

**Testiranje Spring Boot  
REST servisa na primeru „Movie“**

**aplikacije**

(Projektni zadatak)

SE321 – Obezbeđivanje kvaliteta, testiranje   
i održavanje softvera

Student: **Dušan Potić** Asistent: **Nikola Dimitrijević** Broj indeksa: **3432**

Niš 2020.

Sadržaj

[**1.** **Uvod** 3](#_Toc32444710)

[1.1. Apstrakt 3](#_Toc32444711)

[1.2. Opis aplikacije 3](#_Toc32444712)

[**2.** **Struktura** 4](#_Toc32444713)

[2.1. Opis strukture alikacije 4](#_Toc32444714)

[2.2. Struktura aplikacije 4](#_Toc32444715)

[2.3. Arhitektura aplikacije 5](#_Toc32444716)

[**3.** **Testiranje** 7](#_Toc32444717)

[3.1. Opis i tehnike testiranja 7](#_Toc32444718)

[3.2. Poznata unutrašnja struktura 7](#_Toc32444719)

[3.3. Proces testiranja 14](#_Toc32444720)

[3.4. Rezultati testiranja 21](#_Toc32444721)

[**4.** **Reference** 23](#_Toc32444722)

# **Uvod**

U ovom poglavlju biće predstavljen sistem tj. aplikacija koja će služiti za primer testiranja RESTful web servisa.

# Apstrakt

„Movie“ aplikacija predstavlja Maven aplikaciju u Java programskom jeziku koja koristi RESTful servise za komunikaciju sa bazom podataka. Na primeru Movie aplikacije biće predstavljen način testiranja RESTful servisa u Spring Boot okruženju. Prilikom testiranja koristice se JPA anotacije i MockMVC koji dolazi sa Spring okruženjem. Testiraće se kontroleri i servisi, a od tehnika za testiranje koristiće se gray box, tj. kombinacija white i black box testiranja u IntelliJ IDE.

# Opis aplikacije

„Movie“ web aplikacija predstavlja Maven aplikaciju čiji je cilj da obezbedi korisnicima pretraživanje i dodavanje filmova u omiljenu listu. Svaki film opisan je žanru kojem pripada, direktorima i glumcima. Takođe korisnik može biti i administratorskog tipa sto mu daje mogućnosti CRUD (*Create-Read-Update-Delete*) operacija.

# **Struktura**

Ovo poglavlje predstavlja opis strukture aplikacije koja je navedena kao primer i na osnovu koje će biti predstavljen način i proces testiranja.

# Opis strukture alikacije

Aplikacija je bazirana na REST (*Representational State Transfer*) arhitekturi. Korišćenje Spring Boot okruženja i MySQL baza podataka, dovelo je do obezbeđivanja ORM (*Object relation mapping*) mapiranja klasa entiteta u tabele baze podataka pa je i korišćen JPA (*Java Persistence API*), dok je za konekciju sa bazom podataka korišćen JDBC driver. Aplikacija je izrađena u IntelliJ Idea IDE.

# Struktura aplikacije

Na sledećoj slici (slika br. 1) predstavljena je struktura *„Movie“* aplikacije. Aplikacija se sastoji od foldera *„.idea“* koji je kreiran od strane razvojnog okruženja, zatim *„.mvn“* folder jer je reč o Maven Web aplikaciji i „src“ foldera u kome su smestene klase određenih tipova entiteta, kontrolera, servisa i izuzetaka, ali i konfiguracionih, bezbednosnih i klasa skladišta.

Na slici br. 2 prikazana je struktura test klasa koje su takođe deo projekta i nalaze se u folderu *„test“* u okviru *„src“* direktorijuma aplikacije. Test klase su podeljene u dve celine na testove kontrolnih klasa i na testove klasa servisa. Takođe moguće je primetiti klasu i *„AbstractTest“* koja ne pripada ovim celinama, ali o njoj će biti više reči u daljem tekstu.

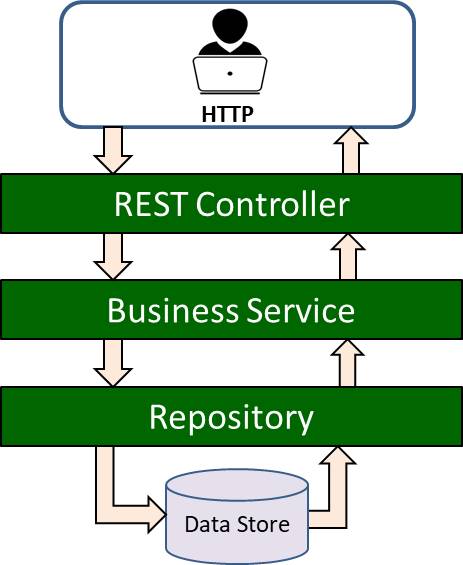
# Arhitektura aplikacije

Slika 1. – Osnovna struktura aplikacije

Slika 2. – Struktura testova unutar aplikacije

U prethodnom poglavlju bilo je reči o REST arhitekturi „Movie“ aplikacije koju je potrebno detaljnije objasniti. Web servisi su aplikacije smeštene na nekom serveru, koji služe za razmenu informacija i podršku interakcije između dve mašine preko mreže.

REST arhitektura predstavlja arhitekturu na kojoj su zasnovani RESTful Web servisi (slika br. 3). Ovi servisi su jednostavnije integrisani sa HTTP-om od SOAP servisa, ne zahtevaju XML poruke ili WSDL opise servisa. Danas se RESTful izdvojio kao dominantan mrežni servis. Podaci se najčešće prebacuju u JSON formatu, ali je takođe dostuman i XML. RESTful servise karakterišu nepostojanje stanja (Stateless), mogućnost keširanja (Cacheable) i izričito korišćenje HTTP metoda dok se transfer obavlja u XML i/ili JSON formatu. REST koristi „CRUD like“ HTTP metode kao što su: GET, POST, PUT ili PATCH i DELETE.



Slika 3. – Primer REST arhitekture, Spring Boot

# **Testiranje**

U ovom poglavlju biće predstavljen opis testiranja i sam proces testiranja RESTful servisa, takođe biće prikazani alati koji su korišćeni za testiranje.

# Opis i tehnike testiranja

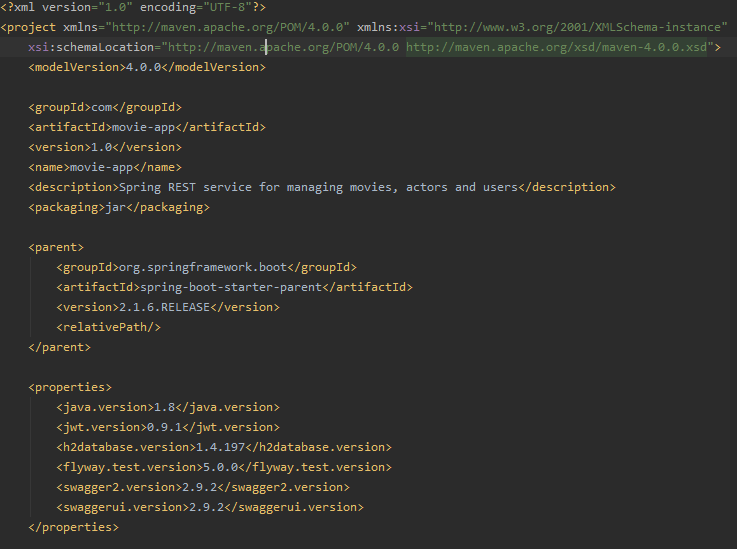
Testiranje je izvršeno po „Unit“ nivou testiranja korišćenjem metoda sive kutije (Gray Box). U ovoj metodologiji unutrašnja struktura sistema je delimično poznata testeru, a tester poznaje strukturu podataka i algoritme koji su korišćeni za razvijanje softvera, što predstavlja osobine „White Box“ testiranja. Primer ovog metoda je analiza koda aplikacije, formiranje funkcionalnih test scenarija na snovu te analize i kasnijim testiranjem primenom „Black Box“ metoda testiranja. Ograničenje GrayBox metode je što pojedini delovi interne strukture moraju biti poznati.

# Poznata unutrašnja struktura

**Konfiguracija**

Unutrašnju strukturu čine svi fajlovi unutar aplikacije, ali kada se prvenstveno govori o testiranju potrebno je napomenuti one najbinije fajlove tj. tipove klasa aplikacije. Kod testiranja RESTful Web servisa u Spring Boot okruženju prvenstveno se misli na klase entiteta, modela, kontrolera i servisa ali ne sme se zaboraviti i pom.xml fajl. Pom fajl sa ekstenzijom „xml“ predstavlja glavni konfiguracioni fajl gde su smetene sve zavisnosti i podešavanja koja se koriste u aplikaciji.

„Parent“ tag u „xml“ fajlu pod imenom „Pom“ predstavlja tag u okviru kog se navode verzije zavisnosti (slika br. 4) koje će se umetnuti kroz tag „Dependencies“, dok tag „Properties“ predstavlja tag koji sa sobom nosi verzije alata koji se koriste i verziju Jave.

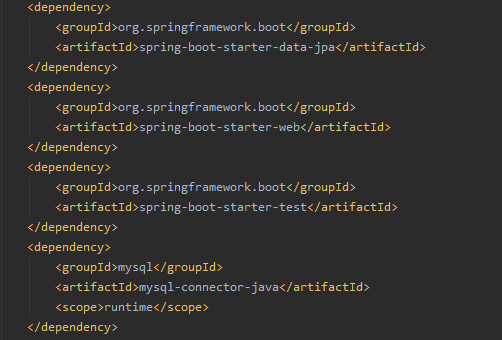


Slika 4. – Parent i Properties tag

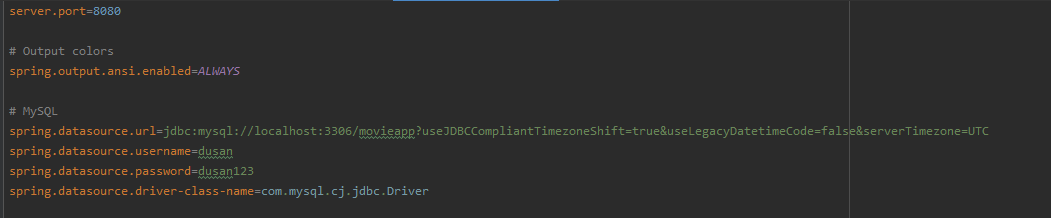
„Dependencies“ tag sadrži sve one zavisnosti neophodne za rad Spring Boot aplikacije i one koje „.jar“ fajlove koje dodajemo naknadno, kao na primer JUnit alat za testiranje. S obzirom da Spring Boot sa svojim osnovnim zavisnostima uključuje i JUnit 4 nije potrebno umetati njenu posebnu zavisnost.

Da bi se najpre započela izrada neke Maven Spring Boot aplikacije neophodno je navesti one osnovne zavisnosti koje će nam omogućiti testiranje i samo pokretanje aplikacije, a to su: starter-web i starter-test.

Na osnovu uvodnog opisa aplikacije: *„Korišćenje Spring Boot okruženja i MySQL baza podataka, dovelo je do obezbeđivanja ORM (Object relation mapping) mapiranja klasa entiteta u tabele baze podataka pa je i korišćen JPA (Java Persistence API), dok je za konekciju sa bazom podataka korišćen JDBC driver“*, potrebno je uvesti i dodatne zavisnosti za JDBC poznati „mysql-connector-java“ i za JPA „spring-boot-starter-data-jpa“ (slika br. 5).



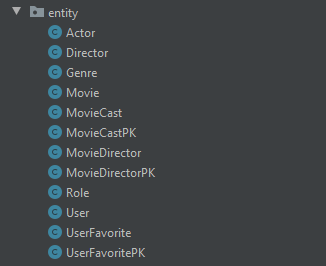
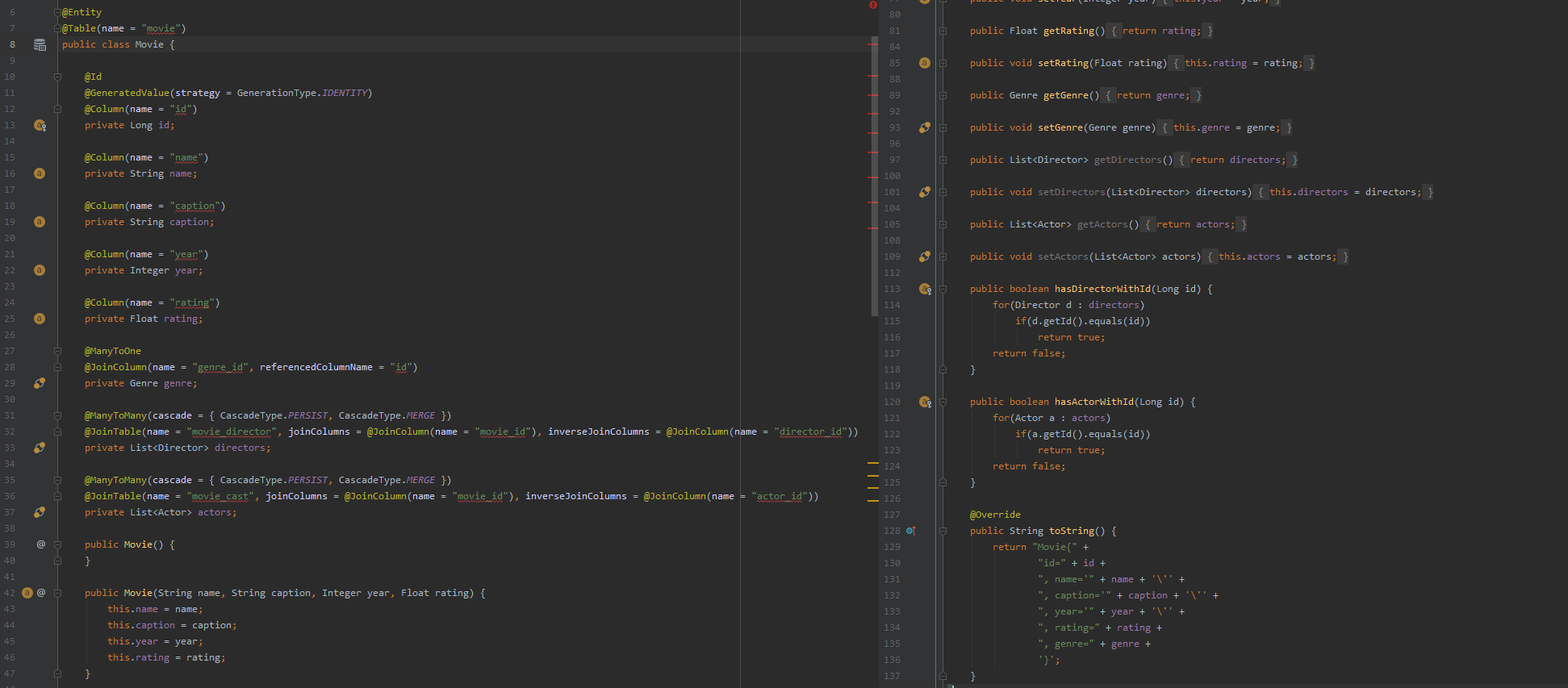
Slika 5 – Umetnute zavisnosti potrebne za testiranje

Nakon uvođenja zavisnosti potrebno je i povezati JDBC sa bazom podataka. Sadržaj za ubacivanje u bazu podataka nalazi u „data.sql“ fajlu kojeg je potrebno izvršiti na već konektovanu bazu podataka. Da bi povezali lokalnu bazu podataka i JDBC drajver potrebno je prvo pokrenuti XAMPP serverski paket, kada je to urađeno potrebno je i konfigurisati „application.properties“ datoteku (slika br. 6).

Slika 6. – Konfiguracija „aplication.properties“ datoteke. Odabir porta i dodavanje kredencijala

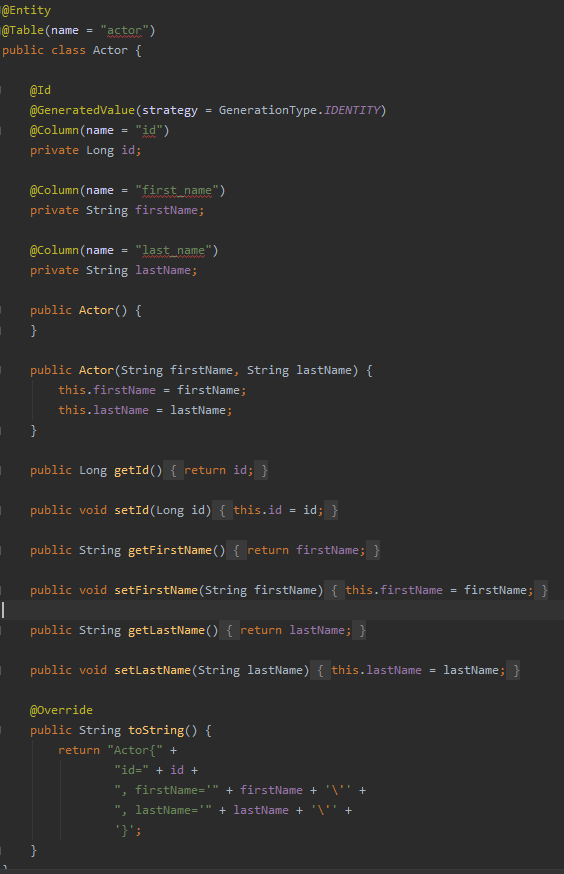
**Klase entita**

Klase entiteta su prikazane na slici br. 7. S obzirom na veliki broj klasa entiteta biće prikazani entiteti Move (slika br. 8) i Actor (slika br. 9).



Slika 8. – Entitet klasa Movie

Slika 7. – Klase entiteta



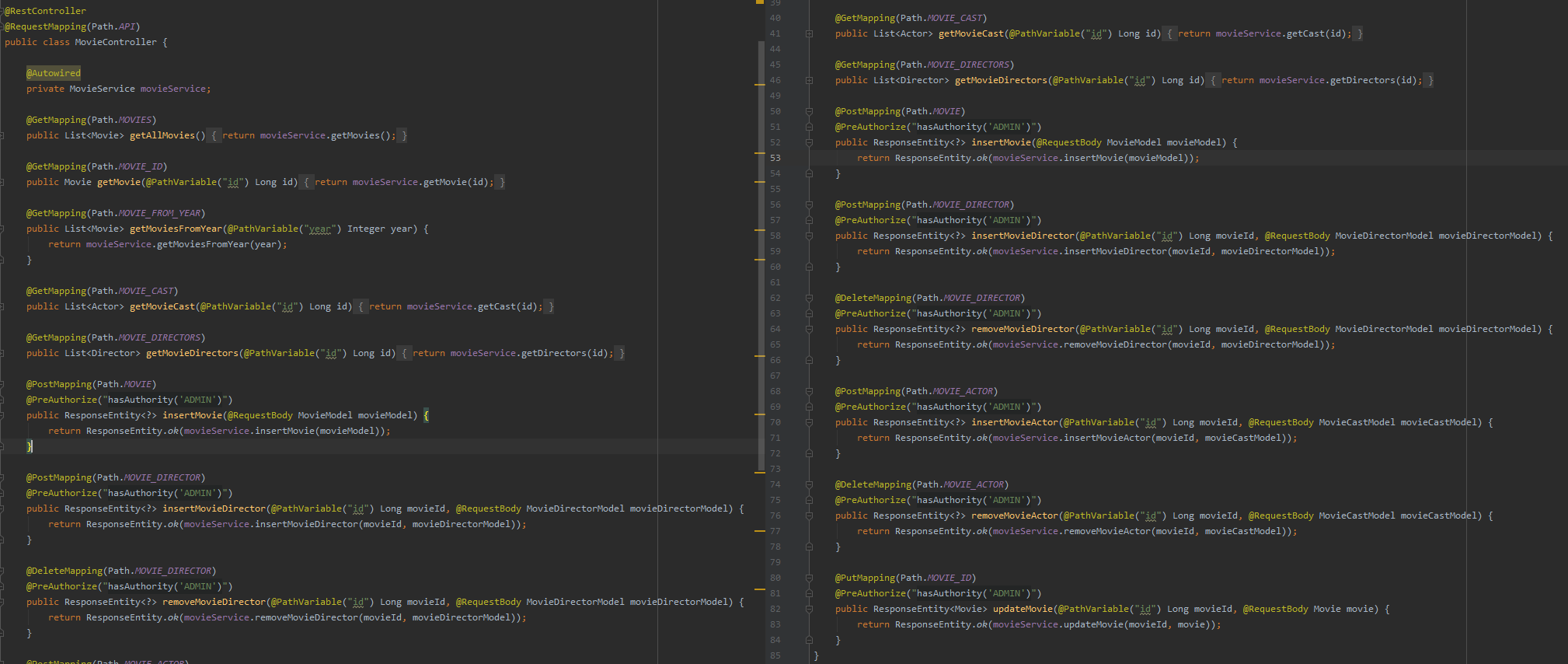
Slika 9. – Entitet klasa Actor

Kao što se iz priloženih slika može zaključiti korišćenjem JPA anotacija obeležene su klase entiteta, tako da će se one preslikati u tabele u bazi podataka. Sami atributi entiteta su mapirani sa odgovarajućim atributima tabele u bazi podataka.

**Klase kontrolera**

****Kontrolne klase obeležene su anotacijama @RestController i @RequestMapping. Svaka klasa kontrolera ima instancu servisa sa kojima komunicira. Svaka metoda je obležena anotacijama @GetMapping I iz Path datoteke izvlace url koji gađaju. U nastavku teksta priloženi su prikazi Actor I Movie kontroler klasa.

Slika 10 – Actor Controller

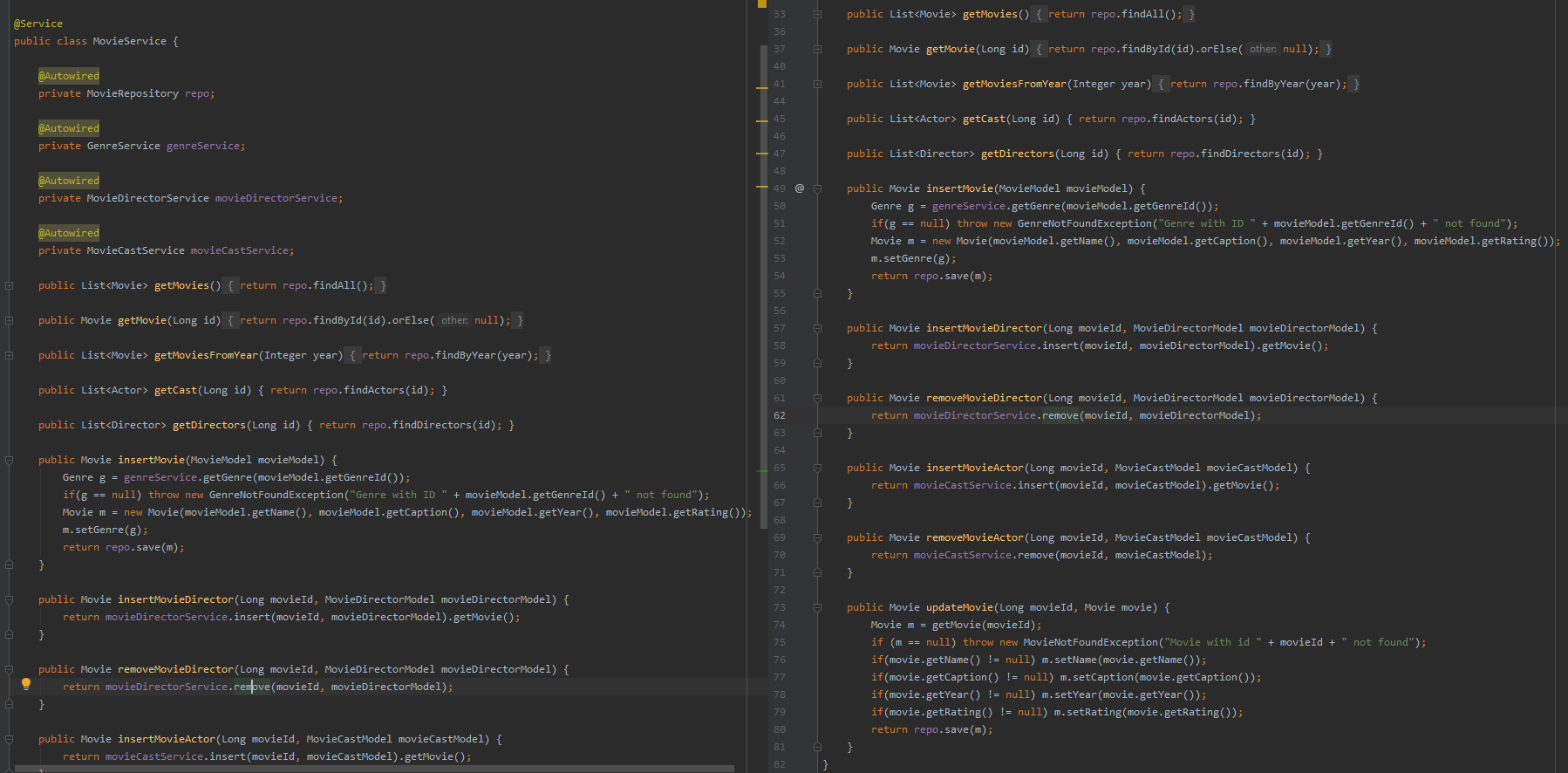


Slika 11 – Movie Controller

**Klase servisa**

Klase servisa su obeležene anotacijama @Service. U nastavku su klase servisa ActorService, MovieService i MovieCastService.

Slika 12 - ActorService



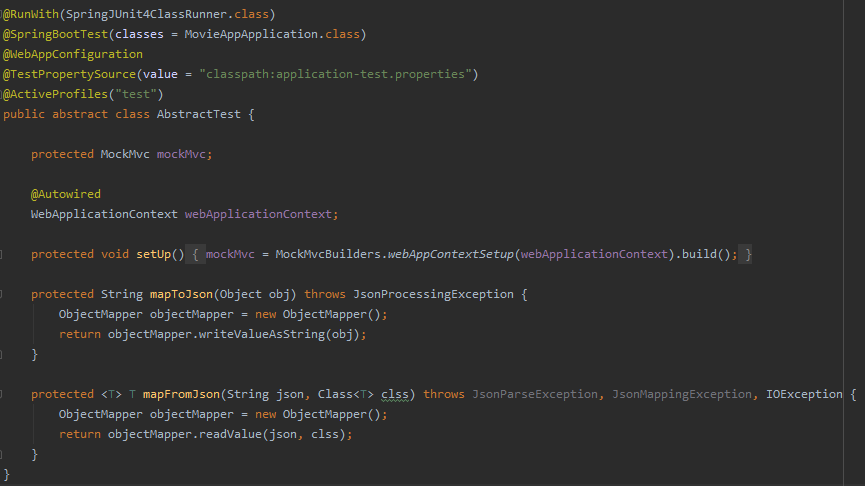
Slika 13 – Movie Service

# Proces testiranja

Kako bi proces testiranja bio izvršen na bezbedan način potrebno je obezbediti „lazne objekte“. Lažni objekti neće ostaviti trag u bazi podataka, a pružiće mogućnost testiranja tako sto će biti simulatori stvarne obrade podataka.

U nastavku je prikazana naša AbstractTest klasa koja predstavlja roditeljsku klasu svim test klasama. Njen zadatak je da za svaku test klasu stvori Mock tj. lažni objekat i tako obezbedi našu bazu podataka sigurnom od izmene podataka tokom testiranja.

AbstractTest je apstraktna klasa i sadrzi u sebi MockMVC objekat koji koristi *MockMvcBuilders.webAppContextSetup()* kako bi obezbedio kreiranje tog lažnog objekta.

Kada je kreirana AbstractTest apstraktna klasa možemo početi kreiranje test klasa.

Slika 14 - AbstractTest

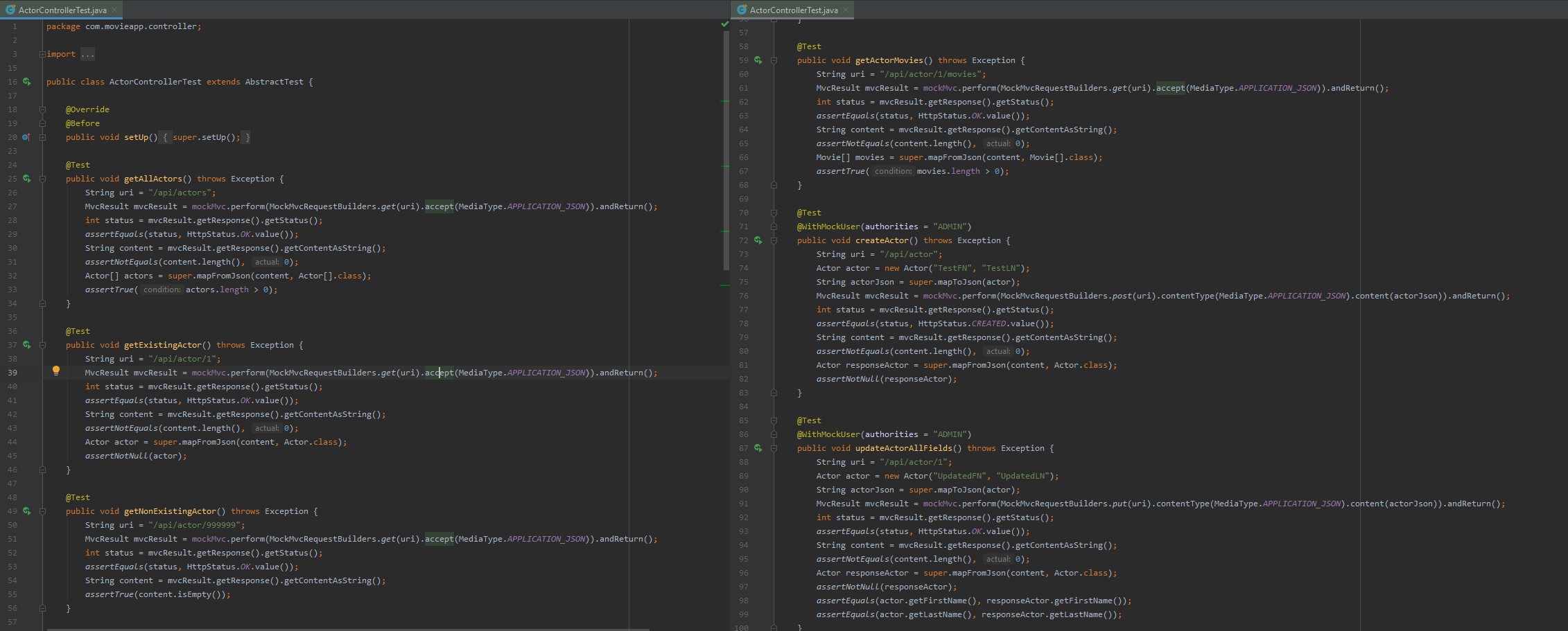
S obzirom da su predstavljene Actor i Movie klase kao i MovieCast model i njihovi kontroleri, ostaćemo dosledni tome pa će biti predstavljene test klase njihovih kontrolera i servisa.

**Testiranje kontrolera i servisa**

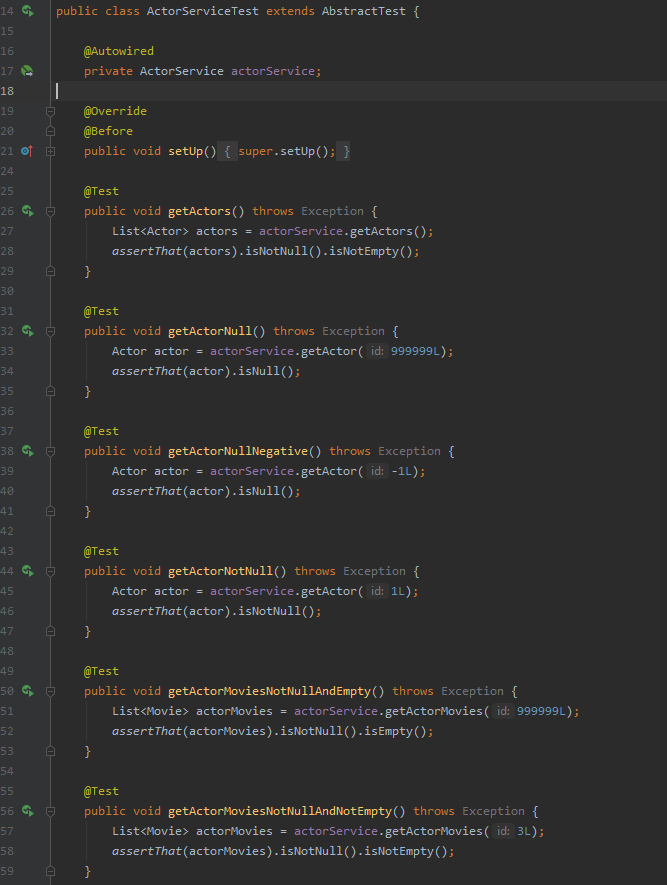
Testiranje kontrolera i servisa mora pokriti sav kod i sve moguće scenarije do kojih može doći.

Potrebno je ispitati kako se ponašaju kontroleri usled zadavanja loših ili nepostojećih parametara. Kod testiranja servisa mora se takođe posebno obratiti pažnja i pokriti svaki slučaj pokušaja upisivanja nekog nevalidnog ili već nepostojećeg objekta ukoliko on dolazi u situaciju među-tabela nekih objekata.

Uzećemo za primer našu MovieCast tabelu koja predstavlja među tabelu između Actor i Movie tabele. Na osnovu MovieCast modela potrebno je ispitivati da li uopšte postoje Actor i Movie objekat sa datim „ID“ ključem i da li su „null“ ili ne, pa tek onda nakon provere upisati nov model MovieCast u bazu.

U nastavku biće predstavljene test klase kontrolera ActorControllerTest, ActorServiceTest, MovieControllerTest, MovieServiceTest i MovieCastServiceTest.

Slika 15 – ActorControllerTest

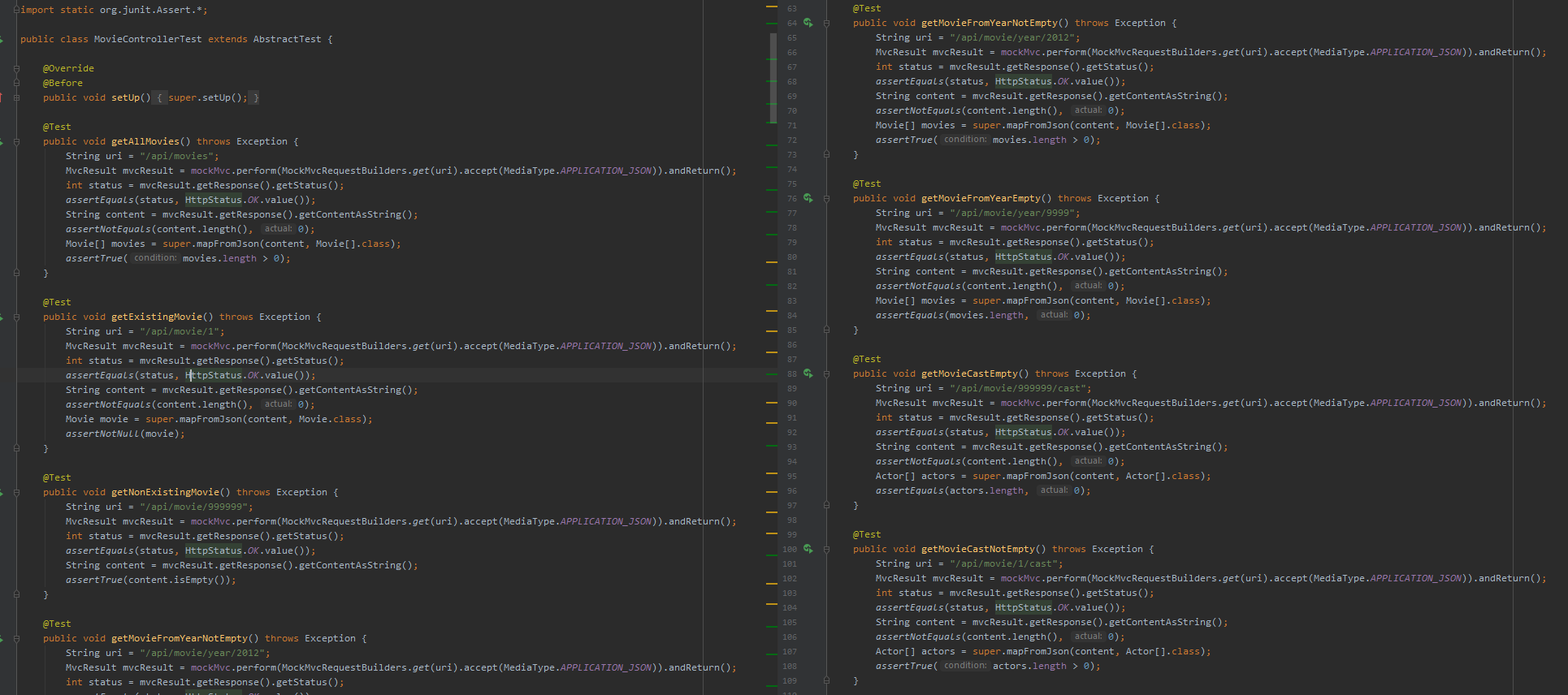


Slika 16 - ActorServiceTest

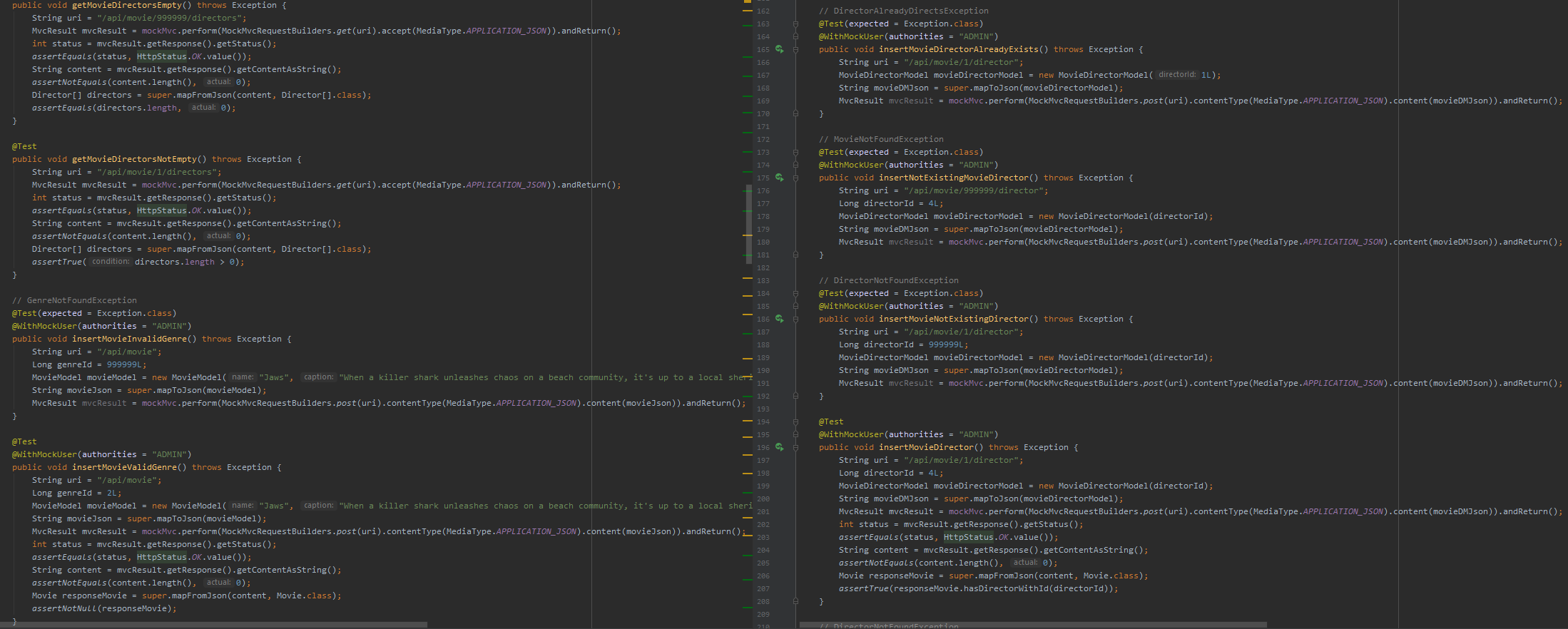
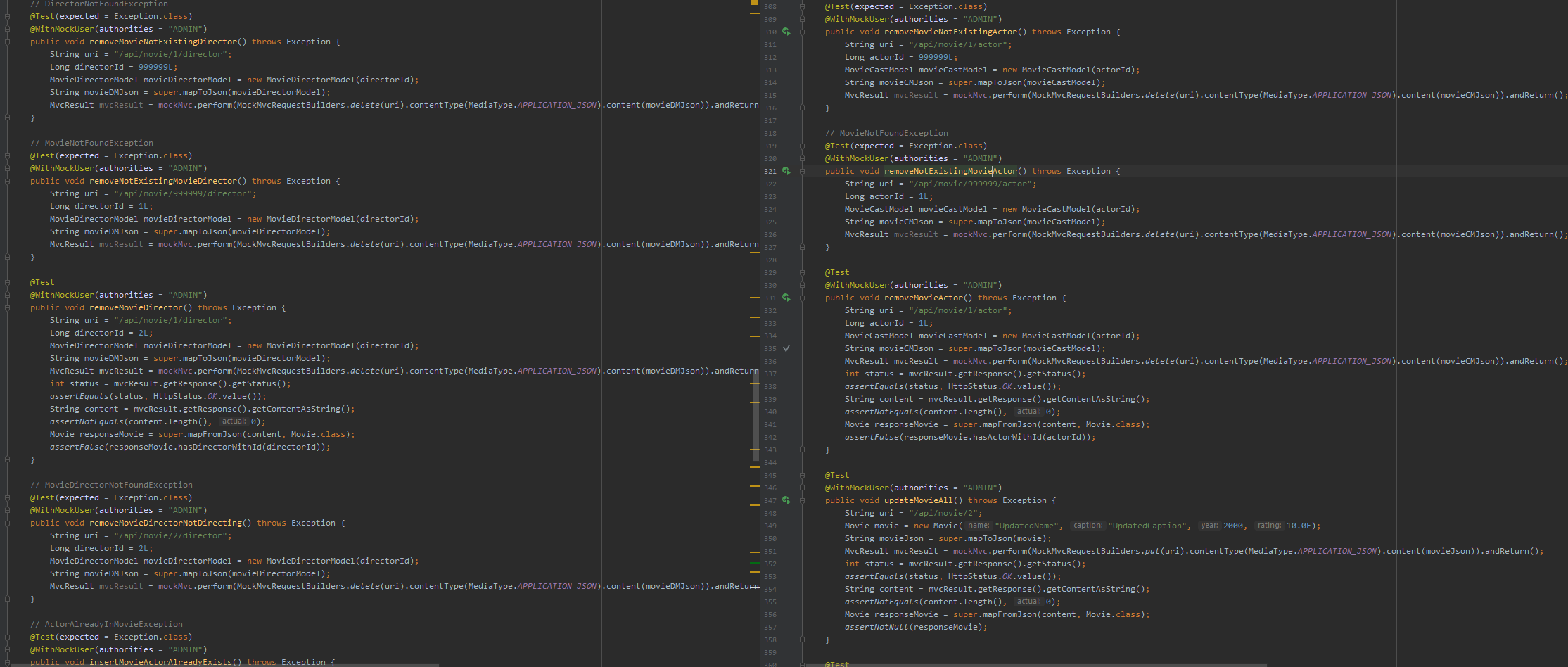
Kao što ste mogli primetiti testirali smo sve moguće slučajeve poklušaja upisivanja u bazu, počevši od provere da li su liste objekata koje nam servis vraća prazne i postojeće do testiranja da li postoje objekti u bazi koji se unose u među-tabele ili su već sadržani u njoj.

Ovakam način testiranja nam omogućava da pokrijemo sve moguće slučajeve koji bi mogli oštetiti podatke u bazi podataka i dovesti je do nekonzistentnog stanja i gubljenja podataka.

U nastavku biće prikazani test klase nad klasama MovieController, MovieService i MovieCastService.

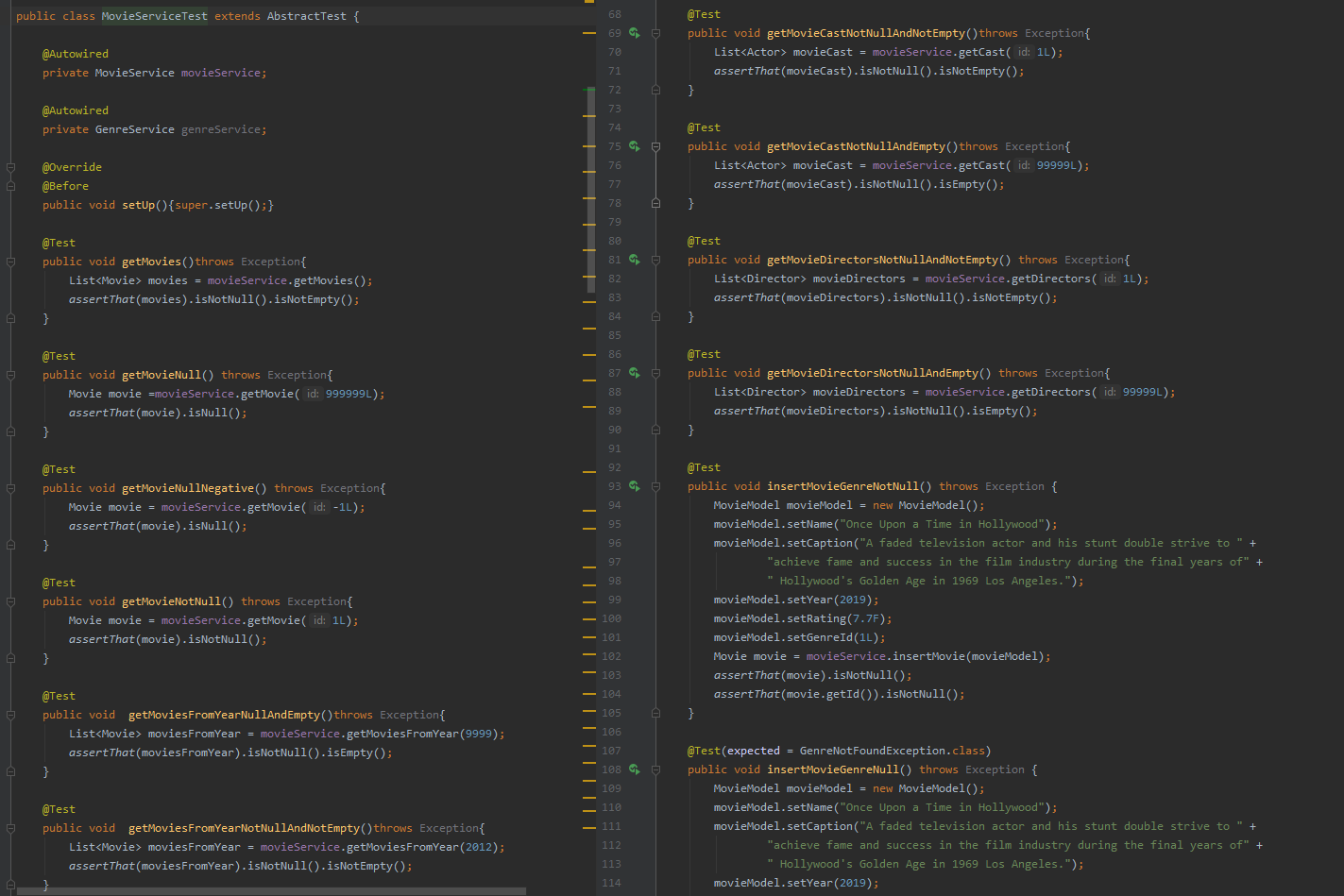
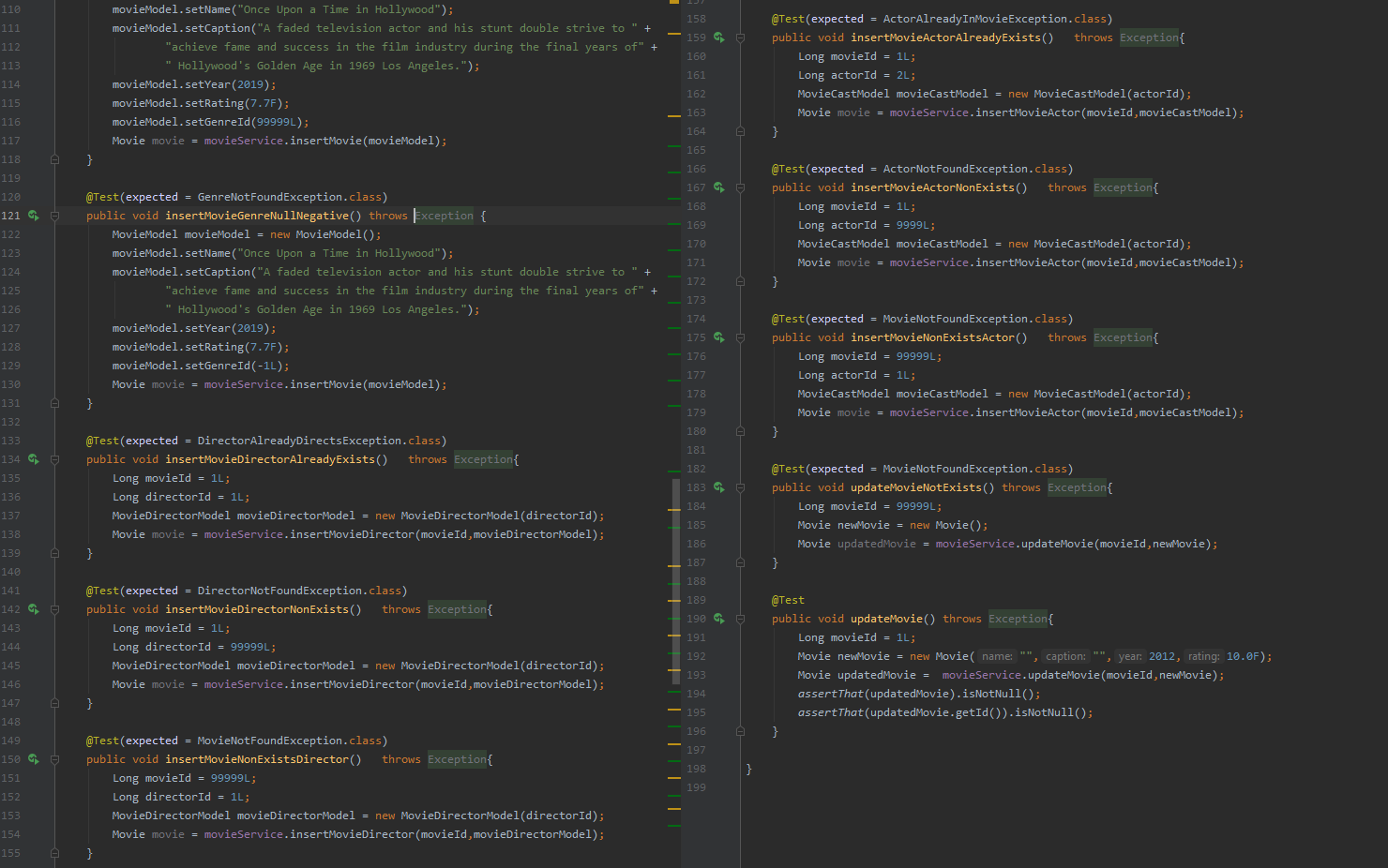


Slika 17 – MovieControllerTest 1.deo koda



Slika 19 – MovieControllerTest 3.deo koda

Slika 18 – MovieControllerTest 2.deo koda

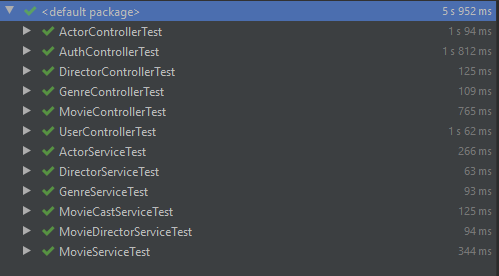


Slika 21 – MovieServiceTest 2.deo koda

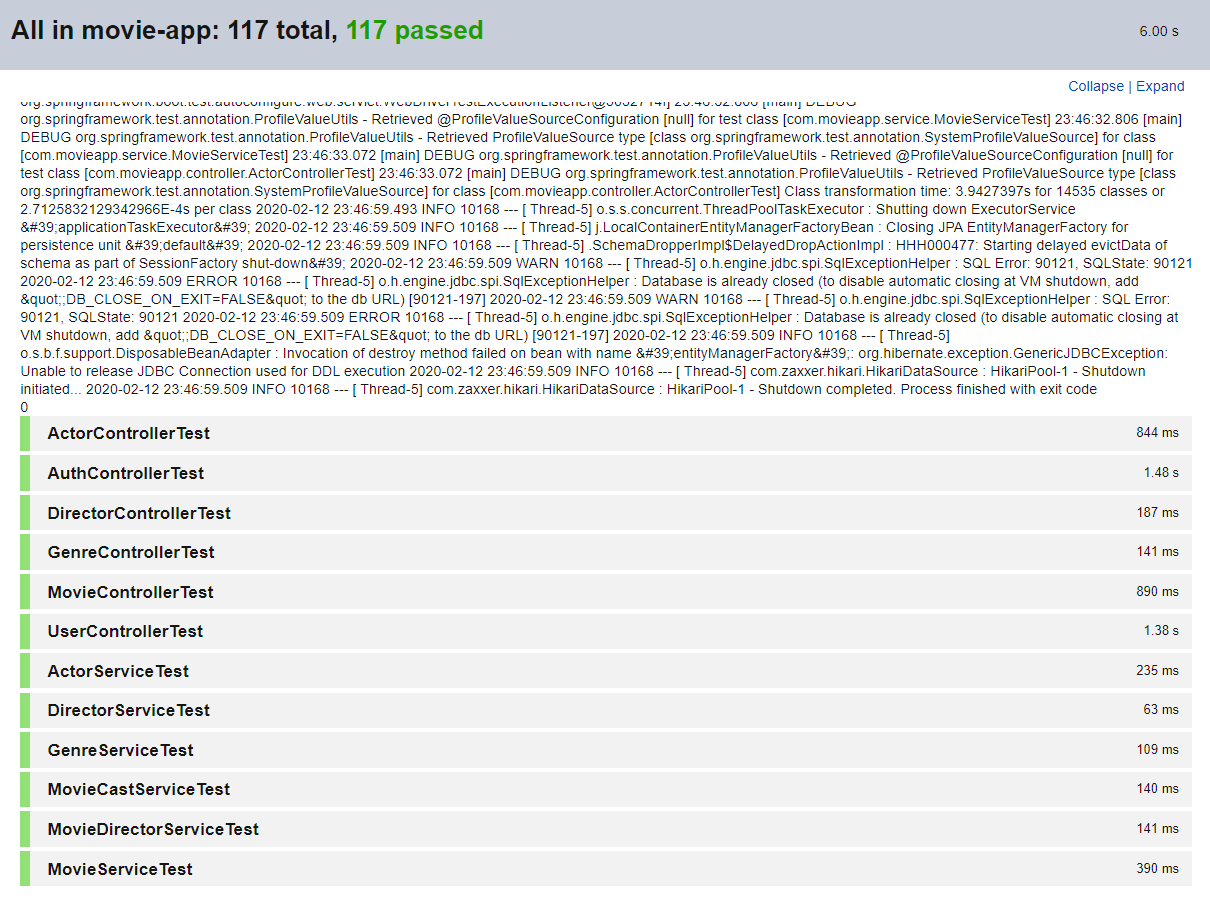
Slika 20 – MovieServiceTest 1.deo koda

# Rezultati testiranja

Slika 21– MovieCastServiceTest

Testiranje je prošlo u najboljem redu. Svi testovi su bili uspešni. Testiranje je rađeno preko IntelliJ Test Coverage – a tako da smo na kraju dobili HTML stranicu sa izveštajem testiranja.

Slika 22 – Rezultati testiranja



Slika 23 – Rezultati testiranja IntelliJ Test Coverage

# **Reference**

SE321 lekcije sa Lams-a - [Lekcije sa Lamsa](http://lams.metropolitan.ac.rs/)

Spring dokumentacija – [Spring testiranje](https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html?fbclid=IwAR1WS5Ab2dlFl7tNHBpZpMU9zBNcGRKa4yA7QYH9Y62jh9mJeMxG8h5oLSw)

Youtube tutorijali o Spring – [Spring Quick Start](https://www.youtube.com/watch?v=msXL2oDexqw&list=PLqq-6Pq4lTTbx8p2oCgcAQGQyqN8XeA1x)

Youtube tutorijali o Spring testiranju – [Spring Boot Unit Testing](https://www.youtube.com/watch?v=RbZvXCAtMus)