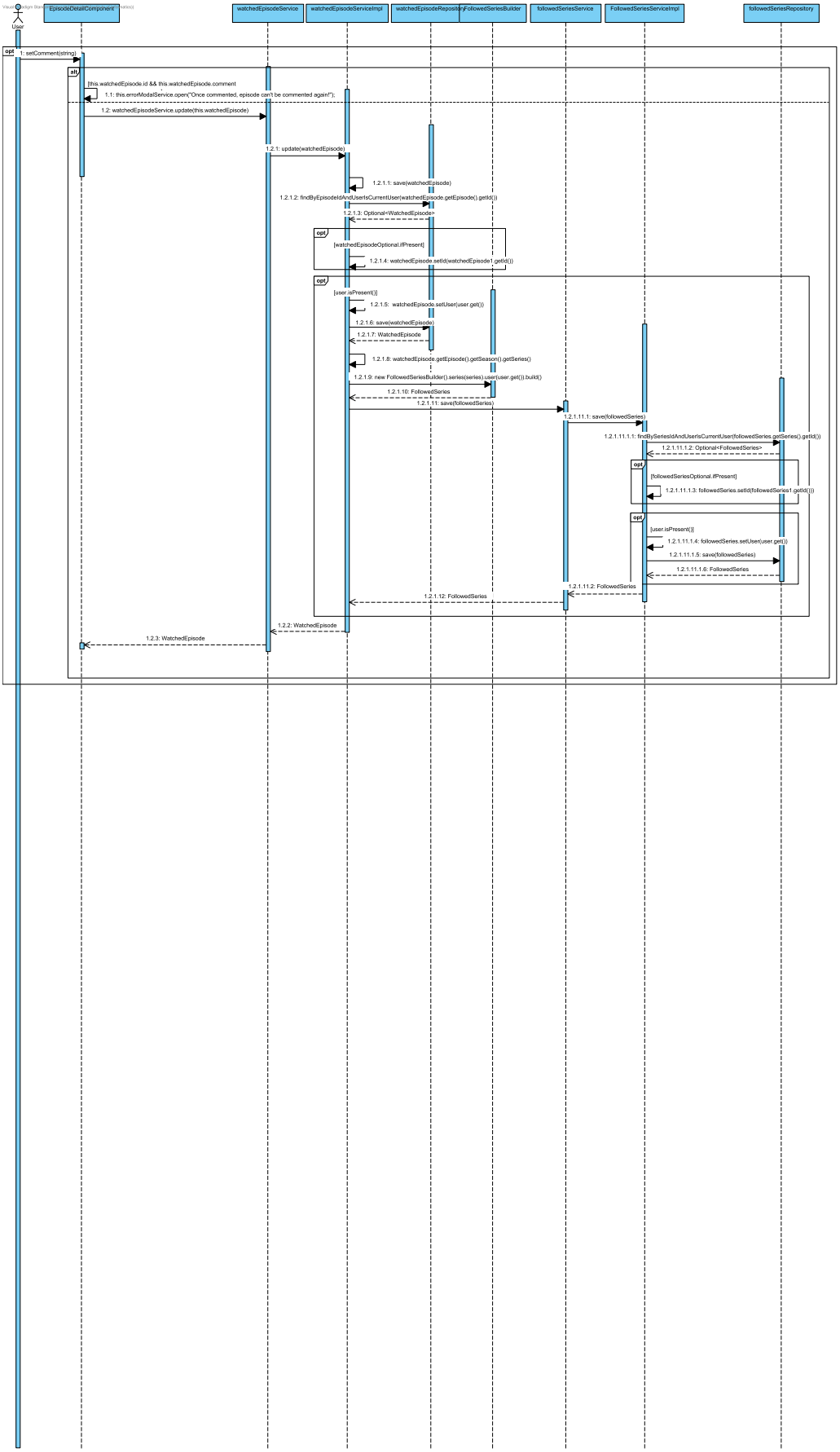
1.16. Messages

From	No.	Name	Type	Action Type	To	Async.
User	1	setRate(number)	→	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.1	this.errorModalService.open("Once rated, episode can't be rated again!");	↩	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.2	watchedEpisodeService.update(this.watchedEpisode)	→	Unspecified	watchedEpisodeService	
watchedEpisodeService	1.2.1	update(watchedEpisode)	→	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.1	save(watchedEpisode)	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.2	findByEpisodeIdAndUserIsCurrentUser(watchedEpisode.getEpisode().getId())	→	Unspecified	watchedEpisodeRepository	

From	No.	Name	Type	Action Type	To	Async.
watchedEpisodeRepository	1.2.1.3	Optional<WatchedEpisode>	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.4	watchedEpisode.setId(watchedEpisode1.getId())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.5	watchedEpisode.setUser(user.get())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.6	save(watchedEpisode)	→	Unspecified	watchedEpisodeRepository	
watchedEpisodeRepository	1.2.1.7	WatchedEpisode	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.8	watchedEpisode.getEpisode().getSeason().getSeries()	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.9	new FollowedSeriesBuilder().series(series).user(user.get()).build()	→	Unspecified	FollowedSeriesBuilder	
FollowedSeriesBuilder	1.2.1.10	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.11	save(followedSeries)	→	Unspecified	followedSeriesService	
followedSeriesService	1.2.1.11.1	save(followedSeries)	→	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.1	findBySeriesIdAndUserIsCurrentUser(followedSeries.getSeries().getId())	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.2	Optional<FollowedSeries>	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.3	followedSeries.setId(followedSeries1.getId()))	↩	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.4	followedSeries.setUser(user.get())	↩	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.5	save(followedSeries)	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.6	FollowedSeries	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.2	FollowedSeries	→	Reply	followedSeriesService	
followedSeriesService	1.2.1.12	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.2	WatchedEpisode	→	Reply	watchedEpisodeService	

From	No.	Name	Type	Action Type	To	Async.
watchedEpisodeService	1.2.3	WatchedEpisode	→	Reply	EpisodeDetailComponent	

1. Realizacja - wyra#enie opinii o odcinku (komentarz)



1.1. CombinedFragment

1.2. CombinedFragment2

1.3. CombinedFragment3

1.4. CombinedFragment4

1.5. CombinedFragment5

1.6. CombinedFragment6

1.7. EpisodeDetailComponent

1.8. FollowedSeriesBuilder

1.9. followedSeriesRepository

1.10. followedSeriesService

1.11. FollowedSeriesServiceImpl

1.12. User

1.13. watchedEpisodeRepository

1.14. watchedEpisodeService

1.15. watchedEpisodeServiceImpl

1.16. Messages

From	No.	Name	Type	Action Type	To	Async.
User	1	setComment(string)	→	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.1	this.errorModalService.open("Once commented, episode can't be commented again!");	↩	Unspecified	EpisodeDetailComponent	
EpisodeDetailComponent	1.2	watchedEpisodeService.update(this.watchedEpisode)	→	Unspecified	watchedEpisodeService	
watchedEpisodeService	1.2.1	update(watchedEpisode)	→	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.1	save(watchedEpisode)	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.2	findByEpisodeIdAndUserIsCurrentUser(watchedEpisode.getEpisode().getId())	→	Unspecified	watchedEpisodeRepository	

From	No.	Name	Type	Action Type	To	Async.
watchedEpisodeRepository	1.2.1.3	Optional<WatchedEpisode>	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.4	watchedEpisode.setId(watchedEpisode1.getId())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.5	watchedEpisode.setUser(user.get())	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.6	save(watchedEpisode)	→	Unspecified	watchedEpisodeRepository	
watchedEpisodeRepository	1.2.1.7	WatchedEpisode	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.8	watchedEpisode.getEpisode().getSeason().getSeries()	↩	Unspecified	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.9	new FollowedSeriesBuilder().series(series).user(user.get()).build()	→	Unspecified	FollowedSeriesBuilder	
FollowedSeriesBuilder	1.2.1.10	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.1.11	save(followedSeries)	→	Unspecified	followedSeriesService	
followedSeriesService	1.2.1.11.1	save(followedSeries)	→	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.1	findBySeriesIdAndUserIsCurrentUser(followedSeries.getSeries().getId())	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.2	Optional<FollowedSeries>	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.3	followedSeries.setId(followedSeries1.getId()))	↩	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.4	followedSeries.setUser(user.get())	↩	Unspecified	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.1.5	save(followedSeries)	→	Unspecified	followedSeriesRepository	
followedSeriesRepository	1.2.1.11.1.6	FollowedSeries	→	Reply	FollowedSeriesServiceImpl	
FollowedSeriesServiceImpl	1.2.1.11.2	FollowedSeries	→	Reply	followedSeriesService	
followedSeriesService	1.2.1.12	FollowedSeries	→	Reply	watchedEpisodeServiceImpl	
watchedEpisodeServiceImpl	1.2.2	WatchedEpisode	→	Reply	watchedEpisodeService	

From	No.	Name	Type	Action Type	To	Async.
watchedEpisodeService	1.2.3	WatchedEpisode	→	Reply	EpisodeDetailComponent	

(11)
Testy jednostkowe

```java
//WatchedEpisodeServiceImplTest.java

package pl.marczynski.seriesapp.service.impl;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import org.mockito.stubbing.Answer;
import pl.marczynski.seriesapp.domain.*;
import pl.marczynski.seriesapp.domain.builder.FollowedSeriesBuilder;
import pl.marczynski.seriesapp.domain.builder.WatchedEpisodeBuilder;
import pl.marczynski.seriesapp.repository.WatchedEpisodeRepository;
import pl.marczynski.seriesapp.service.EpisodeService;
import pl.marczynski.seriesapp.service.FollowedSeriesService;
import pl.marczynski.seriesapp.service.UserService;

import java.util.Optional;

import static org.hamcrest.Matchers.is;
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class WatchedEpisodeServiceImplTest {

    @Mock
    private WatchedEpisodeRepository watchedEpisodeRepository;

    @Mock
    private FollowedSeriesService followedSeriesService;

    @Mock
    private UserService userService;

    @Mock
    private EpisodeService episodeService;

    @InjectMocks
    private WatchedEpisodeServiceImpl watchedEpisodeService;

    private static final long DEFAULT_ID = 1L;

    @Before
    public void setup() {
        ArgumentCaptor<WatchedEpisode> watchedEpisode =
        ArgumentCaptor.forClass(WatchedEpisode.class);
        when(watchedEpisodeRepository.save(watchedEpisode.capture())).thenAnswer(
            (Answer) invocation -> {
                WatchedEpisode captured = watchedEpisode.getValue();
                if (captured == null) return null;

                if (captured.getId() == null) {
                    captured.setId(DEFAULT_ID);
                }
                return captured;
            });
    }

    // ****************************************** Tests for save method
    ******************************************

    @Test
    public void whenLoggedUserNotPresentThenSaveResultIsNull() {
        //given
        Episode episode = new Episode();
        episode.setId(DEFAULT_ID);
        WatchedEpisode watchedEpisode = new
        WatchedEpisodeBuilder().episode(episode).build();

```

```
71          when(userService.findCurrentUser()).thenReturn(Optional.empty());
72
73          //when
74          WatchedEpisode result = watchedEpisodeService.save(watchedEpisode);
75
76          //then
77          assertNull(result);
78      }
79
80      @Test
81      public void whenLoggedUserIsPresentThenSaveWatchedEpisodeInRepository() {
82          //given
83          User user = new User();
84          user.setId(DEFAULT_ID);
85          WatchedEpisode watchedEpisode = createWatchedEpisode();
86          ArgumentCaptor<WatchedEpisode> watchedEpisodeCaptor =
            ArgumentCaptor.forClass(WatchedEpisode.class);
87
88          when(userService.findCurrentUser()).thenReturn(Optional.of(user));
89
90          //when
91          watchedEpisodeService.save(watchedEpisode);
92
93          //then
94          verify(watchedEpisodeRepository,
            times(1)).save(watchedEpisodeCaptor.capture());
95          WatchedEpisode result = watchedEpisodeCaptor.getValue();
96          assertThat(result.getUser(), is(user));
97          assertThat(result.getEpisode(), is(watchedEpisode.getEpisode()));
98          assertThat(result.getComment(), is(watchedEpisode.getComment()));
99          assertThat(result.getRate(), is(watchedEpisode.getRate()));
100     }
101
102     @Test
103     public void whenSavingWatchedEpisodeThenSetSeriesAsFollowed() {
104         //given
105         User user = new User();
106         user.setId(DEFAULT_ID);
107         WatchedEpisode watchedEpisode = createWatchedEpisode();
108
109         when(userService.findCurrentUser()).thenReturn(Optional.of(user));
110         ArgumentCaptor<FollowedSeries> capturedFollowed =
            ArgumentCaptor.forClass(FollowedSeries.class);
111         FollowedSeries expected = new
            FollowedSeriesBuilder().series(watchedEpisode.getEpisode().getSeason().getSeri
            es()).user(user).build();
112
113         //when
114         watchedEpisodeService.save(watchedEpisode);
115
116         //then
117         verify(followedSeriesService, times(1)).save(capturedFollowed.capture());
118         FollowedSeries result = capturedFollowed.getValue();
119         assertThat(result.getUser(), is(expected.getUser()));
120         assertThat(result.getSeries(), is(expected.getSeries()));
121     }
122
123     @Test
124     public void whenSavingWatchedEpisodeAndIdIsPresentThenIdShouldRemain() {
125         //given
126         User user = new User();
127         user.setId(DEFAULT_ID);
128         WatchedEpisode watchedEpisode = createWatchedEpisode();
129
130         watchedEpisode.setId(DEFAULT_ID);
131         when(userService.findCurrentUser()).thenReturn(Optional.of(user));
132
133         //when
134         WatchedEpisode result = watchedEpisodeService.save(watchedEpisode);
135
136         //then
137         assertThat(result.getId(), is(DEFAULT_ID));
138     }
```

```
139
140         @Test
141         public void whenSavingWatchedEpisodeAndIdIsNotPresentThenIdShouldBeAdded() {
142             //given
143             User user = new User();
144             user.setId(DEFAULT_ID);
145             WatchedEpisode watchedEpisode = createWatchedEpisode();
146             watchedEpisode.setId(null);
147             when(userService.findCurrentUser()).thenReturn(Optional.of(user));
148
149             //when
150             WatchedEpisode result = watchedEpisodeService.save(watchedEpisode);
151
152             //then
153             assertNotNull(result.getId());
154         }
155
156         @Test
157         public void
            whenSavingWatchedEpisodeThenCurrentUserShouldBeAddedToWatchedEpisode() {
158             //given
159             User user = new User();
160             user.setId(DEFAULT_ID);
161             WatchedEpisode watchedEpisode = createWatchedEpisode();
162             watchedEpisode.setUser(null);
163             when(userService.findCurrentUser()).thenReturn(Optional.of(user));
164
165             //when
166             WatchedEpisode result = watchedEpisodeService.save(watchedEpisode);
167
168             //then
169             assertThat(result.getUser(), is(user));
170         }
171
172         @Test
173         public void
            whenSavingWatchedEpisodeAndWatchedEpisodeIsNullThenShouldReturnNull() {
174             //given
175
176             //when
177             WatchedEpisode result = watchedEpisodeService.save(null);
178
179             //then
180             verify(episodeService, times(0)).findById(any());
181             assertNull(result);
182         }
183
184         // ****************************************** Tests for
            getWatchedEpisodeForCurrentUserByEpisodeId method
            ******************************************
185
186         @Test
187         public void
            whenUserIsNotLoggedThenGetWatchedEpisodeForCurrentUserByEpisodeIdShouldReturnEmpty
            Optional() {
188             //given
189             when(userService.findCurrentUser()).thenReturn(Optional.empty());
190
191             //when
192             Optional<WatchedEpisode> result =
                watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(DEFAULT_ID);
193
194             //then
195             verify(userService, times(1)).findCurrentUser();
196             assertThat(result.isPresent(), is(false));
197         }
198
199         @Test
200         public void whenWatchedEpisodeFoundForCurrentUserThenReturnIt() {
201             //given
202             User user = new User();
203             user.setId(DEFAULT_ID);
204
```

```
205             WatchedEpisode watchedEpisode = createWatchedEpisode();
206             watchedEpisode.setUser(user);
207             watchedEpisode.setId(DEFAULT_ID);
208 
209             when(userService.findCurrentUser()).thenReturn(Optional.of(user));
210 
                when(watchedEpisodeRepository.findByEpisodeIdAndUserIsCurrentUser(watchedEpiso
                de.getId())).thenReturn(Optional.of(watchedEpisode));
211 
212             //when
213             Optional<WatchedEpisode> result =
                watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(DEFAULT_ID);
214 
215             //then
216             verify(watchedEpisodeRepository,
                times(1)).findByEpisodeIdAndUserIsCurrentUser(watchedEpisode.getId());
217             assertThat(result.isPresent(), is(true));
218             assertThat(result.get(), is(watchedEpisode));
219         }
220 
221         @Test
222         public void
                whenWatchedEpisodeNotFoundForCurrentUserAndEpisodeNotFoundThenReturnEmptyOptional(
                ) {
223             //given
224             User user = new User();
225             user.setId(DEFAULT_ID);
226 
227             WatchedEpisode watchedEpisode = createWatchedEpisode();
228             watchedEpisode.setUser(user);
229             watchedEpisode.setId(DEFAULT_ID);
230 
231             when(userService.findCurrentUser()).thenReturn(Optional.of(user));
232 
                when(watchedEpisodeRepository.findByEpisodeIdAndUserIsCurrentUser(any())).then
                Return(Optional.empty());
233             when(episodeService.findById(any())).thenReturn(Optional.empty());
234 
235             //when
236             Optional<WatchedEpisode> result =
                watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(DEFAULT_ID);
237 
238             //then
239             verify(episodeService, times(1)).findById(DEFAULT_ID);
240             assertThat(result.isPresent(), is(false));
241         }
242 
243         @Test
244         public void
                whenWatchedEpisodeNotFoundForCurrentUserAndEpisodeWasFoundThenReturnNewWatchedEpis
                ode() {
245             //given
246             User user = new User();
247             user.setId(DEFAULT_ID);
248 
249             Episode episode = createWatchedEpisode().getEpisode();
250 
251             when(userService.findCurrentUser()).thenReturn(Optional.of(user));
252 
                when(watchedEpisodeRepository.findByEpisodeIdAndUserIsCurrentUser(any())).then
                Return(Optional.empty());
253             when(episodeService.findById(any())).thenReturn(Optional.of(episode));
254 
255             //when
256             Optional<WatchedEpisode> result =
                watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(DEFAULT_ID);
257 
258             //then
259             verify(episodeService, times(1)).findById(DEFAULT_ID);
260             assertThat(result.isPresent(), is(true));
261             WatchedEpisode watchedEpisodeResult = result.get();
262 
263             assertThat(watchedEpisodeResult.getEpisode(), is(episode));
```

```
264             assertThat(watchedEpisodeResult.getUser(), is(user));
265             assertNull(watchedEpisodeResult.getId());
266             assertNull(watchedEpisodeResult.getComment());
267             assertNull(watchedEpisodeResult.getRate());
268         }
269 
270         @Test
271         public void whenPassedNullParameterThenReturnEmptyOptional() {
272             //given
273 
274             //when
275             Optional<WatchedEpisode> result =
                watchedEpisodeService.getWatchedEpisodeForCurrentUserByEpisodeId(null);
276 
277             //then
278             verify(episodeService, times(0)).findById(any());
279             assertThat(result.isPresent(), is(false));
280         }
281 
282         // ******************************************
283 
284         private WatchedEpisode createWatchedEpisode() {
285             Series series = new Series();
286             series.setId(DEFAULT_ID);
287             series.setName("Sherlock");
288             series.setReleaseYear(2014);
289             series.setDescription("no shit Sherlock");
290             Season season = new Season();
291             series.addSeason(season);
292             season.setSeries(series);
293             season.setId(DEFAULT_ID);
294             season.setNumber(1);
295             season.setReleaseYear(2014);
296             season.setDescription("descrip");
297             Episode episode = new Episode();
298             season.addEpisode(episode);
299             episode.setSeason(season);
300             episode.setId(DEFAULT_ID);
301             episode.setNumber(1);
302             episode.setDuration(20);
303             episode.setTitle("Odc 1");
304             return new WatchedEpisodeBuilder().episode(episode).build();
305         }
306     }
307 
```

(12)

Przypadki testowe dla PU.

Automatyzacja testów funkcjonalnych.

Badanie jakości projektu