

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ORGANIZACIJE I INFORMATIKE**  
**V A R A Ž D I N**

**Zlatko Mehanović**

**Web aplikacija za pregled i pretraživanje  
filmova pomoću TMDb-a**

**ZAVRŠNI RAD**

**Varaždin, 2017.**

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ORGANIZACIJE I INFORMATIKE**  
**V A R A Ž D I N**

**Zlatko Mehanović**

**Matični broj: 41981/13–R**

**Studij: Informacijski sustavi**

**Web aplikacija za pregled i pretraživanje  
filmova pomoću TMDb-a**

**ZAVRŠNI RAD**

**Mentor:**

Prof. dr. sc. Danijel Radošević

**Varaždin, rujan 2017.**

# Sadržaj

1. Uvod .....	1
2. TMDB.....	3
2.1. Metoda search.....	5
2.2. Metoda discover .....	7
2.3. Metoda find .....	9
2.4. Metoda movie.....	10
2.5. Spajanje zahtjeva .....	14
2.6. Metoda genre .....	15
2.7. Dohvaćanje slika .....	16
3. Alati i tehnologije .....	18
3.1. Eclipse .....	18
3.2. Java.....	22
3.3. Vaadin.....	23
3.4. SCSS.....	26
3.5. Alati i tehnologije za izradu baze podataka.....	27
4. Aplikacija.....	29
4.1. Dohvaćanje JSON-a .....	29
4.2. Početna stranica .....	32
4.3. Pretraživanje filmova.....	35
4.4. Obilježavanje stranica .....	38
4.5. Prikaz detalja .....	41
4.6. Pretraživanje po žanrovima .....	48
4.7. Navigacija i povijest .....	51
5. Zaključak .....	54
6. Literatura .....	55

# 1. Uvod

U ovom radu prikazati ću te opisati kako se u današnje vrijeme može jednostavno izraditi web aplikacija koja može pretraživati filmove, serije, glumce i slično. Kroz ovaj rad ću također ukratko prikazati i opisati razne tehnologije koje sam koristio u svrhu izrade primjera aplikacije, od čega je naravno glavni TMDb API (engl. The movie database application programming interface) kojega sam koristio za dohvaćanje podataka i informacija o filmovima.

Sa obzirom da će se u nastavku relativno često pojavljivati pojam API smatram da bi bilo korisno prije nego li započnem sa bilo kakvim primjerima, pojasniti što je to zapravo te obrazložiti samu svrhu i naravno zašto sam se točno odlučio koristiti ovaj API prilikom izrade aplikacije.

API, kao što se može razaznati iz samoga naziva je sučelje koje služi za programiranje aplikacija, najlakše je povući usporedbu sa korisničkim sučeljem (eng. User interface) koje je vidljivo krajnjem korisniku te je izrađeno upravo sa ciljem da bi se korisnik mogao služiti određenom aplikacijom. Za razliku od korisničkog sučelja API je izrađen kao sučelje za programere iliti kao što se ponekada spominje kao „sučelje čitljivo računalu“ naspram korisničkog sučelja koje je „čitljivo ljudima“ (engl. Machine readable vs human readable interfaces) (Berlind David, 2015). Sa obzirom da su API-i izrađeni upravo za programere njihova svrha je uglavnom kako bi se olakšala i ubrzala izrada aplikacija programerima, to je u biti kod koji već postoji te ga se nije potrebno ponovno izrađivati i osmišljavati već samo koristiti, unaprijediti te izraditi nešto što će biti od koristi i običnim korisnicima. Na tržištu već postoji gomila API-a tako da ukoliko je nekome nešto potrebno može lako pronaći nešto što će mu olakšati posao pri izradi aplikacije, neki od korisnih popularnijih API-a su google maps koji omogućuje dodavanje google karte u web stranicu, youtube koji omogućuje između ostaloga pretraživanje videa te mnogi drugi.

Što se tiče potreba za izradu web aplikacije za pretraživanje filmova tv serija te ostalih relevantnih informacija pronašao sam nekoliko dostupnih API-a, glavna tri su vezana uz OMDb (online movie database), TMDb (the movie database) te IMDb (Internet movie database). Iako je IMDb kao sama web stranica najpopularnija kod korisnika te se većina ljudi kad-tad našla na toj stranici prilikom pretraživanja informacija o nekom filmu ili seriji putem google tražilice, nažalost nemaju vlastiti službeni API tako da ukoliko se želi koristiti njihov sadržaj potrebno je koristiti neslužbene izvore koji ne pružaju tu sigurnost, stabilnost i korisničku podršku kao što bi to činili službeni, iz tog razloga sam se odlučio ne koristiti IMDb iako bi se možda na prvi pogled doima kao najbolja opcija. Preostala dva sam malo detaljnije pregledao, proučio neke

funkcionalnosti i usporedio, ukratko OMDb sadrži bolje informacije vezane uz kratki sadržaj filmova te informacije o redateljima i glumcima, također jedna mala prednost je ta što za korištenje OMDb-a nije potrebno posjedovati nikakav ključ međutim dokumentacija te same opcije nisu pretjerano jasno definirane tako da se u početku doima nejasno te složenije nego što zapravo je. Za razliku TMDb nudi poprilično bolje dokumentiran i jasan API te metode koje se koriste su veoma jednostavne za savladati te ukoliko se nađe na nekakav problem često se rješenje može pronaći putem njihovih foruma gdje su korisnici poprilično aktivni. Iako sam ranije napomenuo da OMDb nudi bolje informacije o filmovima to ne znači da su same informacije sa TMDb-a beskorisne ili neispravne, još jedna sitna prednost je ta što TMDb sadrži više te kvalitetniji izbor slika od filmova, za razliku od OMDb-a tu je potrebna registracija te svaki korisnik mora posjedovati API ključ kako bi se mogao njime služiti, da bi se dobio vlastiti ključ potrebna je samo registracije te slanje zahtijeva za njime.

Berlind David (2015) smatra da bi u ovom trenutku tokom izrade nekakve aplikacije bilo poželjno koristiti oba dva prethodno navedena API-a, time bi se mogle iskoristiti prednosti sa obje strane te bi se međusobno mogli nadopunjavati. No za potrebe izrade manje aplikacije poput ove smatram da je bolji izbor ipak TMDb uglavnom zato što ima poprilično bolju dokumentaciju te primjere iz kojih se lagano može shvatiti i naučiti kako njihov API funkcionira te kako ga je najbolje koristiti. U nastavku rada prikazati ću detaljnije kako se može TMDb API koristiti, koje sam tehnologije koristio prilikom izrade aplikacije te naravno kako sama aplikacija izgleda i funkcionira.

## 2. TMDB

Kao što se iz samoga naziva može vidjeti TMDB je u biti javna baza podataka za filmove te glavni njegov aspekt po čemu se ističe i razlikuje od svojih konkurenata je taj što je TMDB potpuno izrađen od strane korisnika odnosno zajednice (engl. community), još od svojeg početka 2008. godine sav sadržaj koji se nalazi u bazi je dodan od strane zajednice. Iako je TMDB trenutno dostupan u 39 različitih jezika on se svakoga dana koristi u preko 180 različitih zemalja, u prosjeku se svakoga dana dodaje preko tisuću novih slika bile one službene ili izrađene od fanova te se svakoga dana zaprima preko dvije milijarde upita odnosno zahtjeva te se njime služe preko 125 tisuća programera i kompanija (TMDb, 2017)). Smatram da i same statistike dokazuju dovoljno o samoj kvaliteti njihovog servisa te pokazuju kroz godine u kojima postoje da se veoma pouzdani i da im se može vjerovati.

Kako bi se započeo koristiti TMDB API prvo je potrebna registracija kako bi dobili vlastiti ključ putem kojega se šalju zahtjevi prema njihovoj bazi, kako bi se dobio ključ potrebno je samo otići na postavke od vlastitog profila na web stranici, kategoriju pod nazivom API te ispuniti ponuđeni obrazac. Svrha ključa je uglavnom kako bi se spriječila zloupotreba servisa te radi njihove sigurnosti, slanje upita je limitirano na 40 zahtjeva te taj limit nije vezan uz sam ključ već je povezan na IP adresu sa koje određeni zahtjev dolazi, stanje limita je vidljivo putem zaglavlja koje se može jednostavno pronaći i putem razvojnih alata unutar odabranog pretraživača. Na slici 1. je vidljiv jedan jednostavan upit u kojemu se vidi i primjer ključa koji je potreban za izvršavanje gotovo svih zahtjeva koji se šalju te na samome dnu odgovora je vidljiv i limit zahtjeva te koliko ih je još dostupno do slijedećeg resetiranja pod nazivom X-RateLimit.



Slika 1. Zaglavlje upita

Prije početka opisivanja i prikazivanja rezultata koji se dohvaćaju pomoću API-a bitno je napomenuti da se svi rezultati dohvaćaju u JSON formatu, tako da ukoliko se često koristi i pregledava JSON format kao u našem slučaju preporučio bih korištenje bilo kakvih proširenja ili dodatak po želji kako bi se JSON format mogao predočiti mnogo jasnije i logičnije. Za vlastite potrebe odlučio sam se koristiti JSON Viewer proširenja za Google Chrome preglednik te će slike primljenih rezultat biti prikazane uz pomoć navedenog proširenja za razliku od prikaza u jednoj liniji koja je poprilično nečitljiva i nepregledna.

U nastavku poglavlja prikazati ću detaljnije kako koristiti API te neke njegove metode i savjete kako se neke stvari mogu jednostavno dohvatiti ili filtrirati. Iako postoji dosta metoda koje se mogu koristiti sve imaju jednaki početni URL „<https://api.themoviedb.org/3>“ te svaki poslani zahtjev mora sadržavati validan API ključ kako bi se zahtjev mogao poslati npr. „[api\\_key=7369e06a600e15f985316d3189192793](https://api.themoviedb.org/3/movie/40016?api_key=7369e06a600e15f985316d3189192793)“, ukoliko ključ nije ispravan bit će nam vraćena poruka koja nas obavještava da ključ nije validan te da nam je potreban validan ključ da bismo se mogli služiti servisom kao što je prikazano na slici 2.

```
1 // 20170823192855
2 // https://api.themoviedb.org/3/movie/40016?api_key=xxxx
3
4 {
5   "status_code": 7,
6   "status_message": "Invalid API key: You must be granted a valid key."
7 }
```

Slika 2. Neispravan ključ

## 2.1. Metoda search

Metoda search je jedna od bitnijih funkcionalnosti pomoću koje se dohvaćaju rezultati pretrage u kojima se nalaze razni detalji ovisno o upitu koji smo poslali za pretraživanje, kao što sam prethodno prikazao svaki zahtjev ima jednaki početni URL na koji se nadovezuju metode. Kako bismo koristili metodu za pretraživanje potrebno je dodati nastavak search u početni URL „<https://api.themoviedb.org/3/search>“. Sa obzirom nam ova metoda omogućuju pretraživanje različitih vrsta podataka te bi pretraga mogla biti pretjerano opširna i vratiti mnogo više rezultata nego što naše potrebe to zahtijevaju, potrebno je dodatno specificirati što točno pretražujemo. Pretraživanja možemo vršiti prema kompanijama koje su izradile određenu tv seriju iliti film dodavanjem nastavka „*/company*“, također možemo pretraživati kolekcije odnosno serijale dodavanjem nastavka „*/collection*“, ključne riječi pomoću „*/keyword*“, osobe poput glumaca, redatelja, scenarista i sličnih pomoću „*/person*“, tv serije pomoću „*/tv*“, filmove sa „*/movie*“, te na kraju možemo koristiti i sufiks „*/multi*“, koji nam omogućuje pretraživanje nekoliko različitih tipova, trenutno dostupni putem ove metode su pretraživanje filmova, tv serija i osoba.

Svaka od prethodno navedenih opcija također prima nekoliko jednakih parametara od kojih je naravno i varijabla koja sadržava ključ „*api\_key*“ te parametar „*query*“, u koji zapisujemo niz znakova odnosno upit koji želimo pretražiti u bazi, ove dvije vrijednosti moraju biti ispunjene kako bismo mogli koristiti metodu za pretraživanje. Preostale mogućnosti po kojima možemo dodatno filtrirati sadržaj koji se pretražuje uključuju i pretraživanje po jezicima tako da npr. ne moramo tražiti filmove koje ionako ne razumijemo, kako bismo specificirali jezik potrebno je dodati parametar „*language*“ u koji možemo zapisati koji jezik nas interesira, vrijednosti koje upisujemo je potrebno zapisati u ISO 639-1 standardu, npr. za filmove na hrvatskome jeziku potrebno je dopisati „*hr*“ u parametar a ukoliko nekoga interesira neki specifični jezik može jednostavno potražiti na internetu standard u kojemu su zapisane mogućnosti koje se mogu pomoću ove metode pretraživati, početna odnosno zadana vrijednost je „*en-US*“.

Pretraživati također možemo i prema regiji pomoću parametra „*region*“, ili čak i po godini u kojoj je neka tv serija počela izlaziti uz pomoć parametra „*first\_air\_date\_year*“ odnosno samo „*year*“ ukoliko se izvršava pretraga samo po filmovima, ukoliko se radi o pretrazi „*multi*“ onda nam pretraživanje prema godini nije omogućeno. Još jedna veoma bitna opcija je i „*page*“ koja nam je veoma bitna kod izrade straničenja te ću ju detaljnije



objasniti uz primjer aplikacije. Na slici 3. se pri vrhu rezultata nalazi upit koji smo poslali te sam odgovor na taj upit. U prikazanom URL-u možemo vidjeti da je započet sa definiranjem API-a, nakon čega slijedi naziv metode koja se koristi, u ovom slučaju je to „search“ te dodatne opcije „multi“. Slijedeće imamo varijablu u kojoj prosljeđujemo ključ, varijablu u kojoj smo zapisali jezik koji pretražujemo, broj stranice koji želimo dohvatiti ukoliko imamo previše rezultata te ne možemo dohvatiti sve rezultate sa jednim upitom, naravno i sami upit koji želimo pretražiti unutar varijable „query“, te na kraju imamo i varijablu koja filtrira sadržaj za maloljetnike.

U rezultatu koji smo zaprimili imamo zapisano na kojoj se stranici trenutno nalazimo, ukupan broj dostupnih rezultata i stranica, te u polju „results“ imamo i informacije o pojedinim filmovima serijama i slično. Neke od vrijednosti koje smo zaprimili uključuju naziv „title“, tip koji je pronađen „media\_type“ koji je u slučaju sa primjera tipa film, ukoliko bi se radilo o seriji bilo bi zapisano kao „tv“, također imamo i kratak opis pod „overview“, datum izlaska „release\_date“, te identifikacijski broj filma „id“ pomoću kojega možemo pronaći još više detalja o filmu te polje u kojemu se nalaze identifikacijski brojevi za žanrove „genre\_ids“ koje ću detaljnije pojasniti u nastavku uz aplikaciju.

```
// https://api.themoviedb.org/3/search/multi?api_key=7369e06a600e15f985316d3189192793&language=en-US&query=defender&page=1&include_adult=false

{
  "page": 1,
  "total_results": 53,
  "total_pages": 3,
  "results": [
    {
      "vote_average": 10,
      "vote_count": 1,
      "id": 447689,
      "video": false,
      "media_type": "movie",
      "title": "Defender",
      "popularity": 1.841303,
      "poster_path": null,
      "original_language": "en",
      "original_title": "Defender",
      "genre_ids": [44],
      "backdrop_path": null,
      "adult": false,
      "overview": "Jeff Adachi is a public defender in San Francisco who takes on the misdemeanor case of 22-year-old Michael Smith, who pleaded not guilty in one of the first body camera cases in the city when he was charged with nine counts of resisting arrest. This urgent documentary shows how far Adachi and his team will fight for the young man's freedom while exposing black-crime bias in ostensibly liberal SF.",
      "release_date": "2017-04-15"
    }
  ]
}
```

Slika 3. Rezultat search metode

## 2.2. Metoda discover

Za razliku od metode za pretraživanje, ova metoda nam služi kako bi se olakšalo pronalaženje odnosno otkrivanje novih filmova ili tv serija uz pomoć mnogih različitih parametara. Za otkrivanje filmova imamo mogućnost selektiranja sa preko 30 opcija gdje kod tv serija imamo samo 20, neke od glavnih opcija koje se koriste su odabir žanrova sa varijablom „*with\_genres*“ ukoliko želimo dohvatiti samo one rezultate koji imaju željene žanrove filmova odnosno serija ili varijablu „*without\_genres*“ ukoliko želimo isključiti neke od žanrova koje ne volimo iz pretrage. Ukoliko želimo navesti više žanrova iliti dodati više vrijednosti po kojima želimo filtrirati sadržaj, imamo na raspolaganju operatore „I“ i „II“ odnosno zarez koji se ponaša kao operator „I“ te okomitu crtu „|“ koja predstavlja operator „II“. Kao i sa žanrovima imamo i opcije za filtriranje pomoću ključnih riječi sa varijablama „*with\_keywords*“ te „*without\_keywords*“ koje se ponašaju jednako kao i sa žanrovima.

Ukoliko želimo pretraživati po varijablama koje ne primaju vrijednosti u obliku riječi, odnosno primaju vrijednosti kao brojeve poput filtriranja sadržaja prema broju glasova, prosjeku glasova, datumu izdanja ili dužini trajanja, potrebno je uvijek i dodatno specificirati dali želimo dohvatiti rezultate koji su manji od unesene vrijednosti ili veće, to možemo napraviti sa dodavanjem ključnih riječi „*gte*“ (engl. Greater than or equal to) ukoliko želimo dohvaćati rezultate koji imaju vrijednosti jednake odnosno veće od zadane ili „*lte*“ (engl. Less than or equal to) ukoliko želimo dohvatiti rezultate koji su manji ili jednaki vrijednosti prema kojoj obavljam filtriranje, na primjer ukoliko želimo samo sadržaj koji ima prosječnu ocjenu veću ili manju od 5 možemo koristiti varijable „*vote\_average.gte=5*“ odnosno „*vote\_average.lte=5*“.

Još jedna bitna mogućnost ove metode je i opcija sortiranja pomoću koje možemo jednostavno napisati po kojoj varijabli želimo da nam primljeni rezultati budu sortirani i dali želimo silazno ili uzlazno. Specificiranje uzlaznog ili silaznog sortiranja se izvršava jednako kao i u prethodnom slučaju kod dohvaćanja vrijednosti koje su manje ili veće od zadane, u ovom slučaju je potrebno dodati riječ „*asc*“ (engl. ascending) za uzlazno sortiranje ili „*desc*“ (engl. descending) za silazno sortiranje. Sortiranje se može izvršavati prema nekoliko različitih varijabli među kojima su sortiranja prema popularnosti, datumu izlaska, nazivu, prosječnoj ocjeni iliti sveukupnome broju ocjena i slično, na primjer ukoliko želimo sortirati prema nazivu možemo dodati varijabli „*sort\_by*“ vrijednosti „*original\_title.asc*“ za uzlazno sortiranje iliti „*original\_title.desc*“ ukoliko rezultate želimo imati sortirane silazno.

Ukoliko je potrebno, također imamo i mogućnost filtriranja rezultata pomoću certifikata sa varijablama „*certification*“ ili „*certification.lte*“, no ukoliko se koristi jedna od ovih varijabli također je obavezno navesti i zemlju određenog certifikata u varijablu „*certification\_country*“. Lista dostupnih certifikata u bazi se može dohvatiti sa metodom *certification* gdje specificiramo dali želimo dohvatiti filmove ili tv serije te da nam je potrebna lista kao što je vidljivo na slici 4.

```
// https://api.themoviedb.org/3/certification/movie/list?api_key=7369e06a600e15f985316d3189192793

{
  "certifications": {
    "US": [
      {
        "certification": "G",
        "meaning": "All ages admitted. There is no content that would be objectionable to most parents. This is one of only two ratings dating back to 1968 that still exists today.",
        "order": 1
      },
      {},
      {},
      {},
      {},
      {}
    ],
    "CA": [{}],
    "AU": [{}],
    "DE": [
      {
        "certification": "0",
        "meaning": "No age restriction.",
        "order": 1
      },
      {},
      {},
      {},
      {}
    ],
  }
}
```

Slika 4. Popis certifikata

Na slici 5. se nalazi primjer jednog jednostavnog zahtjeva u kojemu se pretražuju filmovi koji su na engleskome jeziku, sortirani silazno prema prosječnoj ocjeni te imaju akcijski i avanturistički žanr sa identifikacijskim brojevima 28 i 12. Kao što se vidi sa slike rezultat koji smo zaprimili sadrži jednake vrijednosti kao što je to sadržavao i rezultat metode *search*, izuzevši vrijednost „*media\_type*“ koju u ovom slučaju nije bilo potrebno prikazivati sa obzirom na to da smo naveli u samoj *discover* metodi da želimo pretraživati samo filmove.

```

1 // 20170825162636
2 // https://api.themoviedb.org/3/discover/movie?api_key=7369e06a600e15f985316d3189192793&language=en-US&sort_by=vote_average.desc&with_genres=28,12
3
4 {
5   "page": 1,
6   "total_results": 4231,
7   "total_pages": 212,
8   "results": [
9     {
10      "vote_count": 1,
11      "id": 52068,
12      "video": false,
13      "vote_average": 10,
14      "title": "Pendragon: Sword of His Father",
15      "popularity": 1.021062,
16      "poster_path": "/x1Qhd1VvWJ3NDG8s7vfY6KJbIAO.jpg",
17      "original_language": "pt",
18      "original_title": "Pendragon: Sword of His Father",
19      "genre_ids": [
20        12,
21        14,
22        28,
23        878
24      ],
25      "backdrop_path": "/1SkZa4sPYRwdt09J7FGPhjUmw5Z.jpg",
26      "adult": false,
27      "overview": "Set in 411 AD, Pendragon tells the story of young Artos who is raised to believe that God has a purpose for each day. When his family killed and he is taken into slavery by the Saxons, Artos questions his God. Advancing through the military ranks, Artos begins to understand that his father's vision was not based on the strength of man, but on the plan of God. Further betrayal by his friends forces Artos to decide between following God's plan unto certain death or abandoning God to save his own life.",
28      "release_date": "2008-11-01"
29    },
30    {
31      "vote_count": 1,
32      "id": 142128,
33      "video": false,
34      "vote_average": 10,
35      "title": "Minds in the Water",

```

Slika 5. Primjer discover metode

## 2.3. Metoda find

Iako se unutar izrađenog primjera aplikacije ne koristi ova metoda, smatram da je ipak poprilično korisna te da zaslužuje biti spomenuta. Koristeći ovu metodu možemo vrlo lako pronalaziti informacije o filmovima, tv serijama ili ljudima koristeći se identifikacijskim brojevima sa drugih web stranica poput IMDb-a, TVDb-a, Freebase-a te TVRage-a. Za izradu zahtjeva je potrebno samo specificirati identifikacijsku oznaku sa eksterne stranice u varijablu „*external\_id*“ i izvor na kojemu se određena oznaka nalazi u varijablu „*external\_source*“. Ova metoda također ima i dodatnu mogućnost izbora jezika no ona nije obavezna te će zahtjev funkcionirati i bez odabiranja jezika. Na slici 6 se nalazi jedan primjer find metode za tv seriju čija je oznaka dohvaćena sa TVDb stranice. Kao što je vidljivo na primjeru rezultat je dohvatio podatke o traženoj tv seriji te se vrijednosti nalaze

u polju pod nazivom „*tv\_results*“ ostala vidljiva polja su prazna te da smo unijeli oznaku od neke osobe ili filma onda bi se naravno popunila ta polja te bi ostala bila prazna.

```
2 // https://api.themoviedb.org/3/find/281662?api_key=7369e06a600e15f985316d3189192793&language=en-US&external_source=tvdb_id
3
4 {
5   "movie_results": [],
6   "person_results": [],
7   "tv_results": [
8     {
9       "backdrop_path": "/dpNeXLEnuKzAvbNwveJhNEiQvXZ.jpg",
10      "first_air_date": "2015-04-10",
11      "genre_ids": [
12        28,
13        80
14      ],
15      "id": 61889,
16      "original_language": "en",
17      "original_name": "Marvel's Daredevil",
18      "overview": "Lawyer-by-day Matt Murdock uses his heightened senses from being blinded as a young boy to fight crime at night on the streets of Hell's Kitchen as Daredevil.",
19      "origin_country": [
20        "US"
21      ],
22      "poster_path": "/cid0qJL8tayqv3TpFTQCsgEITu.jpg",
23      "popularity": 11.535192,
24      "name": "Marvel's Daredevil",
25      "vote_average": 7.6,
26      "vote_count": 740
27    }
28  ],
29  "tv_episode_results": [],
30  "tv_season_results": []
31 }
```

Slika 6. Primjer find metode

## 2.4. Metoda movie

Glavna svrha ove metode je dohvaćanje raznih detalja o odabranom filmu ili određenih lista sa filmovima, no kako bi se ova metoda mogla koristiti za dohvaćanje detalja o filmu prvo nam je potreban odgovarajući identifikacijski broj koji je naravno specifičan za svaki film te ga možemo dohvatiti korištenjem jedne od prethodno navedenih metoda za pretraživanje iliti otkrivanje filmova. Sa obzirom da je ovo veoma opširna metoda te obuhvaća razne mogućnosti ovdje ću pojasniti samo nekoliko bitnijih opcija.

Ukoliko ne specificiramo nikakve dodatne opcije već samo iskoristimo oznaku filma ova metoda će nam vratiti određene detalje o filmu, malo opširnije te sa više informacija od

prethodnih metoda za pretraživanja ali još nedovoljno ukoliko želimo izraditi stranicu sa detaljima o filmu, kao što se može vidjeti sa slike 7.

```
// https://api.themoviedb.org/3/movie/22?api_key=7369e06a600e15f985316d3189192793&language=en-US

{
  "adult": false,
  "backdrop_path": "/8AUQ7YlJJ9C8k8P4YMHicFDE.jpg",
  "belongs_to_collection": {↵},
  "budget": 140000000,
  "genres": [↵],
  "homepage": "http://disney.go.com/disneyvideos/liveaction/pirates/main_site/main.html",
  "id": 22,
  "imdb_id": "tt0325980",
  "original_language": "en",
  "original_title": "Pirates of the Caribbean: The Curse of the Black Pearl",
  "overview": "Jack Sparrow, a freewheeling 17th-century pirate who roams the Caribbean Sea, butts heads with a rival pirate bent on pillaging the village of Port Royal. When the governor's daughter is kidnapped, Sparrow decides to help the girl's love save her. But their seafaring mission is hardly simple.",
  "popularity": 12.583752,
  "poster_path": "/tkt9xrl1kNX5R9rCebASKck44si2.jpg",
  "production_companies": [↵],
  "production_countries": [↵],
  "release_date": "2003-07-09",
  "revenue": 655011224,
  "runtime": 143,
  "spoken_languages": [↵],
  "status": "Released",
  "tagline": "Prepare to be blown out of the water.",
  "title": "Pirates of the Caribbean: The Curse of the Black Pearl",
  "video": false,
  "vote_average": 7.5,
  "vote_count": 6697
}
```

Slika 7. Dohvaćanje detalja o filmu

Ukoliko želimo dohvatiti različite nazive na stranim jezicima određenog filma, te informacije možemo dohvatiti korištenjem opcije „*alternative\_titles*“ nakon što smo unijeli oznaku filma. Kao što se vidi sa slike 8. izvršavanjem ovoga zahtjeva dobili smo listu u kojoj su nam zapisani jezici te njihovi pripadajući nazivi za odabrani film.

```
// https://api.themoviedb.org/3/movie/22/alternative_titles?api_key=7369e06a600e15f985316d3189192793

{
  "id": 22,
  "titles": [
    {
      "iso_3166_1": "TW",
      "title": "神鬼奇航：鬼盜船魔咒"
    },
    {
      "iso_3166_1": "IT",
      "title": "Pirati dei Caraibi - La maledizione della prima luna"
    },
    {
      "iso_3166_1": "ES",

```

Slika 8. Dohvaćanje alternativnih naziva

Ukoliko želimo dobiti informacije o svim članovima tima koji su radili na određenom filmu možemo iskoristiti opciju „credits“ koja nam vraća ukratko koja je uloga bila određene osobe na filmu, te na slici 9. možemo vidjeti da su u samome rezultatu odvojeni glumci od ostatka tima te je također dostupan i identifikacijski broj preko kojega možemo, ukoliko je to potrebno doći do još više detalja o pojedinim osobama.

Putem ove metode uz korištenje opcije „keyword“ možemo dohvatiti sve ključne riječi vezane uz film ili ukoliko želimo možemo i uz pomoć opcije „release\_dates“ dohvatiti sve

```
// https://api.themoviedb.org/3/movie/22/credits?api_key=7369e06a600e15f985316d3189192793

{
  "id": 22,
  "cast": [↔],
  "crew": [
    {
      "credit_id": "52fe420fc3a36847f8000f11",
      "department": "Editing",
      "gender": 2,
      "id": 38,
      "job": "Editor",
      "name": "Arthur Schmidt",
      "profile_path": null
    },
    {
      "credit_id": "52fe420fc3a36847f8000e95",
      "department": "Camera",
      "gender": 2,
      "id": 120,
      "job": "Director of Photography",
      "name": "Dariusz Wolski",
      "profile_path": null
    }
  ],
}
```

Slika 9. Dohvaćanje detalja o osobama

relevantne informacije uz objavljivanje filma u različitim zemljama kao što se može vidjeti na slici 10, u rezultatu su prikazani i tipovi objave koji označavaju dali je određena objava filma bila premijera, kino, digitalna ili tv, tip 3 koji se može vidjeti na slici označava da se radi o kinu. Ovom opcijom također možemo i saznati koji su certifikati bili prisutni za vrijeme objave filma u određenim zemljama, no međutim taj podatak nije uvijek zapisan za sve zemlje kao što se može vidjeti iz slike.

```
// https://api.themoviedb.org/3/movie/22/release_dates?api_key=7369e06a600e15f985316d3189192793

{
  "id": 22,
  "results": [
    {
      "iso_3166_1": "BE",
      "release_dates": [
        {
          "certification": "",
          "iso_639_1": "",
          "release_date": "2003-08-13T00:00:00.000Z",
          "type": 3
        }
      ]
    },
    {
      "iso_3166_1": "RS",
      "release_dates": [
        {
          "certification": "",
          "iso_639_1": "",
          "release_date": "2003-10-09T00:00:00.000Z",
          "type": 3
        }
      ]
    }
  ],
}
```

Slika 10. Objave filmova po zemljama

Ukoliko želimo dohvatiti slike ili videa od određenog filma to možemo učiniti sa opcijama „*images*“ ili „*videos*“ koje nam vraćaju listu dostupnih slika i videa sa njihovim poveznicama kako bismo ih mogli prikazati, u nastavku rada detaljnije ću prikazati kako se dohvaćaju i prikazuju slike i videa. Još jedna od bitnih mogućnosti ove metode je i dohvaćanje sličnih filmova te filmova koji su preporučeni od strane drugih korisnika TMDb-a, te informacije možemo dohvatiti putem opcija „*recommendations*“ za dohvaćanje korisničkih preporuka ili „*similar*“ za dohvaćanje sličnih filmova koji se sastavljaju gledajući na žanrove i ključne riječi pojedinog filma. Rezultat odnosno informacije koje dobijemo ovim zahtjevom su jednake kao i da smo koristili neku od metoda za pretraživanje tako da ukoliko želimo ponovno dohvatiti dodatne detalje o filmovima dohvaćenim ovim putem morati ćemo ponovno dohvatiti njihove identifikacijske oznake te ponovno koristiti ovu metodu.

Osim opcija koje sve zahtijevaju korištenje identifikacijske kako bismo dobili dodatne informacije o odabranom filmu, također imamo i metode koje nam vraćaju određene popise iliti liste filmova, poput nedavnih filmova sa opcijom „*latest*“, popularnih filmova na TMDb-u putem opcije „*popular*“, filmova koji trenutno izlaze u kinima sa opcijom



„*upcoming*“ ili korištenjem opcije „*top\_rated*“ možemo dohvatiti trenutno najbolje ocijenjene filmove na TMDb-u.

## 2.5. Spajanje zahtjeva

Sa obzirom da je broj zahtjeva kao što sam prethodno spomenuo, limitiran na 40 zahtjeva svakih deset sekundi, te iako se taj broj u početku čini poprilično veliki i doima se kao da se neće moći prekoračiti, ipak se može naslutiti iz prethodno prikazane metode da ukoliko želimo određene detalje prikazati korisniku ponekad je potrebno iskoristiti različite opcije te poslati više zahtjeva kako bismo dobili sve potrebne informacije. Na primjer, jedna stranica koja pokazuje detalje vrlo vjerojatno treba prikazati ne samo informacije o zadanom filmu za što nam je potreban jedan zahtjev već bit trebala i prikazivati određene informacije o glumcima što nam dodaje još jedan zahtjev, nadalje može prikazivati i kritike od korisnika što nam stvara treći zahtjev, preporučene te slične filmove što nam već podiže broj zahtjeva na pet, taj broj ovisno o potrebama korisnika i aplikacije može još znatno narasti te ukoliko se neispravno pozivaju odnosno koriste metode lako je moguće prekoračiti postojeći limit.

Kako bi se izbjeglo pretjerano slanje sličnih zahtjeva te time i izbjeglo prekoračenje limita omogućeno je i spajanje određenih upita. Metode koje podržavaju spajanje su metode koje služe za dohvaćanje detalja bilo to o filmu, ljudima, tv seriji, sezoni ili pojedinoj epizodi. Parametar koji trebamo dodati ukoliko želimo dodatne zahtjeve odnosno odgovore se naziva „*append\_to\_response*“ te mu možemo dodjeljivati vrijednosti koje se mogu koristiti kao opcije prilikom potrage za detaljima o filmovima ili sličnom sadržaju.

Na slici 11 se može vidjeti rezultat zahtjeva u kojemu su se tražili detalje o filmu, video koja su vezana uz odabrani film, slike, slične filmove te osobe koje su radile na tom filmu, time smo umjesto da pet puta šaljemo zasebne zahtjeve te dobivali pet različitih odgovora koja bismo trebali koristiti, spojili smo u jedan zahtjev koji nam je vratio sve relevantne informacije koje smo tražili i to u jednom odgovoru.

```
// https://api.themoviedb.org/3/movie/22?
api_key=7369e06a600e15f985316d3189192793&append_to_response=videos,images,similar,credits

{
  "adult": false,
  "backdrop_path": "/8AUQ7Y1JJJA9C8kKw8P4YNHicFDE.jpg",
  "belongs_to_collection": {},
  "budget": 140000000,
  "genres": [],
  "homepage": "http://disney.go.com/disneyvideos/liveaction/pirates/main_site/main.html",
  "id": 22,
  "imdb_id": "tt0325980",
  "original_language": "en",
  "original_title": "Pirates of the Caribbean: The Curse of the Black Pearl",
  "overview": "Jack Sparrow, a freewheeling 17th-century pirate who roams the Caribbean Sea, butts heads with a rival pirate bent on pillaging the village of Port Royal. When the governor's daughter is kidnapped, Sparrow decides to help the girl's love save her. But their seafaring mission is hardly simple.",
  "popularity": 12.583752,
  "poster_path": "/tkt9xR1kNXSR9rCebASKck44si2.jpg",
  "production_companies": [],
  "production_countries": [],
  "release_date": "2003-07-09",
  "revenue": 655011224,
  "runtime": 143,
  "spoken_languages": [],
  "status": "Released",
  "tagline": "Prepare to be blown out of the water.",
  "title": "Pirates of the Caribbean: The Curse of the Black Pearl",
  "video": false,
  "vote_average": 7.5,
  "vote_count": 6698,
  "videos": {},
  "images": {},
  "similar": {},
  "credits": {}
}
```

Slika 11. Spajanje zahtjeva

## 2.6. Metoda genre

Prilikom pretraživanja filmova ili dohvaćanja detalja o filmu uvijek ima dostupno polje u kojemu su zapisani pripadajući žanrovi, no sa obzirom da se u tom polje nalaze smo identifikacijski brojevi žanrova a krajnjem korisniku ti brojevi ne znače mnogo, potrebno je povezati pripadajuće nazive sa njihovim identifikacijskim brojevima. Cijeli popis postojećih žanrova u bazi možemo dohvatiti pomoću metode za žanrove te odabirom opcije „list“, čiji se primjer te dio rezultata može vidjeti na slici 12. Ukoliko želimo dodatno specificirati pretragu možemo dodati opciju „movie“ ili „tv“ između poziva žanra i liste, npr. „/genre/movie/list...“.

```
// http://api.themoviedb.org/3/genre/list?api_key=7369e06a600e15f985316d3189192793

{
  "genres": [
    {
      "id": 28,
      "name": "Action"
    },
    {
      "id": 12,
      "name": "Adventure"
    },
    {
      "id": 16,
      "name": "Animation"
    },
    {
      "id": 35,
      "name": "Comedy"
    },
  ],
}
```

Slika 12.Lista žanrova

Ova metoda nam također i omogućuje pretraživanje iliti otkrivanje novih filmova i serija putem oznake nekog od žanrova. Na primjer ukoliko želim pronaći sve filmove koji imaju avanturistički žanr, URL bi trebao sadržavati „/3/genre/12/movies?api\_key=...“.

## 2.7. Dohvaćanje slika

Kako bismo prikazali određenu sliku potrebna su nam tri podatka, početni URL, veličinu datoteke odnosno slike te putanju do slike. Putanja slike se može dohvatiti jednostavnim pretraživanjima kao što se može vidjeti na prethodnim primjerima na mjestu „*poster\_path*“ i sličnih vrijednosti, no za razliku od svih prijašnjih metoda početni URL više ne poziva API tako da više nije potrebno konstantno ispisivati ključ te je potrebno koristiti malo drugačiju verziju URL-a „*https://image.tmbd.org/t/p/*“.

Ukoliko nismo sigurni koja je točna veličina slike ili ukoliko ne znamo koji je početni URL za slike u slučaju da je došlo do nekakvih izmjena možemo pozvati metodu „*configuration*“ u kojoj se nalazi početni URL te veličine za različite tipove slika, kao što se može i vidjeti na slici 13., no ukoliko nam nije toliko bitna veličina možemo jednostavno postaviti vrijednost na

„original“ veličinu sa obzirom da tu veličinu imaju sve slike. Konačan primjer putanje za neku sliku bi trebao biti nalik na

„<https://image.tmbd.org/t/p/w500/q0R4crx2SehcEEQEkYObktdeFy.jpg>“.

```
// https://api.themoviedb.org/3/configuration?api_key=7369e06a600e15f985316d3189192793

{
  "images": {
    "base_url": "http://image.tmbd.org/t/p/",
    "secure_base_url": "https://image.tmbd.org/t/p/",
    "backdrop_sizes": [
      "w300",
      "w780",
      "w1280",
      "original"
    ],
    "logo_sizes": [
      "w45",
      "w92",
      "w154",
      "w185",
      "w300",
      "w500",
      "original"
    ],
    "poster_sizes": [
      "w92",
      "w154",
      "w185",
      "w342",
      "w500",
      "w780",
      "original"
    ]
  },
}
```

Slika 13. Konfiguracija

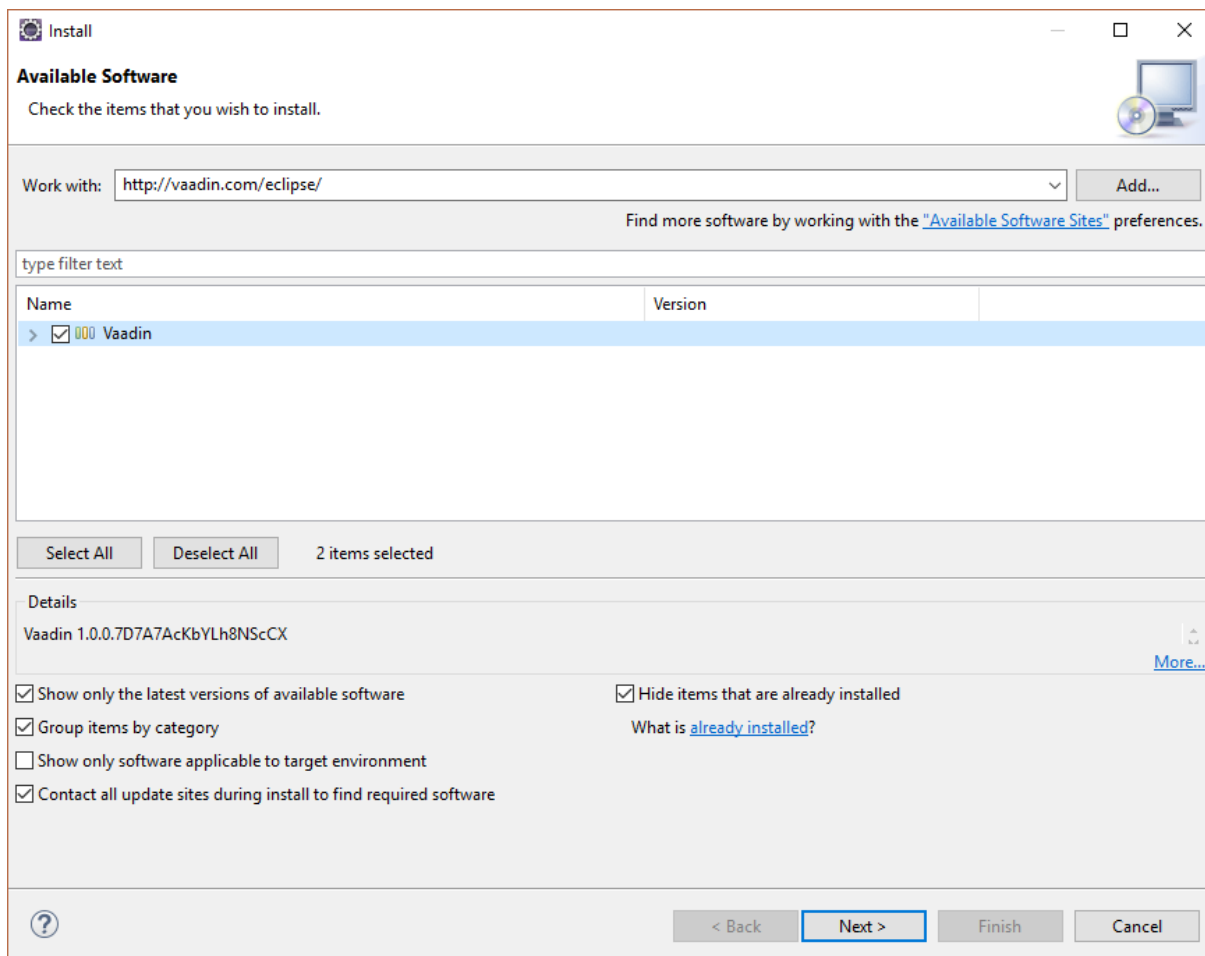
### 3. Alati i tehnologije

Tokom izrade aplikacije za pregled i pretraživanje filmova, osim TMDb API-a koristio sam naravno i druge razne tehnologije te ću kroz ovo poglavlje pokazati i pojasniti svrhu pojedinih korištenih tehnologija te prikazati kako se određeni alati mogu instalirati te pripremiti za korištenje. Glavna tehnologija odnosno programski jezik koji sam koristio prilikom izrade aplikacije je java uz pomoć okvira Vaadin koji uvelike olakšava izradu web korisničkog sučelja u javi. Razvojno okruženje koje sam koristio je Eclipse te sam također koristio i github kako bih mogao spremati verzije koda, na kraju za potrebe baze podataka sam koristio MySQL u kombinaciji sa phpMyAdmin-om te XAMPP-om kako bih si uspostavio lokalno bazu.

#### 3.1. Eclipse

Eclipse je razvojno okruženje za programiranje odnosno razvoj aplikacija koje je veoma poznato po tome što se njime služi veliki broj java programera, to je besplatan, stabilan i poprilično kvalitetan alat za izradu aplikacija te je njegova sama uspostava veoma jednostavna kao i samo njegovo korištenje. Kako bismo instalirali alat potrebno je uglavnom imati instaliranu javu te naravno skinuti i sami Eclipse, instalacija nema nikakvih posebnih koraka te se mogu pratiti ponuđeni koraci te do problema ne bi trebalo doći. Prilikom pokretanja potrebno je navesti direktorij u kojemu želimo raditi odnosno spremati si projekte koje ćemo izrađivati.

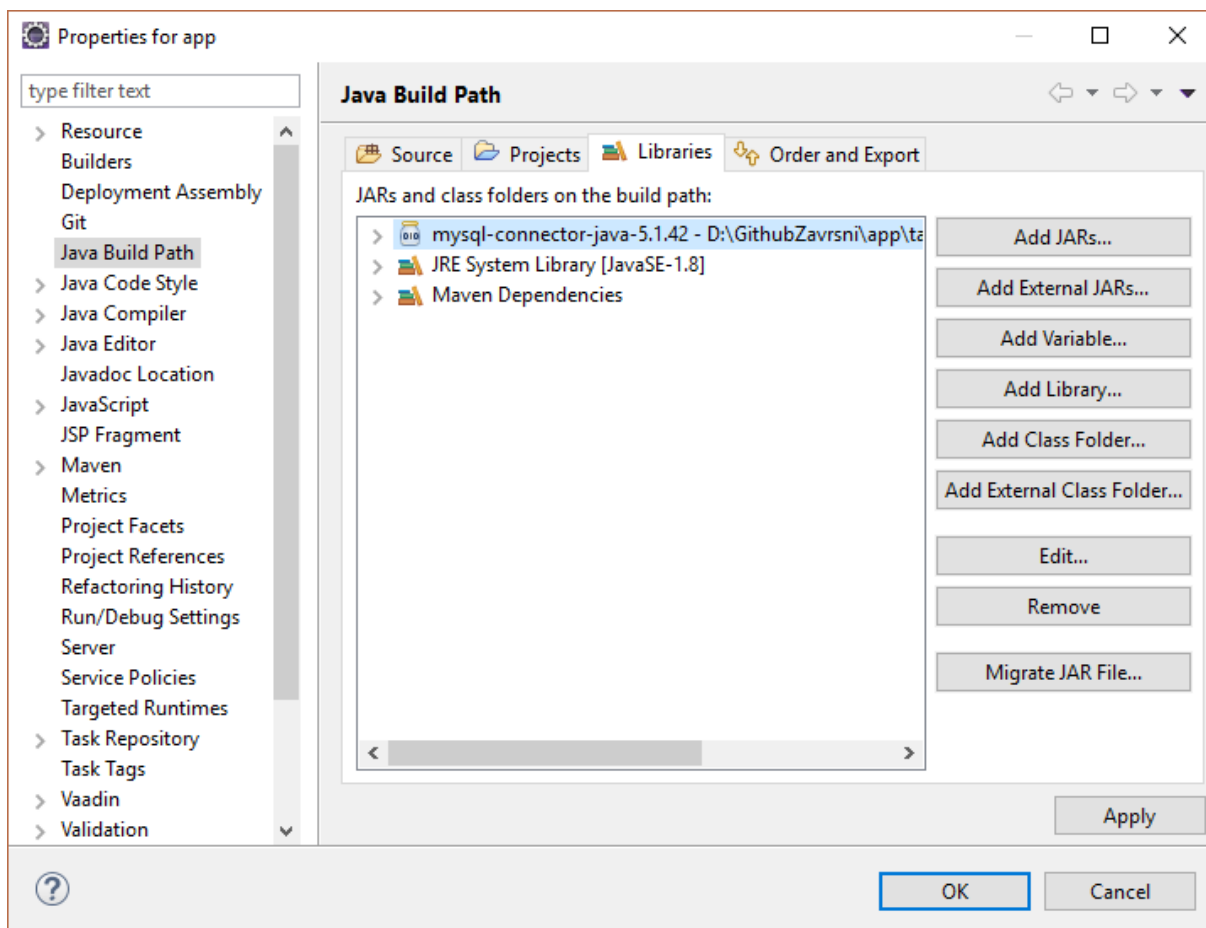
Eclipse također podržava mnoštvo dodataka iliti proširenja koja nam mogu olakšati izradu željene aplikacije te ukoliko želimo instalirati određeni dodatak to možemo učiniti odlaskom na pomoć te odabirom opcije za instaliranje novih aplikacija što će prikazati prozor sa slike 14, na kojemu možemo odabrati što želimo instalirati putem određenog linka te unutar par koraka možemo jednostavno koristiti već postojeće dodatke za razvojno okruženje. Ovim putem sam za svrhe aplikacije instalirao dodatke koji omogućuju korištenje Vaadin okvira za razvoj aplikacija te github dodatak koji omogućuju jednostavno spremanje te verzioniranje projekata.



Slika 14. Prozor za dodavanje proširenja

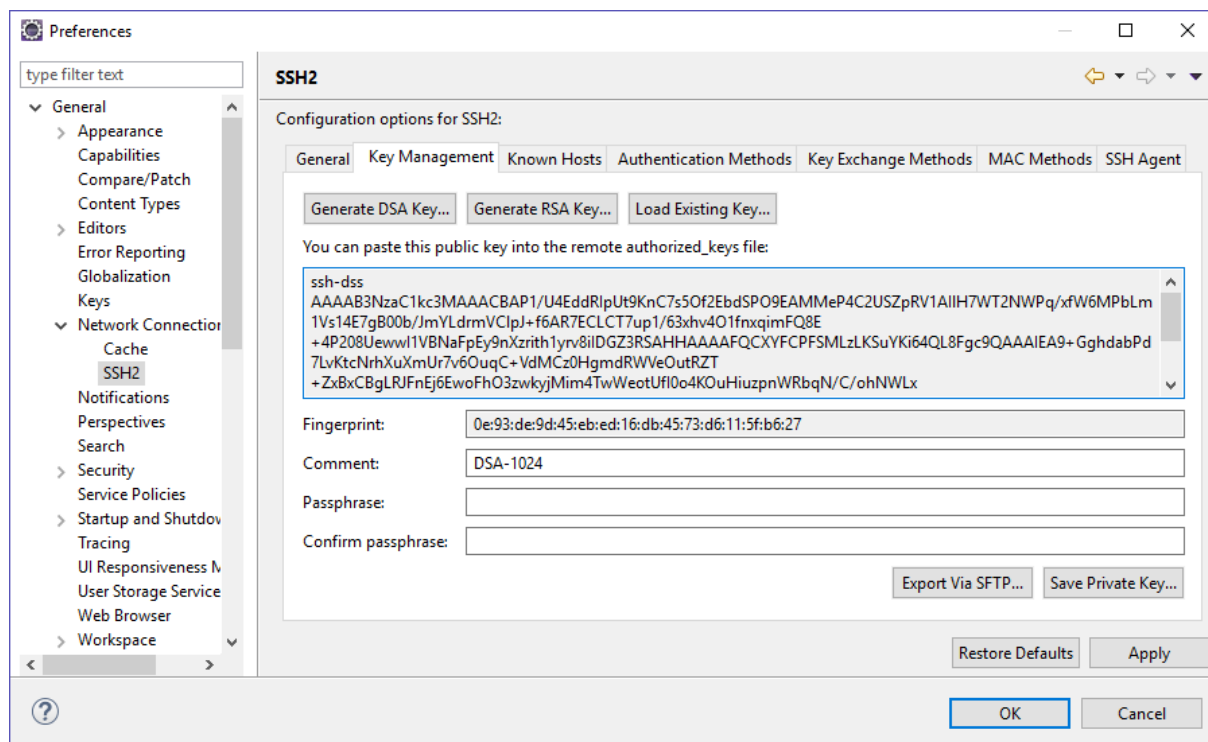
Kako bismo započeli novi projekt možemo jednostavno otići na „file“ te odabrati da želimo izraditi novi projekt te jednostavno odabrati što želimo uključiti od opcija te napisati naziv projekta, još nešto što bih volio napomenuti je da nakon uspostave Eclipse-a biti će označena opcija „build automatically“ koja nam omogućuje da svaki puta dok radimo nešto u Eclipse-u i spremimo sadržaj koji smo napravili, Eclipse će ponovno „izgraditi“ odnosno pokrenuti aplikaciju što može postati zamorno te bih u ovom slučaju preporučio da se projekt ipak „gradi“ ručno što se može ili opcijom „build all“ kod opcija za projekte iliti pritiskom na tipke **ctrl + B** što je zadana kombinacija tipki za navedenu opciju.

Ukoliko želimo dodavati nove jar-ove u projekt možemo jednostavno otići na pregled paketa te označiti željeni projekt i pritiskom desne tipke miša odabrati opciju za svojstva gdje možemo doći do mogućnosti namještanja putanja koje se koriste prilikom pokretanja projekta kao što se vidi na slici 15. Kao što se vidi sa slike tu je dodani eksterno jar za spajanje sa bazom, ukoliko želimo dodati još neke posebne vanjske jar-ove to možemo učiniti putem opcija „Add JARs...“ iliti „Add External JARs...“.



Slika 13. Dodavanje jar-ova

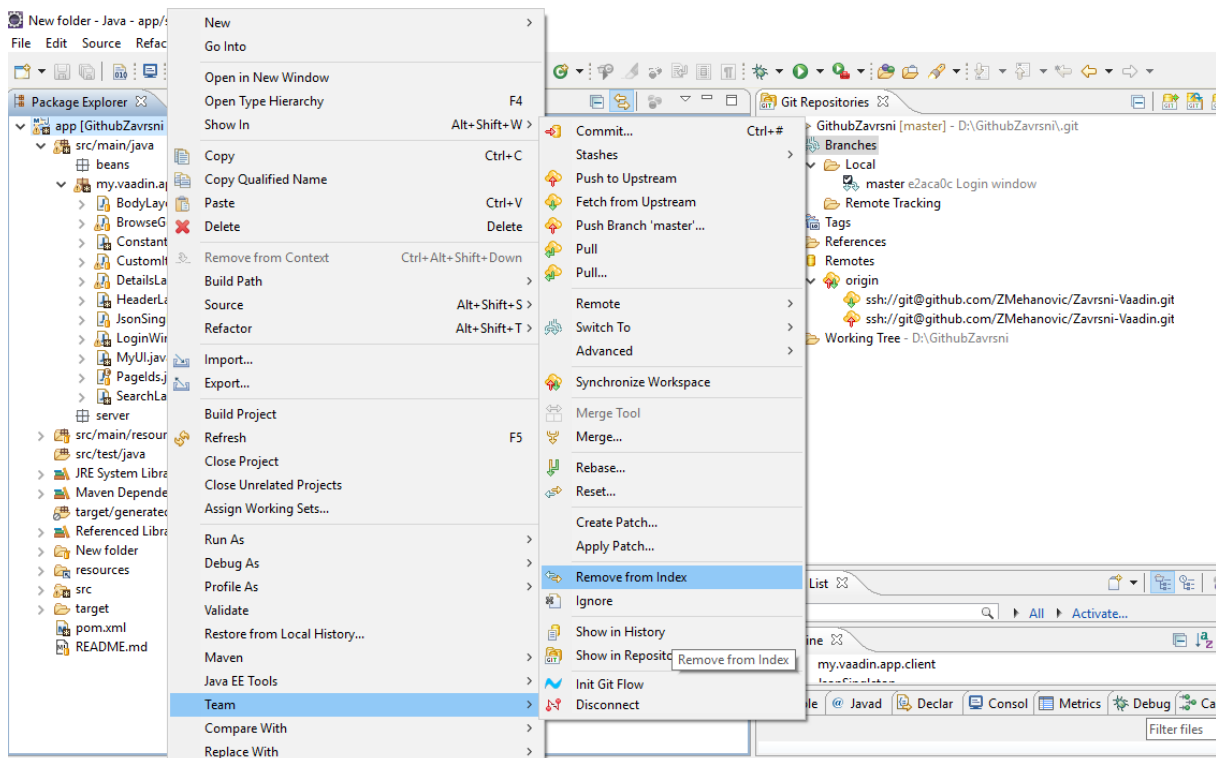
Jedan poprilično koristan dodatak koji možemo koristiti u Eclipse-u je proširenje za github uz pomoću kojega se možemo povezati sa našim računom na githubu te jednostavno spremati verzije koda putem razvojnog okruženja u kojemu radimo. Kako bismo dodali github proširenje prvo je potrebno putem prozora za dodavanje proširenja sa slike 16 pretražiti „egit“ te instalirati odgovarajući dodatak za Eclipse, nakon instalacije potrebno je kreirati te spremiti privatni ključ koji se koristi kako bi se mogli spojiti sa našim github računom. Kako bismo dobili vlastiti privatni ključ potrebno je otići do postavki putem izbornika „window“ te odabirom na „preferences“ nam se prikazuje novi prozor na kojemu trebamo doći putem generalnih postavki te mrežnih postavki do opcije za „SSH2“ te je na prikazanom prozoru potrebno odabrati „key management“ gdje možemo dobiti vlastiti ključ, kao što je prikazano na slici 17, generirani ključ je poželjno spremiti na sigurno mjesto te ga je potrebno dodati u ključeve na vlastitom github računu.



Slika 14. Prozor za generiranje DSA ključa

Nakon što smo spremili privatni ključ, potrebno je kreirati lokalni git repozitorij koji možemo napraviti putem lijevog izbornika unutar Eclipse-a desnim klikom na željeni projekt i odabirom pod izbornika za tim te opcije za dijeljenje projekta što nam otvara novi prozor putem kojega možemo napraviti novi lokalni repozitorij. Nakon što smo napravili lokalni repozitorij u izborniku za tim imamo ponuđeno mnogo više opcija negoli prethodno, slijedeći korak je početni „commit“ gdje će nas zatražiti za korisničko ime te email. Slijedeće što je potrebno je uspostaviti vanjski iliti udaljeni (engl. remote) repozitorij koji možemo namjestiti odlaskom na pregled repozitorija gdje možemo uočiti da nam je lokalni repozitorij već vidljiv te da pod vanjskim repozitorijima nema još nikakvih podataka. Kako bismo kreirali novi vanjski repozitorij možemo jednostavno desnim klikom miša odabrati opciju za kreiranje novog vanjskog repozitorija gdje prvo unosimo naziv te zatim dodajemo podatke iz našeg github računa poput URI-a te korisničkog imena. Na slici 17 je vidljiv pregled repozitorija te dostupne opcije za korištenje Eclipse git proširenja.





Slika 17.. Eclipse github proširenje

## 3.2. Java

Java je jedan od trenutno najpopularnijih objektno orijentiranih programskih jezika koji je prvotno nastao 1991. godine te je bio napravljen od strane jednog malog tima sa namjerom da se koristi na ručnim uređajima i to pod nazivom OAK (Beal Vangie, s.a.). Međutim OAK nije zaživio te ga je kompanija Sun Microsystems 1995. godine modificirala kako bi jezik mogao iskoristiti prednosti interneta te mu je dodijelila naziv Java. U početku Java je bila dizajnirana kako bi se mogla pokretati na virtualnim strojevima odvojenima od fizičkih kako bi se implementirao pristup napiši jednom pokreni bilo gdje (engl. Write Once Run Anywhere - WORA) (Park Se Hoon, 2017).

Uz javu se često spominju i drugi pojmovi odnosno kratice poput JVM (engl. Java virtual machine), JRE (engl. Java runtime environment) te JDK (engl. Java development toolkit) ili SDK (engl. Software development toolkit). JVM ili java virtualni stroj omogućuje našem računalu pokretanje java programa, prilikom pokretanje java programa, java kompajler prvo prevodi java kod u bajt kod te zatim JVM dalje prevodi u strojni jezik. JRE je okruženje u kojemu se izvršavaju java programi te ukoliko nam nije cilj izrada aplikacija ovo je sve što nam je potrebno, JRE je u biti paket koji sadrži java biblioteke, JVM te druge komponente koje su potrebne za pokretanje programa. JDK je u biti skup alata koji su potrebni pri izradi aplikacija

u javi, kada instaliramo JDK uz to instaliramo i JRE te ih nije potrebno skidati zasebno, jedina razlika između JDK i SDK je ta da se SDK odnosi i na druge programske jezike osim java. Set klasa koje se nalaze unutar JDK-a se također nazivaju Java API (Berlind David, 2015)

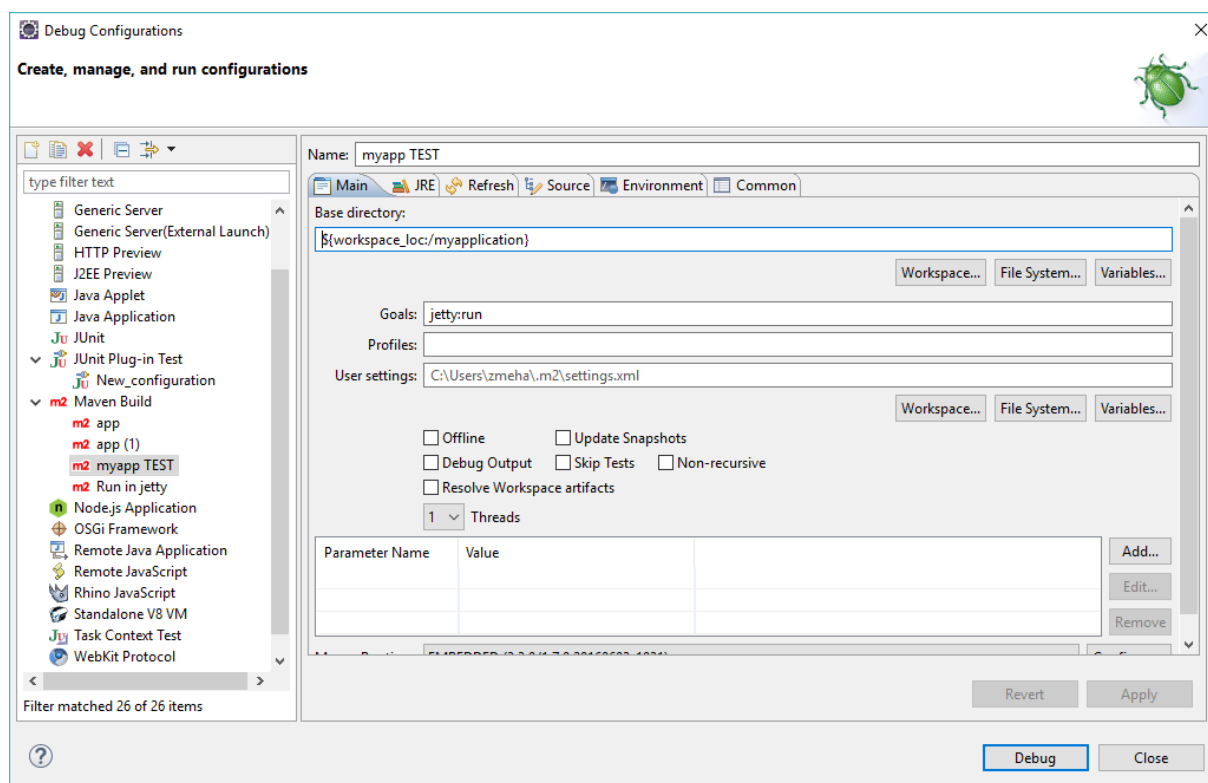
### 3.3. Vaadin

Vaadin je jedan od mnogih dostupnih okvira (engl. framework) za izradu java aplikacija te olakšava izradu samog korisničkog sučelja na web aplikacijama. Vaadin je nastao još 2000. godine te se konstantno razvija i nadograđuje, također još uvijek ima poprilično aktivnu zajednicu koja i sama doprinosi poboljšanju kvalitete Vaadin-a bilo to kreiranjem raznih dodataka koji su dostupni putem službene stranice ili prisustvovanjem na forumima. Vaadin nam omogućuje izradu izgleda web stranice te se pobrine za samu asinkronu komunikaciju između preglednika i servera tako da i na taj način olakšava programerima posao pošto ne moraju učiti niti se baviti sa tehnologijama koje su namijenjene za preglednike poput HTML-a ili JavaScripta već se može potpuno posvetiti samoj logici aplikacije što uvelike ubrzava proces izrade web aplikacije (Vaadin Team, 2017).

Jedna od prednosti Vaadin-a naspram ostalih opcija poput GWT-a (engl. Google web toolkit) ili sličnih okvira koji služe za olakšavanje izrade web aplikacija u javi je ta da je sama arhitektura Vaadin-a fokusirana na serversku stranu što nam dodatno omogućuje korištenje biblioteka namijenjenih javi te čak i olakšava sam proces pronalaženja grešaka u kodu pošto je samo praćenje tijeka koda omogućeno i putem Eclipsea te ne moramo koristiti varijante sa web preglednika. GWT je poprilično koristan zato što ima poprilično dobru podršku za različiti web preglednike pošto on mijenja kod ovisno o korištenom pregledniku, to nas dovodi do još jedne prednosti Vaadina koji u pozadini koristi GWT tako da sve preglednike koje podržava GWT također podržava i Vaadin, tako da možemo razvijati aplikaciju bez brige o različitim preglednicima pošto će se za same razlike pobrinuti GWT (Frankel Nicolas, 2011).

Kako bismo koristili Vaadin potrebno je prvo instalirati Vaadin dodatak za Eclipse koji možemo jednostavno pronaći direktno iz Eclipsea kao što je prikazano u prethodnome poglavlju. Nakon instalacije možemo vidjeti da kada idemo na kreiranje projekta imamo dodatnu opciju za kreiranje Vaadin projekta ,također je potrebno postaviti i konfiguraciju za pokretanje programa što možemo učiniti tako što odemo u postavke te kreiramo novi „*Maven build*“ preko kojega ćemo pokretati aplikaciju zatim odredimo putanju do projekta te pod cilj upišemo „*jetty:run*“ kako bi aplikacija mogla funkcionirati putem jetty-a kao što se može vidjeti na slici 18, još je samo potrebno otići na izbornik sa izvorima te odabrati java projekt i selektirati

novo kreirani projekt, treba pripaziti na to da se prilikom izbora odabere java projekt pošto ima i opcija pod nazivom projekt.



Slika 15. Konfiguracija za pokretanje programa

Nakon kreiranja i konfiguriranja projekta možemo uočiti da nam je kreirana i jedna početna klasa sa nazivom „MyUT“ odakle sve što se odvija u aplikaciji počinje. U generiranoj klasi već imamo neke jednostavne primjere za korištenje elementa iz Vaadin-a koje možemo odmah pokrenuti te vidjeti kako rade, slike 19, 20.

```

@Theme("mytheme")
public class MyUI extends UI {

    @Override
    protected void init(VaadinRequest vaadinRequest) {
        final VerticalLayout layout = new VerticalLayout();

        final TextField name = new TextField();
        name.setCaption("Type your name here:");

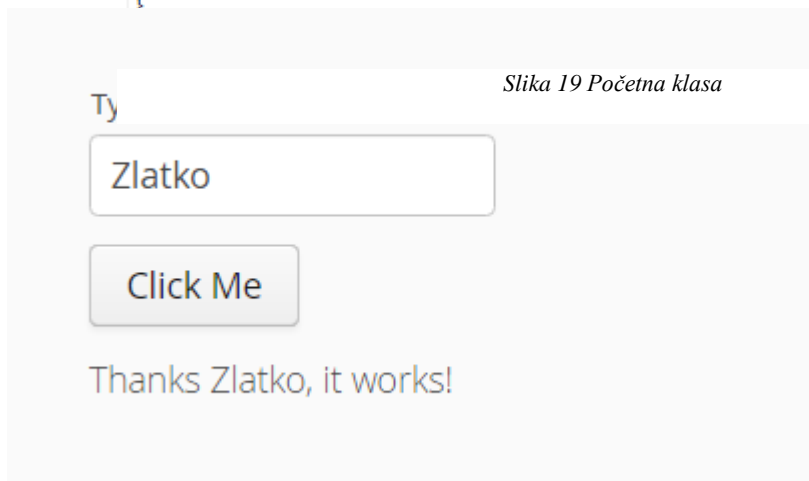
        Button button = new Button("Click Me");
        button.addClickListener( e -> {
            layout.addComponent(new Label("Thanks " + name.getValue()
                + ", it works!"));
        });

        layout.addComponents(name, button);

        setContent(layout);
    }

    @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
    @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
    public static class MyUIServlet extends VaadinServlet {
    }
}

```



Slika 20. Početni primjer

Kao što se može zaključiti iz prethodnih primjera, metoda „*init*“ je ulazna odnosno početna metoda u koju se zalazi prilikom otvaranja aplikacije te se upravo kroz tu metodu treba aplikacija dalje razvijati. U navedenom primjeru možemo uočiti četiri elementa iz Vaadin-a, jedna ćelija u koju se unose tekstualne vrijednosti „*TextField*“ zatim jedno dugme sa svojom funkcijom koja će se izvršiti nakon njegovog pritiska „*Button*“ što će u ovom slučaju kreirati jednostavan element za prikaz teksta „*Label*“ te glavni dio svega je element prema kojemu se određuje raspored za prikaz ostalih elemenata, u ovom slučaju je to „*VerticalLayout*“, dostupno je nekoliko različitih sličnih elemenata no uglavnom se koriste oni koji elemente redaju u horizontalu ili vertikalnu kao što je primjer sa slike, kroz primjer aplikacije će se vidjeti još neke dodatne mogućnosti ovog okvira. Još jedna napomena ukoliko želimo izrađivati te koristiti vlastite teme ili stilove potrebno ih je zasebno kompilirati putem alatne trake na kojoj se nakon instalacije Vaadin-a pojavi nova mogućnost sa nazivom „*Compile Vaadin Theme*“.

### 3.4. SCSS

CSS (engl. Cascading Style Sheets ) je jezik koji se koristi kako bi se izradila odnosno definirao izgled, stil iliti prezentacija web stranice, koristi se uz HTML elemente gdje se pojedine CSS klase mogu koristiti. CSS se koristi na praktički svim web stranicama no njegova glavna mana je ta što u svojoj sintaksi nema omogućeno korištenje varijabli, zato se mogu koristiti i druge opcije poput Sass-a (engl. Syntactically Awesome StyleSheets).

Sass je nadogradnja na CSS koja dodaje nekoliko novih mogućnosti poput prethodno navedenih varijabli zatim ugniježđenih pravila, uvoz drugih stilova te razne druge mogućnosti. Također je bitno napomenuti da je sama sintaksa potpuno kompatibilna sa običnim CSS-om tako da je moguće i običan CSS koristiti unutar Sass-a. Sass se sastoji od dvije različite sintakse, starija sintaksa koja se bazira na uvlakama te nema u sebi nikakve vitičaste zagrade ili točkice-zarez koja označava kraj linije i njeni dokumenti imaju ekstenziju .sass, te druga sintaksa koja je gotovo ista kao sintaksa od CSS-a uz naravno prethodno navedene mogućnosti te se naziva SCSS (engl. Sassy CSS) i njen nastavak je .scss. Bez obzira na to koja se sintaksa koristi i dalje je moguće uvesti stilove koji su napisani u drugoj sintaksi. Na slici 21 se može vidjeti kako izgleda sintaksa na primjeru koji je korišten za funkcionalnost aplikacije.

```

1 $v-app-loading-text: "Loading";
2 $v-background-color: #444d50;
3 $v-focus-color: #07a9ca;
4 $v-focus-style: 0 0 3px 2px $v-focus-color;
5 $v-bevel-depth: 40%;
6 $v-gradient: v-linear 12%;
7 $v-border-radius: 10px;
8 $v-font-family: Roboto, sans-serif;
9 $v-font-weight: 400;
10 $v-font-weight--header: 400;
11 $v-bevel: inset 0 1px 2px v-tint, inset 0 0 1px (v-tint 0.1);
12 $v-shadow: 0 0 0 3px rgba(0,0,0,0.32), 0 1px 0 3px rgba(255,255,255,0.14);
13 $v-textfield-bevel: inset 0 2px 2px v-shade;
14 $v-textfield-shadow: $v-shadow;
15 $v-unit-size: 40px;
16 $v-overlay-shadow: 0 0 0 3px (v-shade 8), 0 5px 10px (v-shade 4);
17 $v-component-group-spacing: 6px;
18
19 @import "../valo/valo.scss";
20
21 .tests-valo-light .valo-menu .valo-menu-title {
22   background: $v-app-background-color;
23   color: $v-selection-color;
24   text-shadow: none;
25   border-color: first-color(valo-border($color: $v-app-background-color, $strength: 0.5));
26 }
27 @mixin mytheme {
28   @include valo;
29 }
30 .searchLayoutBorders{
31   border: solid;
32   border-width: 2px;
33   border-radius: 6px;
34   padding: 10px;
35 }
36 .hlLabelStyle{
37   font-size: 32px;
38   line-height: normal;
39 }
40 .detailsImageMarginFix{
41   margin: 10px;
42 }

```

Slika 21. Primjer SCSS-a

### 3.5. Alati i tehnologije za izradu baze podataka

Sa obzirom da je fokus aplikacije koja je izrađena za svrhe ovoga rada bila TMDb te njihov API sama baza podataka i nije pretjerano potreban pošto sve potrebne informacije konstanto dohvaćamo putem API-a iz njihove baze podataka, neću detaljno prolaziti kroz samu uspostavu ovih alata. Za izradu lokalne baze koju sam koristio pri izradi aplikacije koristio sam MySQL, XAMPP te PhpMyAdmin.

MySQL je sustav za upravljanje relacijskim bazama podataka (engl. relational database management system - RDBMS) otvorenog koda te je trenutno u vlasništvu korporacije Oracle (Oracle Corporation, 2017). Iako je u svojoj prošlosti imao nekoliko problema danas je još

uvijek jedan od najčešće korištenih sustava u svojoj kategoriji. Ukoliko želimo korisničko sučelje imamo na izbor mnoge različite alate od kojih je najučestaliji MySQL Workbench koji je dostupan u besplatnoj verziji te uglavnom služi za modeliranje i strukturiranje željene baze podataka.

PhpMyAdmin je kao i prethodno navedeni Workbench također grafičko sučelje za MySQL, no njegova glavna svrha je omogućavanje administracije MySQL baze direktno iz web preglednika, ima mogućnost kreiranja, modificiranja ili brisanja baze, tablice ili redova, može izvršavati upite te kontrolirati korisnike te njihova prava.

Kako bismo mogli koristiti prethodno navedeni alat potrebno je i napraviti lokalni web poslužitelj kako bismo mogli testirati našu bazu a to je moguće uz pomoć XAMPP-a, te Apache servera koji je uz njega dostupan. Nakon što smo instalirali XAMPP možemo tretirati „localhost“ kao da je vanjski a ne lokalni.

Kako bismo mogli koristiti odnosno spajati se sa našom bazom iz aplikacije potrebno je prvo skinuti konektor za spajanje java sa MySQL-om te jar koji smo skinuli je potrebno dodati u projekt kao što je prikazano na slici 16 u prikazu Eclipse-a, osim dodavanja jar-a potrebno je i dodati zavisnost (engl. dependency) za MySQL u datoteku pom.xml kao što je prikazano na slici 22.

```
<dependency>
  <groupId>com.vaadin</groupId>
  <artifactId>vaadin-themes</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.42</version>
</dependency>
</dependencies>
```

Slika 2216. Dodavanje zavisnosti za MySQL

## 4. Aplikacija

Putem ove aplikacije prikazati ću kako se može izraditi jednostavna web aplikacija koja služi za prikazivanje detalja o filmovima te njihovom pretraživanju iliti filtriranju putem TMDb API-a. Ova aplikacija je potpuno izrađena u java programskom jeziku i u odabranom Vaadin okviru te svi primjeri koji će biti prikazani su napravljeni korištenjem navedenih tehnologija.

Kroz ovo poglavlje prikazati ću kako se može pozivati korišteni API odnosno dohvatiti JSON te koristiti podatke iz njega te ću kroz funkcionalnosti aplikacije pojasniti detaljnije kako funkcioniraju određene korištene API metode te naravno kako je sve u kodu složeno kako bi moglo funkcionirati. Unutar koda će se moći uočiti na više mjesta da je implementirano nizanje (engl. serializing) što je također poznato kao šifriranje objekata u tokove bajta, obrnuti proces se naziva dešifriranjem, jednom kada se objekt šifrira može ga se prosljeđivati od jednog virtualnog stroja do drugog ili ga je moguće spremati na disk kako bi čekao dešifriranje (Bloch Joshua, 2008). Također u kodu će se moći uočiti i par uzoraka dizajna, općenito oni se koriste kako bi kod mogao biti fleksibilnijim te da ga se lakše održava i da ga se može lakše iskoristiti ponovno. Uzorci nisu kompletan kod ali ih se može iskoristiti kao predložak za neki problem te ih se može implementirati na slične probleme bez obzira na domenu Joshi Rohit (2015).

### 4.1. Dohvaćanje JSON-a

Kao što sam već prethodno napomenuo, svi rezultati koje zaprimamo kao odgovore na upite koje smo slali putem API-a su zapisani u JSON formatu. Kao što se vidi na slici 23, napravio sam singleton klasu u kojoj mi se nalaze određene metode poput metode za dohvaćanje JSON objekata pod nazivom „*getJsonObjectFromURL*“, koja nam kao što se može i iz samoga naziva zaključiti vraća „*JsonObject*“ iz traženog URL-a. Singleton uzorak dizajna nam omogućuje da ne moramo konstantno kreirati nove objekte određene klase već imamo samo jedan kreirani objekt koji koristimo u svim slučajevima kada se određena klasa želi koristiti, U ovu klasu sam stavio metodu za dohvaćanje JSON-a pošto se ona poziva u gotovo svakom slučaju odnosno svim funkcionalnostima ove aplikacije te njenim postavljanjem u singleton omogućuje poprilično jednostavniji pristup te manje kreiranja novih objekata. Uz ovu metodu u klasi se također nalazi i mapa koja u sebi sadrži žanrove filmova, vrijednosti iz te mape se koriste tokom korištenja aplikacije na nekoliko mjesta odnosno svuda gdje se žele prikazati žanrovi tako da je mnogo jednostavnije i efikasnije dohvatiti vrijednosti za ovu mapu jednom prilikom inicijalizacije te spremati te vrijednosti kako se ne bi morale konstantno dohvaćati za svaki



željeni zahtjev gdje želimo određeni žanr prikazati, sadržaj klase „*JsonSingleton*“ se može vidjeti na slici 23.

```
1 package my.vaadin.app.client;
2
3+ import static my.vaadin.app.client.Constants.GENRES;
17
18 public class JsonSingleton {
19
20     private JsonSingleton() {
21     }
22
23     private static class SingletonHelper {
24         private static final JsonSingleton INSTANCE = new JsonSingleton();
25     }
26
27     public static JsonSingleton getInstance() {
28         return SingletonHelper.INSTANCE;
29     }
30
31     private final HashMap<Integer, String> genreMap = getGenres();
32
33     public JsonObject getJsonObjectFromURL(String linkURL) {
34         JsonObject jsonObject = null;
35         try {
36             URL myservice = new URL(linkURL);
37
38             InputStream openStream = myservice.openStream();
39             BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(openStream));
40             String line;
41             String str = "";
42
43             while ((line = bufferedReader.readLine()) != null) {
44                 str += line;
45             }
46             jsonObject = JsonUtil.parse(str);
47         } catch (IOException e) {
48             e.printStackTrace();
49         }
50         return jsonObject;
51     }
52
53     private HashMap<Integer, String> getGenres() {
54         JsonArray jsonObject = getJsonObjectFromURL(LIST_OF_GENRES).getArray(GENRES);
55         HashMap<Integer, String> genresMap = new HashMap<>();
56         for (int i = 0; i < jsonObject.length(); i++) {
57             genresMap.put((int) jsonObject.getObject(i).get("id").asNumber(),
58                 jsonObject.getObject(i).get("name").asString());
59         }
60         return genresMap;
61     }
62
63     public ArrayList<String> getGenreList(int... genreKeys) {
64         ArrayList<String> genreList = new ArrayList<String>();
65         for (int key : genreKeys) {
66             if (genreMap.get(key) != null) {
67                 genreList.add(genreMap.get(key).toString());
68             } else {
69                 genreList.add("");
70             }
71         }
72         return genreList;
73     }
}
```

Slika 23. Sadržaj klase "JsonSingleton"

Kao što se može vidjeti na samome početku klase se definira da je se radi o singleton klasi gdje smo prvo napravili privatni konstruktor kako bismo onemogućili daljnje kreiranje objekata

te smo napravili statičnu te konačnu instancu objekta koju dohvaćamo uz pomoć metode „*getInstance*“ što će se moći uočiti u daljnjim primjerima. Slijedeća je metoda koja nam služi za dohvaćanje JSON objekata, ona prima jedan parametar koji nam se odnosi na punu poveznicu koju želimo koristiti, primjeri za neke od korištenih primjera se mogu naći unutar drugog poglavlja ovoga rada. Sa zadane poveznice dohvaćamo vrijednosti odnosno vraćeni sadržaj kojega čitamo te ga zapisujemo u niz znakova kojega na kraju transformiramo iliti raščlanjujemo u objekt tipa „*JsonObject*“ kojega vraćamo kao rezultat pretrage.

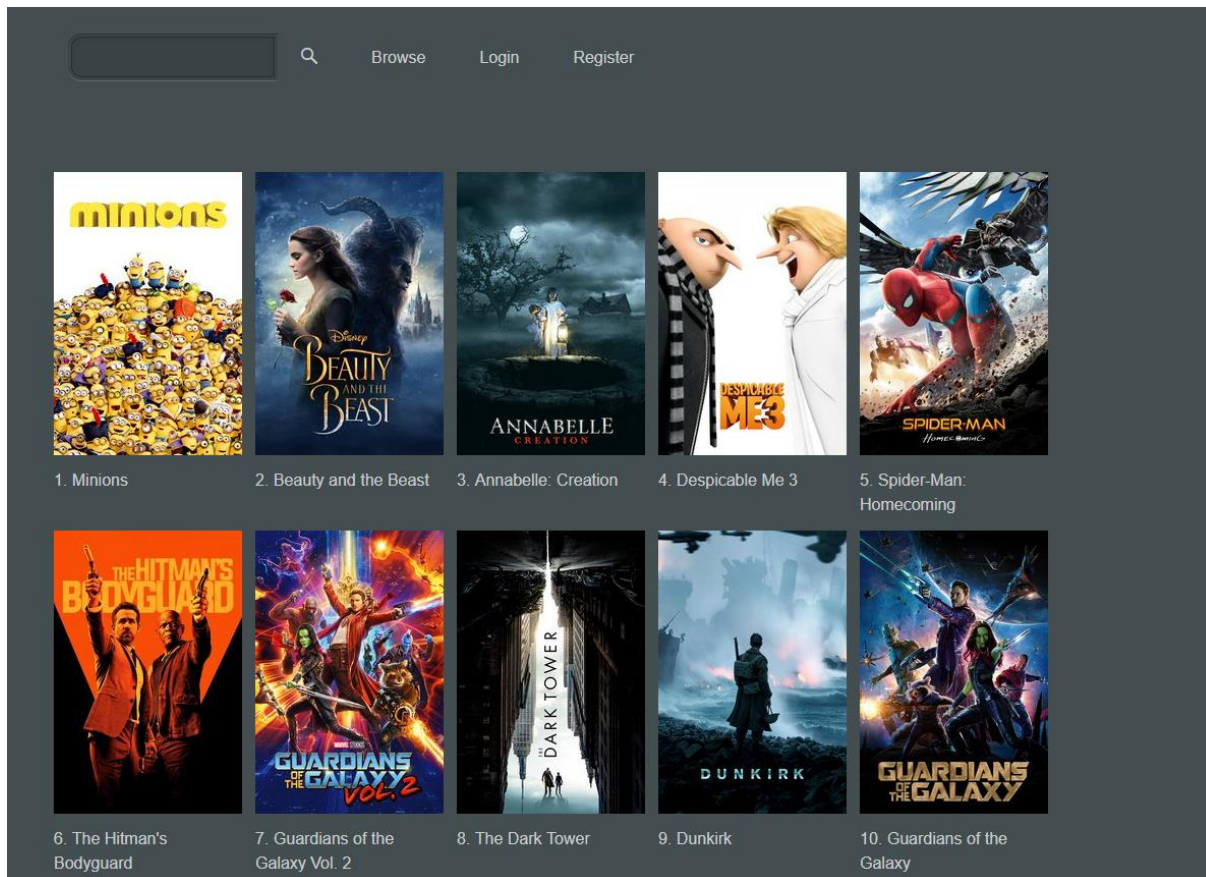
Slijedeća metoda „*getGenres*“ služi za dohvaćanje svih dostupnih žanrova te popunjavanje mape sa žanrovima koja se koristi dalje u aplikaciji, ova metoda se koristi samo jednom te unutar same klase pa je stoga i njena vidljivost namještena na privatno. Odmah u prvoj linije koristimo prethodno pojašnjenu metodu kako bismo dobili objekt u kojemu nam se nalazi željeni rezultat, poveznica koja je u ovom slučaju bila korištena je spremljena kao konstanta u zasebnoj klasi kako bi kod bio čitljiviji i pregledniji, korištena poveznica je bila prethodno prikazana sa svojim djelomičnim rezultatom u poglavlju 2 na slici 12. Dodatno kao što se može vidjeti nakon što smo dohvatili JSON objekt ulazimo „dublje“ u njega sa dodatnom metodom „*getArray*“ koja ima parametar koji joj služi kao ključ prema kojemu raspoznaje koji dio JSON objekta treba dohvatiti, u ovom slučaju je to polje žanrova koje je također vidljivo na prethodno spomenutoj slici. Zatim u metodu jednostavnim prolaskom kroz polje spremamo elemente u mapu sa njivom ključem te nazivom

Metoda „*getGenreList*“ vraća listu u kojoj se nalaze nazivi žanrova, s obzirom da većina metoda koje koristimo putem API-a u svojem povratnom JSON-u sadrže samo zapisane ključeve od žanrova koji korisniku nisu od prevelike koristi, putem ove metode možemo za dobivene identifikacijske ključeve dohvatiti njihove pripadajuće nazive. Ova metoda kao parametar prima polje odnosno niz varijabli tipa cjelobrojnih brojeva te zatim prolazi kroz primljeno polje te za svaki ključ dohvaća vrijednost iz mape žanrova koju smo prethodno dohvatili. U ovom dijelu treba pripaziti na mogućnost da nisu svi identifikacijski ključevi od žanrova validni, na primjer žanr za strane filmove je prethodno bio prisutan u dostupnim žanrovima no međutim on je bio uklonjen pošto se taj žanr smatrao subjektivnim, nešto što je strano za jednu osobu možda nije strano za nekoga drugoga, no iako je taj žanr uklonjen iz njihove baze još uvijek nije uklonjen njegova ključ sa svih filmova na kojima se prethodno nalazio tako da nam ovaj žanr može stvarati probleme te rušiti aplikaciju.

## 4.2. Početna stranica

Na početnoj stranici ove aplikacije dohvaćamo te prikazujemo tablicu sa trenutno 20 najpopularnijih filmova na TMDb-u, na slici 24 se nalazi slika početne stranice ove aplikacije.

Na vrhu slike se može uočiti i opcija za pretraživanje filmove te ukoliko je ona prazna



Slika 24.. Izgled početne stranice

te pritisnemo na pretraživanje ponovno će nas aplikacija dovesti do trenutno prikazane početne stranice, ukoliko želimo vidjeti detalje o nekom od prikazanih filmova u tablici možemo doći do njih jednostavnim pritiskom miša na jednu od slika.

Ovu stranicu prvi puta možemo dobiti prilikom otvaranja same aplikacije u kojem slučaju se kao što sam ranije prikazao poziva klasa „MyUI“ iz koje se dalje vrši navigacija kroz aplikaciju, u ovom slučaju sam dodao objekt tipa „BodyLayout“ što je klasa koju sam napravio kako bih si odvojio zaglavlje stranice od njenog tijela te kako ne bih morao konstantno uništavati objekte te tako reći precrtavati preko elemenata na stranici. Ovim načinom sam jednom deklarirao ovaj objekt te ukoliko je potrebno i bilo kojem slučaju nešto zapisati odnosno prikazati na stranici jednostavno je potrebno dodati komponente unutar tog objekta. Na slici 25

se može vidjeti dio klase „*BodyLayout*“ koji je potreban kako bi se prikazala početna stranica aplikacije.

Kao što se može vidjeti u samom konstruktoru klase se poziva metoda za dohvaćanje početne stranice „*getStartBody*“ u kojoj se prvo počisti tijelo stranice za svaki slučaj ukoliko

```
1 package my.vaadin.app.client;
2
3 import static my.vaadin.app.client.Constants.*;
15
16 public class BodyLayout extends FormLayout {
17
18     private static final long serialVersionUID = 1L;
19
20     private MyUI myUI;
21
22     public BodyLayout(MyUI myUI) {
23         this.myUI = myUI;
24         getStartBody();
25     }
26
27
28     private void getStartBody() {
29         removeAllComponents();
30         JsonObject jsonObject = JsonSingleton.getInstance().getJsonObjectFromURL(POPULAR_MOVIES);
31
32         if (jsonObject != null) {
33             addComponent(getPopularMoviesPosterGrid(jsonObject.getArray(RESULTS)));
34
35         } else {
36             Notification.show("Error fetching data..", Type.ERROR_MESSAGE);
37         }
38     }
39 }
40
41 private GridLayout getPopularMoviesPosterGrid(JsonArray jsResults) {
42     GridLayout popularMoviesGrid = new GridLayout(5, 8);
43
44     for (int rowCounter = 0, i = 0; rowCounter < popularMoviesGrid.getRows(); rowCounter += 2) {
45         for (int columnCounter = 0; columnCounter < popularMoviesGrid.getColumns(); columnCounter++, i++) {
46
47             Label movieTitle = new Label((i + 1) + ". " + jsResults.getObject(i).getString(TITLE));
48             movieTitle.setWidth(185, Unit.PIXELS);
49
50             String poster = jsResults.getObject(i).getString(POSTER_PATH);
51             String title = jsResults.getObject(i).getString(TITLE);
52             String imageID = String.valueOf((int) jsResults.getObject(i).getNumber("id"));
53
54             popularMoviesGrid.addComponent(
55                 CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_185, false, this), columnCounter,
56                 rowCounter);
57             popularMoviesGrid.addComponent(movieTitle, columnCounter, rowCounter + 1);
58         }
59     }
60     popularMoviesGrid.setSpacing(true);
61     return popularMoviesGrid;
62 }
63
64 public void searchForMovies(boolean scrollToTop, String searchString, int nape) {
```

Slika 25. Kod za prikaz početne stranice

se nešto na njoj već nalazi te se nakon toga dohvaća JSON objekt putem već prethodno opisane metode uz pomoć URL-a za dohvaćanje popularnih filmova. Nakon što smo dohvatili objekt koji nam sadrži relevantne informacije za prikaz, koristimo metodu „*getPopularMoviesPosterGrid*“ koja prima parametar tipa JSON polja te nam vraća element tipa „*GridLayout*“ koji omogućuje spremanje odnosno poslagivanje elemenata u obliku tablice. U metodi smo prvo definirali element za raspoređivanje te mu dodijelili vrijednosti koje označavaju koliko će se stupaca ili redova prikazati, u ovom slučaju je 5 stupaca te 8 u redova, cilj nam je prikazati sve skupa 20 filmova te njihove nazive ispod njih stoga smo morali dodati više elemenata u kolone nego li u redove pa stoga smo morali dodati i dodatne redove. Ugniježdenim petljama prolazimo kroz dobiveno polje te dodajemo elemente u tablicu putem

njihovih pripadajućih brojeva. Podaci koji su nam u ovom slučaju potrebni su putanja do željene slike, naziv filma te identifikacijski broj filma koji će nam služiti kako bismo mogli jednostavnim klikom miša doći do dodatnih detalja o filmu. Kao što se vidi za naziv filma koristimo običan element za prikaz niza znakova te za prikaz slike dohvaćamo element iz druge klase pod nazivom „*CustomItems*“ te koristimo metodu „*getImage*“ koja se može vidjeti na slici 26.

Sa obzirom da je ova metoda poprilično generalna te nam služi za dohvaćanje elementa u kojemu se nalazi slika filma nju smo izdvojili u zasebnu klasu gdje se nalaze i druge slične

```
public static Image getImage(String poster, String title, String imageID, String imageSize, boolean isFromDetails, BodyLayout bodyLayout){
    Image img = new Image();

    if (poster==null||poster.isEmpty()) {
        img.setSource(new ThemeResource("images/Default_Image.jpg"));
    } else {
        img.setSource(new ExternalResource(imageSize + poster));
    }
    setImageSize(img, imageSize);

    img.setId(imageID);
    img.setDescription(title);

    if (isFromDetails) {
        BrowserWindowOpener imgOpener = new BrowserWindowOpener(POSTER_IMAGE_ORIGINAL + poster);
        imgOpener.setWindowName("_blank");
        imgOpener.extend(img);
    } else {
        img.addClickListener(e -> {
            bodyLayout.showMovieDetails(Integer.parseInt(img.getId()));
        });
    }
}
```

Slika 26. Metoda za dohvaćanje slike

metode koje se koriste na nekoliko različitih mjesta, te je statična kako bi joj se moglo jednostavnije pristupiti. Unutar metode prvo provjeravamo dali nam je proslijeđena putanja do slike validna odnosno dali ta slika zapravo uopće postoji, ukoliko ne postoji onda dodjeljujemo običnu sliku koja nam označava da slika nije dostupna, inače postavljamo izvor slike na eksterni te dodajemo putanju odnosno poveznicu do željene slike. Nadalje provjeravamo ukoliko je tražena slika sa detalja o filmu te ukoliko nije dodajemo joj mogućnost da se pritiskom na nju otvaraju detalji putem metode „*showMovieDetails*“ koja će biti u nastavku detaljnije objašnjena, te ukoliko se slika nalazi na prikazu detalja o filmu, nećemo je ponovno povezivati sa već prikazanim detaljima već ćemo odabirom slike prikazati željenu sliku u njenoj punoj veličini.

### 4.3. Pretraživanje filmova

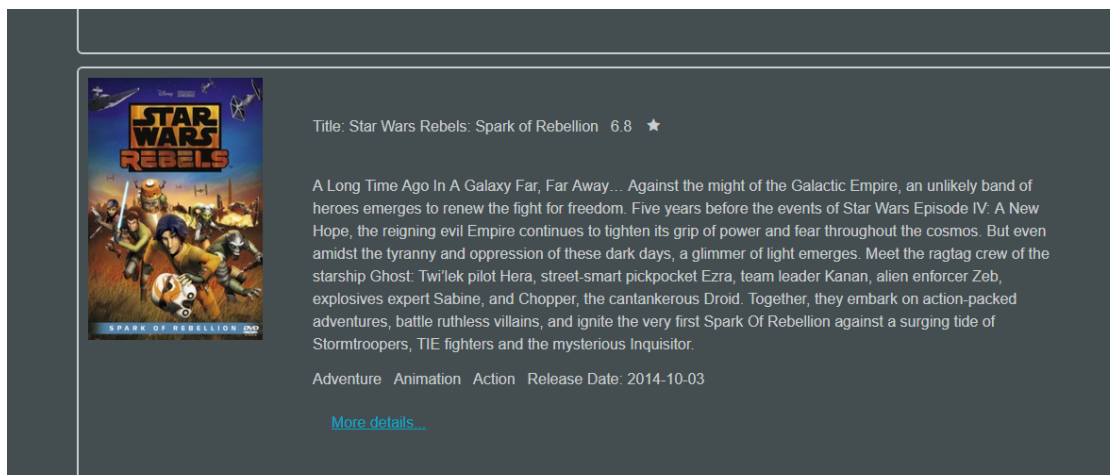
Pretraživanje filmova je omogućeno putem zaglavlja gdje možemo upisati dio naziva filma te ga jednostavnim pritiskom na ikonicu za pretraživanje iliti pritiskom tipke „enter“ možemo pretražiti TMDb sa unesenim upitom, na slici 27 se nalazi metoda „*getSearchLayout*“ koja nam dohvaća tekstualno polje te jedno dugme sa ikonom. Ova metoda je poprilično jednostavna te jedino što bih izdvojio je način na koji postavlja tekstualno polje da reagira na pritisak tipke te ukoliko želimo omogućiti pretraživanje upita sa razmakom potrebno je taj razmak zapisati kodiranog kako bi se moglo ispravno pretraživati.

```
private Component getSearchLayout() {
    CssLayout searchLayout=new CssLayout();

    btnSearch.setStyleName(ValoTheme.BUTTON_BORDERLESS);
    btnSearch.setIcon(VaadinIcons.SEARCH);
    btnSearch.setClickShortcut(KeyCode.ENTER);
    btnSearch.addClickListener(e-> {
        myUI.getBodyLayout().searchForMovies(false, txtSearch.getValue().replaceAll(" ", "%20"), 1);
    });
    searchLayout.setStyleName(ValoTheme.LAYOUT_COMPONENT_GROUP);
    searchLayout.addComponents(txtSearch,btnSearch);
    return searchLayout;
}
```

Slika 27. Dohvaćanje elemenata za pretraživanje

Nakon uspješne pretrage prikazujemo popis pronađenih filmova te u kojima piše njihov naziv, prosječna ocjena, kratak opis filma, lista žanrova, datum premijere te opcija za otvaranjem više detalja o filmu koje je također dostupno klikom na sliku filma sa lijeve strane, što se može vidjeti na slici 28. Rezultat pretrage iz API-a dohvaća samo 20 rezultata tako da moramo također imati i nekakvo obilježavanje za stranice kao što se može vidjeti i na slici što ću detaljnije objasniti u nastavku.



Slika 28. Izgled rezultata pretrage



Kao što se vidjelo na kodu sa slike 27, metoda koju pozivamo kako bismo uspješno pretražili te dobili prikaz dobivenih rezultata se zove „*searchForMovies*“ te prima parametre koji označavaju dali želimo da se prikaz ekrana vrati na vrh, tekstualna vrijednost odnosno upit koji želimo izvršiti te redni broj stranice koju želimo prikazati, na slici 30 je vidljiva spomenuta metoda.

```
public void searchForMovies(boolean scrollToTop, String searchString, int page) {
    if (searchString == null || searchString.isEmpty()) {
        getStartBody();
        Page.getCurrent().pushState(NAVIGATION_START_PAGE);
    } else {
        String url = MOVIE_SEARCH + QUERY + searchString + PAGEID + page;
        JsonObject jsonObject = JsonSingleton.getInstance().getJsonObjectFromURL(url);
        if (jsonObject != null && jsonObject.getNumber(TOTAL_RESULTS) > 0) {
            removeAllComponents();
            addComponent(new SearchLayout(this, jsonObject, false, searchString, null));
            if (scrollToTop) {
                UI.getCurrent().scrollIntoView(myUI.getHeaderLayout());
            }
        } else {
            Notification.show("Data not found.", Notification.Type.ERROR_MESSAGE);
        }
    }
}
```

Slika 29. Metoda za pretraživanje filmova

U početku ove metode prvo se provjerava vrijednost teksta kojega šaljemo kao upit te ukoliko je on nepostojeći, prebacujemo korisnika na početnu stranicu, inače napravimo URL gdje upišemo uneseni upit te željenu stranicu koju želimo prikazati. Na temelju sagrađenog URL-a dohvaćamo JSON objekt kojemu prvo provjeravamo broj rezultata te ukoliko rezultata nema ispisujemo poruka na ekran da podaci nisu pronađeni, inače očistimo tijelo stranice te dodajemo novi objekt klase „*SearchLayout*“ koji će nam prikazati rezultate pretrage, na kraju još imamo i prebacivanje na vrh stranice ukoliko je to bilo zatraženu putem parametra metode što se koristi uglavnom prilikom korištenja stranica. Na slici 30 su prikazane metode koje služe za dohvaćanje vrijednosti iz dobivenog JSON objekta te njihovo strukturiranje i spremanje kako bi se omogućio prikaz sa slike 28. U konstruktoru prethodno navedene klase se samo postavljaju određene vrijednosti te razrješava pitanje o stranicama koje ću kasnije opisati te se poziva prva metoda sa slike „*setSearchLayout*“. Navedena metoda prolazi kroz polje JSON objekata te dohvaća vrijednosti koje su nam potrebne kako bismo prikazali sliku kao i na prošlom primjeru.

```

private void setSearchLayout(JsonArray jsArray) {
    for (int i = 0; i < jsArray.length(); i++) {
        JsonObject jsonObject = jsArray.get(i);
        HorizontalLayout imageDetailsLayout = new HorizontalLayout();

        String poster = jsonObject.get(POSTER_PATH).equals(Json.createNull()) ? ""
            : jsonObject.getString(POSTER_PATH);
        String title = jsonObject.get(TITLE) == null || jsonObject.get(TITLE).equals(Json.createNull()) ? ""
            : jsonObject.getString(TITLE);
        String imageID = String.valueOf((int) jsonObject.getNumber("id"));

        imageDetailsLayout.addComponents(
            CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_185, false, bodyLayout),
            getMovieDetails(jsonObject));
        imageDetailsLayout.addStyleName("searchLayoutBorders");
        addComponent(imageDetailsLayout);
    }
}

private VerticalLayout getMovieDetails(JsonObject jsonObject) {
    VerticalLayout movieDetailsLayout = new VerticalLayout();

    HorizontalLayout detailsHeaderLayout = new HorizontalLayout();
    Label title = new Label("Title: " + jsonObject.getString(TITLE));
    HorizontalLayout vote = CustomItems.avgVoteStarLayout(String.valueOf(jsonObject.getNumber(VOTE_AVERAGE)));
    detailsHeaderLayout.addComponents(title, vote);

    String desc=jsonObject.get(DESCRIPTION).equals(Json.createNull())?"Missing description":jsonObject.getString(DESCRIPTION);
    Label description = CustomItems.descriptionLabel(desc, isRecommendation);

    HorizontalLayout detailsFooterLayout = new HorizontalLayout();

    HorizontalLayout genres = CustomItems.genresLayout(jsonObject.getArray(GENRE_IDS), false);

    String rDate=jsonObject.get(RELEASE_DATE).equals(Json.createNull())?"Missing release date":jsonObject.getString(RELEASE_DATE);
    Label releaseDate = new Label("Release Date: " + rDate);
    detailsFooterLayout.addComponents(genres, releaseDate);

    Button moreDetails = new Button("More details...");
    moreDetails.setStyleName(ValoTheme.BUTTON_LINK);
    moreDetails.addClickListener(e->{
        bodyLayout.showMovieDetails((int) jsonObject.getNumber("id"));
    });
}

```

Slika 30. Metode za prikaz rezultata pretrage

Nakon što smo dohvatili sliku, slijedeći su detalji o filmu koje dohvaćamo metodom „getMovieDetails“ kojoj kao parametar prosljeđujemo JSON objekt što je prvi zapis u polju objekata iz prethodne metode koje smo dobili pretraživanjem. Plan odnosno raspored elemenata na ekranu namještamo sa vertikalnim te horizontalnim rasporedima kao što se vidi u metodi, zapisi koji su vidljivi su jednostavno prikazani kao osnovni tekstualni objekti jedino na što trebamo paziti jest dali je određena vrijednost koju želimo dostupna odnosno dali je možemo dohvatiti, stoga se u metodi nalazi nekoliko provjera dali su vrijednosti validne. Jedina novost u ovoj metodi je metoda koja nam dohvaća raspored sa žanrovima „genresLayout“ koja je vidljiva na slici 31.



```

public static HorizontalLayout genresLayout(JsonArray genreIDs, boolean isFromDetails) {
    HorizontalLayout genreLayout = new HorizontalLayout();
    if (genreIDs == null || genreIDs.length() == 0) {
        genreLayout.addComponent(new Label("No genres found!"));
        return genreLayout;
    }
    int[] genreIds = new int[genreIDs.length()];
    ArrayList<String> genresList = new ArrayList<>();
    for (int i = 0; i < genreIDs.length(); i++) {
        if (isFromDetails) {
            genresList.add(genreIDs.getObject(i).getString("name"));
        } else {
            genreIds[i] = (int) genreIDs.getNumber(i);
        }
    }
    if (!isFromDetails) {
        genresList = JsonSingleton.getInstance().getGenreList(genreIds);
    }
    for (String genre : genresList) {
        Label l = new Label(genre);
        genreLayout.addComponent(l);
    }
    return genreLayout;
}

```

Slika 31. Dohvaćanje žanrova

U ovoj metodi prvo provjeravamo uopće postoje zapisani žanrovi za određeni film te ukoliko ne postoje vraćamo samo poruku da žanrovi nisu pronađeni. Ukoliko žanrovi postoje, deklariramo jednu listu gdje ćemo zapisivati nazive žanrova te jedno polje u koje ćemo spremati identifikacijske brojeve žanrova ukoliko u rezultatu nemamo zapisane nazive. Zatim prolazimo kroz dobiveno polje JSON objekata te provjeravamo dali je ova metoda pozvana sa detalja o filmovima, ukoliko je onda znamo da imamo zapisano u žanrovima i njihove nazive a ne samo brojeve tako da možemo odmah popuniti listu sa nazivima, ukoliko smo pozvali metodu iz nekog drugog mjesta poput pretraživanja onda imamo samo brojeve zapisane u JSON objektu tako da ih zapisujemo u prethodno navedeno polje koje zatim prosljeđujemo u metodu koju smo opisali uz „*JsonSingleton*“ klasu koja nam vraća lista sa nazivima žanrova.

## 4.4. Obilježavanje stranica

S obzirom da nam TMDb API za naše upite iliti pretrage vraća samo do 20 rezultate potrebno je bilo i napraviti pripadajuće obilježavanje stranica, rezultati koje primamo u JSON formatu imaju u sebi i varijable u kojima su zapisane vrijednosti odnosno lokacija na kojoj se stranici trenutno nalazimo iliti koja je stranica bila dohvaćen, ukupan broj rezultata te ukupan broj stranica što nam sve mnogo olakšava izradu iliti obilježavanje stranica. U ovoj aplikaciji pretraživanje radi na način da uvijek prikazuje maksimum 5 dostupnih stranica s time da uvijek obilježi broj stranice na kojoj se korisnik trenutno nalazi te ukoliko lista malo više prikazuje

dvije prethodne stranice te dvije slijedeće, ukoliko se korisnik nalazi na prvoj stranici također ima prikazane opcije za odabir zadnje stranice te odabir iduće stranice kako ne bi morao pritiskati na brojeve. Ukoliko se korisnik nalazi na drugoj stranici prikazati će se i opcija za izbor prethodne stranice te ukoliko broj prve stranice više nije vidljiv također će se pojaviti i opcija za izbor prve stranice, ova logika također vrijedi i kod nestajanja te kod stranica pri kraju liste, dio koda koji omogućuje ovu logiku se nalazi na slici 32.

```
public SearchLayout(BodyLayout bodyLayout, JsonObject jsonObject, boolean isRecommendation, String searchQuery,
    String genreIds) {
    this.bodyLayout = bodyLayout;
    this.isRecommendation = isRecommendation;
    this.genreIds = genreIds;
    this.searchQuery = searchQuery;
    setSearchLayout(jsonObject.getJSONArray(RESULTS));

    int currentPage = (int) jsonObject.getNumber(CURRENT_PAGE);
    if (!isRecommendation && jsonObject.getNumber(TOTAL_PAGES) > 1) {
        resolvePagingLayout(currentPage, (int) jsonObject.getNumber(TOTAL_PAGES));
    }
    setwidth("1050px");
    if (genreIds == null) {
        Page.getCurrent()
            .pushState(NAVIGATION_SEARCH_PAGE + searchQuery + NAVIGATION_RESULT_PAGE_NUMBER + currentPage);
    }
}

private void resolvePagingLayout(int currentPage, int totalPages) {
    if (currentPage <= 3) {
        int endCounter = 6;
        if (totalPages < 5) {
            endCounter = totalPages + 1;
        }
        setPagingLayout(currentPage, totalPages, 1, endCounter);
    } else if (currentPage + 2 >= totalPages) {
        setPagingLayout(currentPage, totalPages, totalPages - 5, totalPages + 1);
    } else {
        setPagingLayout(currentPage, totalPages, currentPage - 2, currentPage + 3);
    }
}
```

Slika 32. Kod za određivanje stranica

Na slici nam je prvo prikazan konstruktor prethodno opisane klase za pretraživanje gdje se također određuje na kojoj se stranici korisnik nalazi kako bi se moglo prolaziti kroz stranice na pretraživanju te se također provjerava dali se nalazi na detaljima odnosno dali se radi o pretraživanju preporučenih filmova sa detalja gdje prikazujemo samo top 20 preporuka pa stoga nema potrebe implementirati opcije sa stranicama na toj funkcionalnosti. Metoda „*resolvePagingLayout*“ prima parametre za trenutnu stranicu na kojoj se nalazi te ukupan broj stranica koje se nalaze u dobivenom rezultatu. Ova metoda nam služi kako bismo razaznali koliko je točno stranica dostupno odnosno koje sve stranice treba prikazati za svojim varijablama koje označavaju od koje stranice te kojega broja se trebaju prikazivati stranice što se prikazuju putem metode „*setPagingLayout*“ koja je prikazana na slici 33.

```

private void setPagingLayout(int currentPage, int lastPage, int startCounter, int endCounter) {
    HorizontalLayout pagesLayout = new HorizontalLayout();
    if (currentPage != 1) {
        if (lastPage > 5 && currentPage > 3) {
            Button first = CustomItems.createBorderLessButton("First", false);
            first.addClickListener(e -> {
                openPage(1);
            });
            pagesLayout.addComponent(first);
        }

        Button previous = CustomItems.createBorderLessButton("Previous", false);
        previous.addClickListener(e -> {
            openPage(currentPage - 1);
        });
        pagesLayout.addComponent(previous);
    }
    for (int i = startCounter; i < endCounter; i++) {
        Button page = CustomItems.createBorderLessButton(i + "", false);
        final int pageNum = i;
        if (currentPage == i) {
            page.setStyleName(ValoTheme.BUTTON_BORDERLESS_COLORED);
        }

        page.addClickListener(e -> {
            openPage(pageNum);
        });
        pagesLayout.addComponent(page);
    }
    if (currentPage != lastPage) {
        Button next = CustomItems.createBorderLessButton("Next", false);

        next.addClickListener(e -> {
            openPage(currentPage + 1);
        });
        pagesLayout.addComponent(next);

        if (currentPage + 2 < lastPage) {
            Button last = CustomItems.createBorderLessButton("Last", false);
            last.addClickListener(e -> {
                openPage(lastPage);
            });
            pagesLayout.addComponent(last);
        }
    }

    addComponent(pagesLayout);
}

private void openPage(int pageNumber) {
    if (genreIds != null) {
        bodyLayout.browseGenres(true, genreIds, pageNumber);
    } else {
        bodyLayout.searchForMovies(true, searchQuery, pageNumber);
    }
}

```

Slika 33. Kod za prikaz stranica

Ova metoda služi kao što sam ranije spomenuo za prikaz dostupnih stranica u objektu kojega smo dobili pretraživanjem, u prvom koraku provjeravamo dali se korisnik nalazi na nekoj prvoj stranici te ukoliko se ne nalazi znamo da sigurno trebamo prikazati opciju za prethodnu stranicu te provjeravamo dali je dostupno više od 5 stranica te dali se korisnik nalazi na stranici više od 3 ukoliko je to istina onda možemo prikazati i opciju za prvu stranicu. Ista

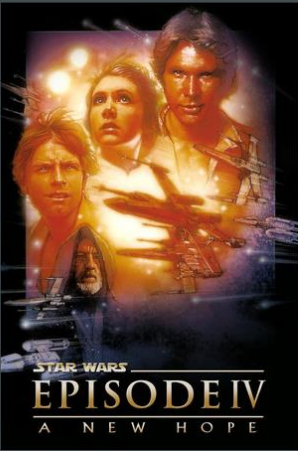
logika se nalazi i za opcije koje šalju na zadnju iliti iduću stranicu samo u obrnutom stilu, bez obzira gdje se korisnik nalazi imamo brojače za početak te kraj stranica koje je potrebno prikazati te prolaskom kroz petlju prikazujemo željene stranice te kada dođemo do trenutno prikazane stranice označimo kako bi se mogla prepoznati.

Sve mogućnosti koje možemo odabrati pozivaju metodu „*openPage*“, kao što se može naslutiti iz samoga naziva ova metoda otvara stranicu koja je odabrana, naravno kao parametar zaprima broj stranice koji je potrebno otvoriti te također koristi i globalne varijable u kojima su zapisane vrijednosti koje su potrebne kako bi se prikazali opet isti rezultati samo sa drugom stranicom. Unutar metode prvo provjeravamo dali se radi o funkcionalnosti za otkrivanje iliti o nekoj drugoj funkcionalnosti te se pozivaju odgovarajuće metode.

## 4.5. Prikaz detalja

Prikaz detalja o filmu možemo dobiti kao što sam ranije spomenuo putem opcije za prikaz više detalja o filmu na funkcionalnostima gdje su pronađeni filmovi ili putem klika na određenu sliku filma. Na slici 34 su prikazane detalji koji su dostupni putem ove funkcionalnosti.

### Star Wars



[IMDB](#) 8.0 ★  
Princess Leia is captured and held hostage by the evil Imperial forces in their effort to take over the galactic Empire. Venturesome Luke Skywalker and dashing captain Han Solo team together with the loveable robot duo R2-D2 and C-3PO to rescue the beautiful princess and restore peace and justice in the Empire.  
Adventure Action Science Fiction

[Details](#) [Trailers](#) [Images](#) [Cast](#) [Crew](#) [Recommendations](#)

**Homepage:** <http://www.starwars.com/films/star-wars-episode-iv-a-new-hope>

**Status:** Released

**Release date:** 1977-05-25

**Runtime:** 121.0

**Budget:** 11.000.000,00

**Revenue:** 775.398.007,00

**Collection:** Star Wars Collection

**Production companies:** Lucasfilm, Twentieth Century Fox Film Corporation

Slika 34. Prikaz detalja o filmu

Na slici se mogu prvo vidjeti jednaki detalji koji su dostupni kao i sa pretraživanja poput opisa filma, ocjena, žanrova i sličnih informacija no uz te podatke se također nalazi i poveznica koja otvara i web stranicu od TMDb-a ukoliko korisnik želi dodatne informacije sa drugih izvora. Također se unutar odabrane kartice mogu vidjeti i detalji koji nisu vidljivi putem pretraživanja.

```
private JSONObject jsObject;
private BodyLayout bodyLayout;
private int movieID;

private boolean appendVideos;
private boolean appendSimilar;
private boolean appendCredits;
private boolean appendImages;

public static class DetailsLayoutBuilder {
    private BodyLayout bodyLayout;
    private int movieID;

    private boolean appendVideos;
    private boolean appendSimilar;
    private boolean appendCredits;
    private boolean appendImages;

    public DetailsLayoutBuilder(BodyLayout bodyLayout, int movieID) {
        this.bodyLayout = bodyLayout;
        this.movieID = movieID;
    }

    public DetailsLayoutBuilder appendVideos(boolean appendVideos) {
        this.appendVideos = appendVideos;
        return this;
    }

    public DetailsLayoutBuilder appendSimilar(boolean appendSimilar) {
        this.appendSimilar = appendSimilar;
        return this;
    }

    public DetailsLayoutBuilder appendCredits(boolean appendCredits) {
        this.appendCredits = appendCredits;
        return this;
    }

    public DetailsLayoutBuilder appendImages(boolean appendImages) {
        this.appendImages = appendImages;
        return this;
    }

    public DetailsLayout build() {
        return new DetailsLayout(this);
    }
}

private DetailsLayout(DetailsLayoutBuilder builder) {
    this.bodyLayout = builder.bodyLayout;
    this.movieID = builder.movieID;

    this.appendVideos = builder.appendVideos;
    this.appendSimilar = builder.appendSimilar;
    this.appendCredits = builder.appendCredits;
    this.appendImages = builder.appendImages;

    setMargin(false);
    setSpacing(false);
    addComponent(getMovieDetails());
}
```

Slika 17. Builder uzorak u klasi za prikaz detalja

Na slici 36. se nalazi dio klase koju sam koristio za prikaz detalja o filmovima gdje možemo vidjeti da sam koristio takozvani „*Builder*“ uzorak dizajna koji omogućuje mnogo lakše proširivanje klase koje smatram da je u ovom slučaju poprilično korisno pošto nam API omogućava dodavanje ili spajanje metoda sa njihovom metodom „*append\_to\_response*“ koju sam već prethodno prikazao i opisao kroz poglavlje 2, ukoliko se u budućnosti odlučimo dodavati još neke upite u zahtjev te ih budemo htjeli prikazivati na detaljima ne moramo se nepotrebno mučiti sa proširivanjem ili kreiranjem novih konstruktora te doprinosti nepreglednosti koda već možemo jednostavno dodati varijable unutar ovog uzorka. Na slici 36 se nalazi dio koda koji je zaslužan za prikaz detalja koji su prikazani na slici 34.

```
private Component getMovieDetails() {
    JSONObject = JsonSingleton.getInstance()
        .getJSONObjectFromURL(MOVIE_DETAILS.replace(REPLACE_STRING, "" + movieID) + appendToResponse());

    VerticalLayout leftLayout = new VerticalLayout();
    VerticalLayout middleLayout = new VerticalLayout();
    HorizontalLayout movieDetailsLayout = new HorizontalLayout();
    HorizontalLayout movieDetailsHeaderLayout = new HorizontalLayout();

    Label movieName = new Label(JSONObject.getString(TITLE));
    movieName.addStyleName(ValoTheme.LABEL_H2);

    String poster = JSONObject.get(POSTER_PATH).equals(Json.createNull()) ? null : JSONObject.getString(POSTER_PATH);
    String title = JSONObject.getString(TITLE);
    String imageID = String.valueOf((int) JSONObject.getNumber("id"));

    Image movieImage = CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_300, true, null);

    Button imdbLink = new Button("IMDB");
    imdbLink.setStyleName(ValoTheme.BUTTON_LINK);
    BrowserWindowOpener opener = new BrowserWindowOpener(new ExternalResource(IMDB_LINK + JSONObject.getString(IMDB_ID)));
    opener.setFeatures("");
    opener.extend(imdbLink);

    HorizontalLayout avgVote = CustomItems.avgVoteStarLayout(String.valueOf(JSONObject.getNumber(VOTE_AVERAGE)));
    avgVote.setDescription(String.valueOf(JSONObject.getNumber(VOTE_COUNT)));

    Label description = CustomItems.descriptionLabel(JSONObject.getString(DESCRIPTION), true);
    HorizontalLayout genres = CustomItems.genresLayout(JSONObject.getArray(GENRES), true);

    movieDetailsHeaderLayout.setDefaultComponentAlignment(Alignment.MIDDLE_CENTER);
    movieDetailsHeaderLayout.addComponents(imdbLink, avgVote);

    middleLayout.addComponents(movieDetailsHeaderLayout, description, genres);

    movieDetailsLayout.addComponents(movieImage, middleLayout);

    leftLayout.addComponents(movieName, movieDetailsLayout, getDetailsTabSheet());

    HorizontalLayout hl = new HorizontalLayout();
    hl.addComponent(getDetailsTabSheet());

    Page.getCurrent().pushState(NAVIGATION_DETAILS_PAGE + imageID);

    return leftLayout;
}

private TabSheet getDetailsTabSheet() {
    TabSheet movieTabs = new TabSheet();

    movieTabs.addTab(getDetailsTab(), "Details");
    movieTabs.addTab(getTrailerTab(), "Trailers");
    movieTabs.addTab(getImageTab(), "Images");
    movieTabs.addTab(getCastTab(), "Cast");
    movieTabs.addTab(getCrewTab(), "Crew");
    movieTabs.addTab(new SearchLayout(bodyLayout, JSONObject.getObject(RECOMMENDATIONS), true, null, null),
        "Recommendations");

    return movieTabs;
}
```

Slika 18. Kod za prikaz detalja o filmu



U metodi „*getMovieDetails*“ dohvaćamo gotovo iste vrijednosti kao i sa metodom koja nam je prikazivala rezultate pretraživanja tako da neću previše ulaziti u same detalje ove metode pošto su sličnosti poprilično vidljive te je glavna razlika njihov raspored to jest prikaz na ekranu, glavne razlike su prvo dodatak opcije za otvaranje filma na web stranici IMDb-a koja se može vidjeti kod objekta „*imdbLink*“ gdje se na taj objekt veže poveznica koja otvara vanjsku stranicu te ju prikazuje u novoj kartici na pregledniku. Drugo odnosno glavna razlika jest metoda „*getDetailsTabSheet*“ koja nam je zadužena za prikaz kartica koje su vidljive na slici 35, kako bismo se služili karticama potrebno je samo dohvatiti komponentu koju želimo prikazati odabirom određene kartice te naziv same kartice koji se veže uz pripadajuću komponentu. Kao što se vidi za karticu sa dodatnim detaljima smo koristili metodu „*getDetailsTab*“ koja nam vraća komponentu u kojoj se nalaze dodatni detalji o filmu, ta metoda je vidljiva na slici 37.

```
private VerticalLayout getDetailsTab() {
    VerticalLayout movieDetails = new VerticalLayout();

    DecimalFormat decimalFormat = new DecimalFormat("#,##0.00");

    String home = jsonObject.getString(MOVIE_HOME_PAGE);
    Label homePage = CustomItems
        .htmlLabel("<b>Homepage:</b> <a href='" + home + "' target='_blank'" + home + "</a>");
    Label status = CustomItems.htmlLabel("<b>Status: </b>" + jsonObject.getString(MOVIE_STATUS));
    Label releaseDate = CustomItems.htmlLabel("<b>Release date: </b>" + jsonObject.getString(MOVIE_RELEASE_DATE));
    Label runtime = CustomItems.htmlLabel("<b>Runtime: </b>" + String.valueOf(jsonObject.getNumber(MOVIE_RUNTIME)));
    Label budget = CustomItems
        .htmlLabel("<b>Budget: </b>" + decimalFormat.format(jsonObject.getNumber(MOVIE_BUDGET)));
    Label revenue = CustomItems
        .htmlLabel("<b>Revenue: </b>" + decimalFormat.format(jsonObject.getNumber(MOVIE_REVENUE)));

    String collectionString = "<b>Collection: </b>";
    if (!jsonObject.get(MOVIE_COLLECTION).equals(Json.createNull())) {
        collectionString += jsonObject.getObject(MOVIE_COLLECTION).getString("name");
    }
    Label collections = CustomItems.htmlLabel(collectionString);
    Label productionCompanies = getArrayLabel(jsonObject.getArray(MOVIE_PRODUCTION_COMPANY), "Production companies: ");
    Label productionCountries = getArrayLabel(jsonObject.getArray(MOVIE_PRODUCTION_COUNTRY), "Production countries: ");
    Label spokenLanguages = getArrayLabel(jsonObject.getArray(MOVIE_SPOKEN_LANGUAGE), "Spoken languages: ");

    movieDetails.addComponents(homePage, status, releaseDate, runtime, budget, revenue, collections,
        productionCompanies, productionCountries, spokenLanguages);

    return movieDetails;
}

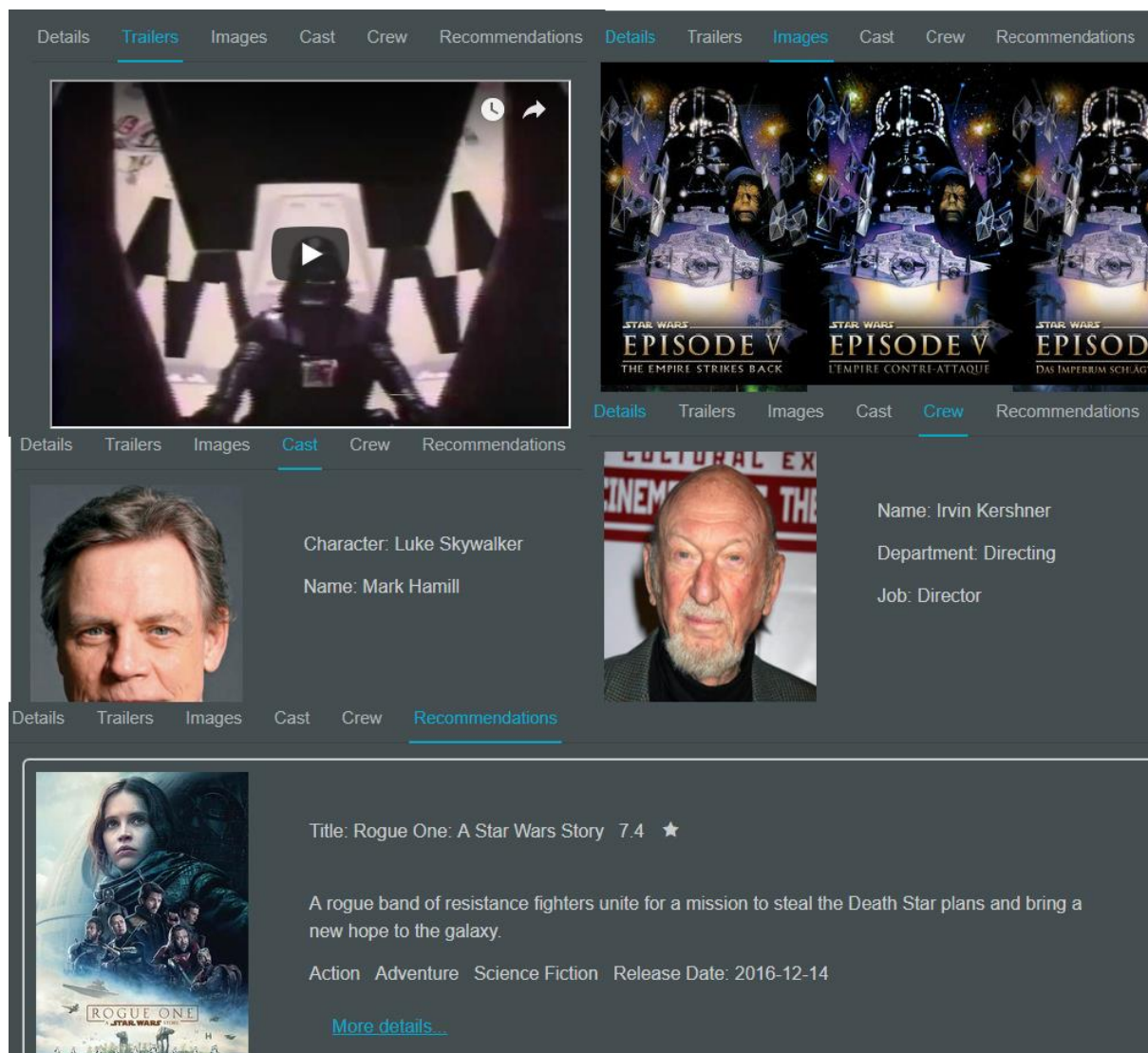
private Label getArrayLabel(JsonArray array, String title) {
    String names = "<b>" + title + "</b>";
    String delim = "";
    for (int i = 0; i < array.length(); i++) {
        names += delim + array.getObject(i).getString("name");
        delim = ", ";
    }
    Label l = CustomItems.htmlLabel(names);

    return l;
}
```

Slika 37.. Dodatni detalji o filmu

Na prikazanom kodu se uglavnom nalazi prikaz dohvaćanih vrijednosti kao i u prethodnim funkcionalnostima sa razlikom da su se ovdje za određene vrijednosti koristile opcije da prikazujemo tekst u obliku HTML-a što nam omogućuje lagano modificiranje određenih stilova teksta poput podebljavanja kako bi se lakše raspoznale vrijednosti od samih opisa na ekranu ili izrade poveznice na vanjsku stranicu putem HTML-a, također je i

napravljena metoda koja prolazi kroz polje određenih tekstualnih vrijednosti te ih zapisuje kao u nizu odvojenim sam zarezmom. Na slici 38 se nalazi prikaz ostalih kartica koje su dostupne na detaljima o filmu.



Slika 19. Prikaz preostalih mogućnosti na detaljima

Preostale opcije koje imamo na detaljima o filmu kao što se vidi iz slike su prikaz videa putem Youtube-a, zatim prikaz svih dostupnih slika, prikaz glumaca te prikaz ostalih članova tima koji su radili na filmu te na kraju imamo i preporuke za filmove. Na slici 39 se nalaze metode za dohvaćanje videa i slika za odabrani film koje se pozivaju kao i dodatni detalji o filmovi što je i vidljivo na slici 36.



```

private Component getTrailerTab() {
    CssLayout trailers = new CssLayout();
    trailers.setWidth("1000px");
    for (int i = 0; i < jsonObject.getObject(VIDEOS).getArray(RESULTS).length(); i++) {
        Label l = CustomItems.htmlLabel("<iframe width='450' height='300' style='margin:15px;' src='"
            + YOUTUBE_TRAILER_ROOT + jsonObject.getObject(VIDEOS).getArray(RESULTS).getObject(i).getString("key")
            + "' allowfullscreen ></iframe>");
        trailers.addComponent(l);
    }
    if (trailers.getComponentCount() < 1) {
        trailers.addComponent(new Label("No trailers available..."));
    }
    return trailers;
}

private Component getImageTab() {
    CssLayout images = new CssLayout();
    images.setWidth("1000px");

    String title = jsonObject.getString(TITLE);
    String imageID = String.valueOf((int) jsonObject.getNumber("id"));
    JSONArray moviePosters = jsonObject.getObject(IMAGES).getArray(POSTERS);

    for (int i = 0; i < moviePosters.length(); i++) {
        String poster = moviePosters.getObject(i).getString(POSTER_FILE_PATH);
        Image img = CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_185, true, null);
        images.addComponent(img);
    }
    if (images.getComponentCount() < 1) {
        images.addComponent(new Label("No images available..."));
    }
    return images;
}

```

*Slika 39. Dohvaćanje videa i slika*

Kao što se vidi za videa smo prvo provjerili dali uopće postoje te ukoliko ne postoje ispisali smo odgovarajuću poruku, ukoliko postoje videa korišten je ponovno HTML prikaz kako bismo mogli vidjeti videa unutar stranice te kako ne bismo morali odlaziti ili praviti poveznice na vanjske stranice kako bismo vidjeli željeni video. Unutar Vaadina postoji nekoliko različitih opcija koje omogućuju prikaz Youtube videa putem njihovih objekata unutar samoga prozora no za većinu tih opcija nije omogućeno prikazivanje videa preko cijeloga ekrana tako da sam se odlučio za opciju putem HTML-a koja se ispostavila najjednostavnijom. Dohvaćanje slika je već bilo prikazano nekoliko puta u prethodnim funkcionalnostima tako da u ovom slučaju nema mnogo razlika, glavna je ta da iz ove metode koristimo opciju koja nam omogućuje pritiskom na sliku prikaz sliku u zadanoj veličini unutar druge kartice na pregledniku. Na slijedećoj slici 40 se nalaze metode koje prikazuju osobe koje su sudjelovale kod izrade odabranog filma. Jedino na što treba pripaziti je da nemaju sve osobe jednake detalje odnosno mnoge osobe nemaju popunjene neke varijable što znači da moramo pripaziti kako čitamo JSON objekt te paziti na nepostojeće vrijednosti.

```

private Component getCastTab() {
    VerticalLayout castsLayout = new VerticalLayout();
    JSONArray castsArray = jsonObject.getJSONObject(CREDITS).getArray(CAST);

    for (int i = 0; i < castsArray.length(); i++) {
        HorizontalLayout hl = new HorizontalLayout();
        VerticalLayout castDetails = new VerticalLayout();

        String title = castsArray.getJSONObject(i).getString(CAST_CHARACTER);
        String imageID = String.valueOf((int) castsArray.getJSONObject(i).getNumber("id"));
        String poster = castsArray.getJSONObject(i).get(CAST_CREW_PROFILE_PATH).jsEquals(Json.createNull()) ? ""
            : castsArray.getJSONObject(i).getString(CAST_CREW_PROFILE_PATH);

        Image img = CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_185, true, null);

        Label character = new Label("Character: " + title);
        Label actor = new Label("Name: " + castsArray.getJSONObject(i).getString(CAST_CREW_NAME));

        castDetails.addComponents(character, actor);
        hl.addComponents(img, castDetails);
        castsLayout.addComponent(hl);
    }

    return castsLayout;
}

private Component getCrewTab() {
    VerticalLayout crewLayout = new VerticalLayout();

    JSONArray crewArray = jsonObject.getJSONObject(CREDITS).getArray(CREW);

    for (int i = 0; i < crewArray.length(); i++) {
        HorizontalLayout hl = new HorizontalLayout();
        VerticalLayout crewDetails = new VerticalLayout();

        String title = crewArray.getJSONObject(i).getString(CAST_CREW_NAME);
        String imageID = String.valueOf((int) crewArray.getJSONObject(i).getNumber("id"));
        String poster = crewArray.getJSONObject(i).get(CAST_CREW_PROFILE_PATH).jsEquals(Json.createNull()) ? ""
            : crewArray.getJSONObject(i).getString(CAST_CREW_PROFILE_PATH);
        String departmentName = crewArray.getJSONObject(i).get(CREW_DEPARTMENT).jsEquals(Json.createNull()) ? ""
            : crewArray.getJSONObject(i).getString(CREW_DEPARTMENT);
        String jobName = crewArray.getJSONObject(i).get(CREW_JOB).jsEquals(Json.createNull()) ? ""
            : crewArray.getJSONObject(i).getString(CREW_JOB);

        Image img = CustomItems.getImage(poster, title, imageID, POSTER_IMAGE_185, true, null);

        Label name = new Label("Name: " + title);
        Label department = new Label("Department: " + departmentName);
        Label job = new Label("Job: " + jobName);

        crewDetails.addComponents(name, department, job);
        hl.addComponents(img, crewDetails);
        crewLayout.addComponent(hl);
    }

    return crewLayout;
}

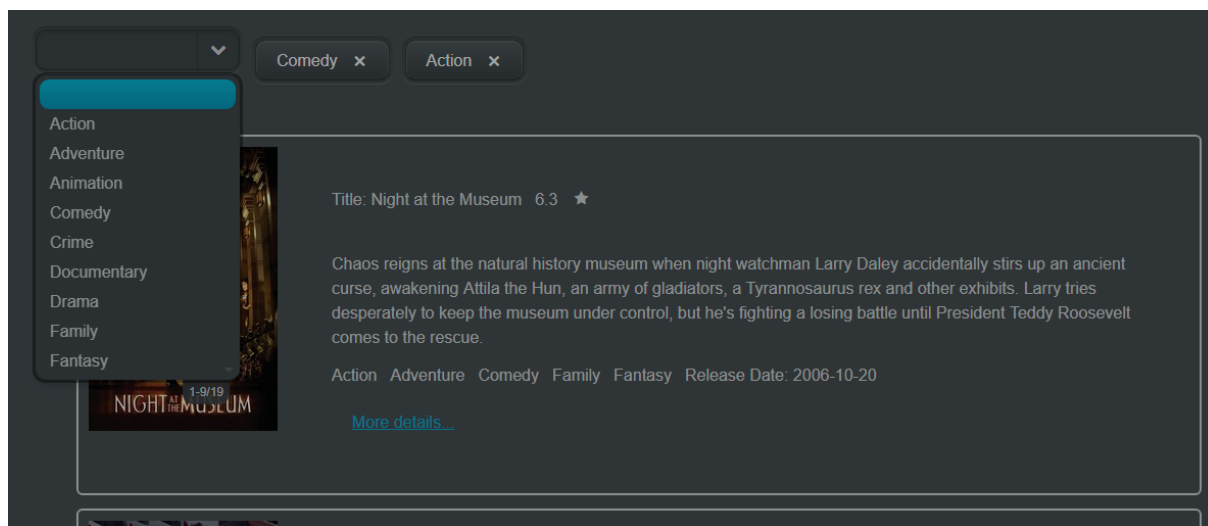
```

Slika 40. Dohvaćanje osoba na filmu

Preostala kartica koja prikazuje preporučene filmove se služi već postojećim metodama za pretraživanje koje su već bile prethodno opisane, jedina razlika je ta što se prosljeđuje JSON objekt koji dohvaćen sa drugim upitom odnosno sa upitom koji dohvaća preporučene filmove za odabrani film te je poslana varijabla koja pokazuje da se radi o upitu sa stranice sa detaljima iliti da se radi o preporučenim filmovima, što se koristi kako se ne bi prikazivale stranice s obzirom da se prema odabranoj funkcionalnosti prikazuje samo top 20 preporuka.

## 4.6. Pretraživanje po žanrovima

Pretraživanje je također moguće i vršiti odabirom žanrova te prikazom svih dostupnih filmova prema odabranim žanrovima što je poprilično korisno ukoliko želimo pronaći neki novi film ili možda neki film koji smo već vidjeli ali smo zaboravili naziv. Unutar aplikacije možemo doći do prikaza za pretraživanje pomoću žanrova putem opcije „Browse“ iz zaglavlja, jedan primjer takvog pretraživanja se može vidjeti na slici 41.



Slika 4120. Pretraživanje po žanrovima.

Na slici se može vidjeti jedan padajući izbornik u kojemu se nalaze svi dostupni žanrovi pomoću kojih možemo vršiti pretragu te odabirom bilo kojeg žanra sa tog pipisa će kreirati novi element u kojemu će biti prikazan kako bi korisnik znao, ukoliko vrši pretragu putem više žanrova, koje sve žanrove trenutno ima unutar svoje pretrage. Ukoliko se više ne želi filtrirati po određenom žanru možemo jednostavno pritisnuti na njega unutar liste sa elementima koje prikazuju trenutno prikazane žanrove. Sam popis rezultata sadrži jednake rezultate kao i rezultat običnog pretraživanja koji je bio već prethodno pojašnjen, iako se na samoj slici ne vidi, i ovaj način prikazivanja ima omogućeno obilježavanje stranica koje se nalazi na dnu prikazane stranice te funkcionira jednako kao i za obično pretraživanje filmova po nazivu. Na slici 42 se može vidjeti dio koda koji je zaslužan za prikaz prethodno opisane funkcionalnosti.

```

public class BrowseGenresLayout extends VerticalLayout {

    private static final long serialVersionUID = 1L;

    private BodyLayout bodyLayout;
    private String genreHistory = "";
    private Set<Integer> genreIds = new HashSet<>();
    private String genres;
    private int pageId;
    private HashMap<Integer, String> genreMap = JsonSingleton.getInstance().getGenreMap();

    private CssLayout genrePickerLayout = new CssLayout() {
        private static final long serialVersionUID = 1L;

        @Override
        protected String getCss(Component c) {
            if (c instanceof Button) {
                return "margin-left: 15px; margin-top: 10px;";
            }
            return super.getCss(c);
        }
    };

    public void loadData(BodyLayout bodyLayout, String genres, int page) {
        this.bodyLayout = bodyLayout;
        this.genres = genres;
        this.pageId = page;

        setPickerLayout();
        if (genres != null) {
            genreHistory = NAVIGATION_GENRE_IDS + genres + NAVIGATION_RESULT_PAGE_NUMBER + pageId;
            fillData();
        }
        Page.getCurrent().pushState(NAVIGATION_BROWSE_PAGE + genreHistory);
    }

    private void fillData() {

        genreIds = Stream.of(genres.split(",")).map(Integer::parseInt).collect(Collectors.toSet());

        for (Integer i : genreIds) {
            selectionListener(genreMap.get(i), i, true);
        }
        browse();
    }

    private void browse() {
        if (getComponentCount() > 1 && getComponent(1) != null) {
            removeComponent(getComponent(1));
        }
        JsonObject jsonObject = JsonSingleton.getInstance()
            .getJsonObjectFromURL(BROWSE_GENRE.replace(WITH_GENRES, WITH_GENRES + genres) + pageId);

        addComponent(new SearchLayout(bodyLayout, jsonObject, false, null, genres));
    }
}

```

Slika 42. Kod za prikaz rasporeda elemenata kod pretraživanja žanrovima .

Kao što vidimo i sama klasa nasljeđuje drugu klasu koja nam služi za prikaz elemenata tako da unutar ove klase možemo jednostavno dodavati željene elemente kao što bi to činili i sa nekim drugim objektima ovoga tipa. Također unutar dijela sa deklaracijom možemo uočiti da dohvaćamo vrijednost mape žanrova iz prethodno objašnjene singleton klase, nakon čega nam slijedi jedan element za izvršavanje rasporeda koji će nam prikazivati elemente sa žanrovima koji se mogu vidjeti na slici 41, te smo kod njega nadjačali metodu koja se koristi za prikaz stilova nad elementima unutar njega te smo dodali određene popravke margina kako se elementi ne bi preklapali međusobno. Slijedeća metoda „loadData“ nam poziva prvo drugu

metodu koja služi za postavljanje elemenata te ukoliko je bilo prosljeđenih identifikacijskih brojeva žanrova poziva metodu „*fillData*“ koja se nalazi odmah nakon nje. S obzirom da se popis žanrova za pregled sprema u tekstualnom obliku odvojenim zarezom ova metoda prvo transformira taj zapis te zatim poziva metodu za pretraživanje „*browse*“ u kojoj se dohvaća novi JSON objekt sa rezultatima pretrage po žanrovima, također kao što sam napomenuo dodaje se isti objekt kao i kod pretraživanja, na slijedećoj slici 43 se nalazi ostatak koda koji nam omogućuju ovu funkcionalnost.

```
private void setPickerLayout() {
    genrePickerLayout.setWidth("1100px");
    ArrayList<String> genreNames = new ArrayList<>();

    genreMap.forEach((key, value) -> {
        genreNames.add(value);
    });
    genreNames.sort(String::compareToIgnoreCase);
    ComboBox<String> genreBrowse = new ComboBox<>();

    genreBrowse.addSelectionListener(e -> {
        final Integer idGenre = genreMap.entrySet().stream()
            .filter(entry -> Objects.equals(entry.getValue(), e.getValue()))
            .map(Map.Entry::getKey)
            .findFirst()
            .orElse(null);
        selectionListener(e.getValue(), idGenre, false);
        browse();
    });
    genreBrowse.setItems(genreNames);
    genrePickerLayout.addComponent(genreBrowse);
    addComponents(genrePickerLayout);
}

private String getGenreIds(Set<Integer> genresId) {
    String result = "";
    String delim = "";
    for (Integer id : genresId) {
        result += delim + id;
        delim = ",";
    }
    return result;
}

private void selectionListener(String name, int idGenre, boolean refresh) {
    if (refresh || !genreIds.contains(idGenre)) {
        Button selectedBtn = CustomItems.createComboBoxButtons(name);
        selectedBtn.addClickListener(event -> {
            selectedBtn.setVisible(false);
            genreIds.remove(idGenre);
            genres = getGenreIds(genreIds);
            browse();
            Page.getCurrent().pushState(NAVIGATION_BROWSE_PAGE + NAVIGATION_GENRE_IDS + genres
                + NAVIGATION_RESULT_PAGE_NUMBER + pageId);
        });
        genrePickerLayout.addComponent(selectedBtn);

        genreIds.add(idGenre);
        genres = getGenreIds(genreIds);
        Page.getCurrent().pushState(
            NAVIGATION_BROWSE_PAGE + NAVIGATION_GENRE_IDS + genres + NAVIGATION_RESULT_PAGE_NUMBER + pageId);
    }
}
```

Slika 43. Kod za prikaz elemenata te izbor žanrova

Prva prikazana metoda na slici nam služi za kreiranje te dodavanje padajućeg izbornika sa popisom svih žanrova, kao što se vidi prvo prolazimo kroz mapu gdje imamo zapisane nazive žanrova te sve nazive prebacujemo unutar druge liste koju zatim sortiramo te postavljamo kao popis unutar izbornika. Prilikom odabira odnosno pritiska na neki od ponuđenih žanrova sa

popisa dohvaćamo identifikacijski broj odabranog žanra te ga prosljeđujemo u metodu „*selectionListener*“ koja nam služi za prikaz elementa koji prikazuju žanrove koji su trenutno prikazani, također se nakon samog pritiska na žanr poziva i metoda „*browse*“ koju smo prethodno opisale te nam ovdje služi kako bismo mogli prikazati nove rezultate sa ovim filtrima. Unutar spomenute metode kreiramo elemente koji prikazuju trenutne filtre te tim elementima dodajemo naredbu da nakon što se pritisnu maknu iz samog popisa te da se ponovno obavi pretraga po žanrovima ovoga puta bez njihovog doprinosa.

## 4.7. Navigacija i povijest

S obzirom da Vaadin omogućuje kreiranje web aplikacija kao aplikaciju sa samo jednom web stranicom (engl. Single Page Application - SPA) što iako je poprilično interesantno te se doima da stranica radi mnogo usklađenije nego li stranice koje konstanto kreiraju novu web stranicu te konstantno osvježavaju njen sadržaj, ima svojih mana od čega je naravno glavno povijest pretraživanja te sama navigacija aplikacije putem URL-a koji je konstantno jednak tokom korištenja aplikacije. Kako bismo riješili ove probleme bilo je potrebno za svaku stranicu koju želimo zabilježiti forsirati njeno bilježenje u povijest preglednika te u sami URL kako bismo mogli vršiti navigaciju ili čak ukoliko želimo nastaviti sa sadržajem koji smo prethodno pregledavali. Kako bismo natjerali URL da se promjeni te zapiše u povijest preglednika koristili smo Vaadin-ovu metodu „*pushState*“ koju se može uočiti u prijašnjim slikama iako ju nisam spomenuo.

Kao što je prethodno već spomenuto, svaki puta kada se učitava aplikacija pokreće se metoda „*init*“ unutar klase „*MyUI*“ što nam znači da je najjednostavnije rješenje pratiti vrijednosti iz URL-a prilikom pokretanja aplikacije iz te metode te kao što se može vidjeti na slici 44 tako sam i ja započeo pregledavanje URL-a. Prvo što u ovoj metodi možemo vidjeti je dohvaćanje stranice te dodavanje iliti osluškivanje promjena putem metode „*addPopStateListene*“, ova metoda nam omogućuje reagiranje na bilo kakve promjene unutar URL-a koje se dešavaju tranzicijom tipki naprijed ili natrag unutar našeg web preglednika, ukoliko se nalazimo na početnoj stranici dodajemo joj oznaku te stranice ili u suprotnom vršimo navigaciju uz pomoć URL-a te metode „*navigateToPage*“.

```

@Override
protected void init(VaadinRequest vaadinRequest) {
    getPage().addPopStateListener(new PopStateListener() {

        private static final long serialVersionUID = 1L;

        @Override
        public void uriChanged(PopStateEvent event) {
            navigateToPage(Page.getCurrent().getLocation().toString());
        }
    });

    if (Page.getCurrent().getLocation() == null || Page.getCurrent().getLocation().toString().isEmpty()
        || !Page.getCurrent().getLocation().toString().contains("?")) {
        Page.getCurrent().pushState(NAVIGATION_START_PAGE);
    } else {
        navigateToPage(Page.getCurrent().getLocation().toString());
    }
    mainLayout.addComponents(headerLayout, bodyLayout);
    setContent(mainLayout);
}

private void navigateToPage(String url) {
    String[] splitUrl = url.split("\\?");
    String[] splitVariables = splitUrl[1].split("&");
    int pageId = Integer.parseInt(splitVariables[0].split("=")[1]);

    switch (pageId) {
        case 1:
            bodyLayout.searchForMovies(false, null, 0);
            break;
        case 2:
            String searchQuery = splitVariables[1].split("=")[1];
            int resultPageNum = Integer.parseInt(splitVariables[2].split("=")[1]);
            bodyLayout.searchForMovies(false, searchQuery, resultPageNum);
            break;
        case 3:
            int movieId = Integer.parseInt(splitVariables[1].split("=")[1]);
            bodyLayout.showMovieDetails(movieId);
            break;
        case 4:
            String genres = null;
            int resPage = 1;
            if (splitVariables.length > 1) {
                genres = splitVariables[1].split("=")[1];
                resPage = Integer.parseInt(splitVariables[2].split("=")[1]);
            }
            bodyLayout.browseGenres(false, genres, resPage);
            break;
        default:
            break;
    }
}
}

```

Slika 44. Kod za navigaciju kroz aplikaciju

Prije nego li pojasnim navedenu metodu potrebne je spomenuti da je za potrebe identifikacijskih brojeva bila izrađena zasebna „enum“ klasa u kojoj sam povezao brojeve sa nekim logičnim tekstualnim pojmovima te sam ih nakon toga spojio unutar klase sa konstantama što se može vidjeti na slici 45, te samo korištenje se može vidjeti i metodi „init“ gdje se postavlja vrijednost na početnu stranicu. Izvršavanje same navigacije je poprilično jednostavno pošto imamo URL u kojemu smo prethodno spremili vrijednosti koje su nam potrebne kako bismo prikazali željene stranice. Prvi korak u metodi je odvajanje varijabli te

njihovih vrijednosti od ostatke trenutno bespotrebnog teksta, te nakon izdvajanje varijabli možemo jednostavno pozvati metode koje smo tokom prethodnih poglavlja opisali.

```
public final static String NAVIGATION_START_PAGE=NAVIGATION_PAGE_ID+PageIds.START.getPageId();  
public final static String NAVIGATION_SEARCH_PAGE=NAVIGATION_PAGE_ID+PageIds.SEARCH.getPageId()+QUERY;  
public final static String NAVIGATION_DETAILS_PAGE=NAVIGATION_PAGE_ID+PageIds.MOVIE_DETAILS.getPageId()+NAVIGATION_MOVIE_ID;  
public final static String NAVIGATION_BROWSE_PAGE=NAVIGATION_PAGE_ID+PageIds.BROWSE.getPageId();
```

*Slika 45. Konstante za navigaciju*



## 5. Zaključak

Putem ovoga rada sam prikazao kako se može izraditi jedna jednostavna web aplikacija za pretraživanje ili pregledavanje detalja o filmovima te sam također na primjeru same aplikacije prikazao kako se može koristiti TMDb API te obrađivati JSON format kako bismo mogli izvući potrebne informacije iz JSON zapisa. Prethodno sam spomenuo da postoje i druge mogućnosti osim TMDb-a te iako on nije možda idealan za neke svrhe smatram da je definitivno najbolji izbor pošto je veoma stabilan, brz te kao što sam već i napomenuo ima odlično izrađenu dokumentaciju iz koje se može veoma lako izvući što nam je potrebno te se može brzo naučiti služiti njihovim API-em. Osim korištenog API-a također sam malo detaljnije prikazao korišteni okvir za izradu web aplikacija putem java jezika, ovaj izbor smatram da je poprilično subjektivan te ovisi uglavnom što programer želi te u kojoj tehnologiji voli raditi u mojem slučaju odlučio sam se za javu.

TMDb sadrži poprilično veliku količinu podataka te je sama brzina kojom se dobivaju rezultati veoma impresivna te je limit koji postoji kod slanja zahtjeva poprilično velik i obični korisnik ga ne bih trebao moći nadmašiti pogotovo zato što nam je omogućeno i spajanje upita odnosno dohvaćanje više rezultata sa samo jednim upitom. Smatram da je TMDb odličan izbor za programere koji izrađuju aplikaciju u kojoj su im potrebne bilo kakve informacije o filmovima te kao što se može vidjeti iz prethodno prikazanih primjera korištenja na samoj aplikaciji poprilično je pouzdan te ga se veoma lako može naučiti koristiti.

## 6. Literatura

- [1] Beal Vangie (s.a.) Java Dostupno 10.09.2017  
<http://www.webopedia.com/TERM/J/Java.html>
- [2] Berlind David (2015) APIs are like user interfaces just with different users in mind. Dostupno 12.08.2017  
<https://www.programmableweb.com/news/apis-are-user-interfaces-just-different-users-mind/analysis/2015/12/03>
- [3] Bloch Joshua (2008) Effective Java Dostupno 10.09.2017  
[http://files.blogjava.net/jlin/Effective\\_Java\\_2nd\\_Edition.pdf](http://files.blogjava.net/jlin/Effective_Java_2nd_Edition.pdf)
- [4] Casciato Michael (2016) The Movie DB(TMDb) vs. Open Movie DB(OMDB). Dostupno 12.08.2017 <https://medium.com/@mcasciato/no-imdb-api-check-out-these-options-75917d0fe923>
- [5] Frankel Nicolas (2011) Learning Vaadin Dostupno 10.09.2017  
[https://books.google.hr/books?id=pQDJAAAAQBAJ&pg=PT329&lpg=PT329&dq=Learning+Vaadin+nicolas&source=bl&ots=t-LLhKT44M&sig=LKQFF\\_Va34aSSXEdNKN-ekXziGU&hl=hr&sa=X&ved=0ahUKEwiugqyl9JPWAhVEWRQKHZlrBOKQ6AEIVjAG#v=onepage&q=Learning%20Vaadin%20nicolas&f=false](https://books.google.hr/books?id=pQDJAAAAQBAJ&pg=PT329&lpg=PT329&dq=Learning+Vaadin+nicolas&source=bl&ots=t-LLhKT44M&sig=LKQFF_Va34aSSXEdNKN-ekXziGU&hl=hr&sa=X&ved=0ahUKEwiugqyl9JPWAhVEWRQKHZlrBOKQ6AEIVjAG#v=onepage&q=Learning%20Vaadin%20nicolas&f=false)
- [6] Joshi Rohit (2015) Java Design Patterns Dostupno 10.09.2017  
<http://enos.itcollege.ee/~jpoial/java/naited/Java-Design-Patterns.pdf>
- [7] Oracle Corporation (2017) What is MySQL Dostupno 10.09.2017  
<https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>
- [8] Park Se Hoon (2017) Understanding JVM Internals Dostupno 10.09.2017  
<http://www.cubrid.org/blog/understanding-jvm-internals>
- [9] Sintes Tony (2001) Just what is the Java API Anyway Dostupno 10.09.2017  
<https://www.javaworld.com/article/2077392/java-se/just-what-is-the-java-api-anyway.html>
- [10] TMDb (2017) Let's talk about TMDb Dostupno 10.09.2017  
<https://www.themoviedb.org/about>
- [11] Vaadin Team (2017) Vaadin Framework – Introduction – Overview Dostupno 10.09.2017 <https://vaadin.com/docs/-/part/framework/introduction/intro-overview.html>