

# Форум за филмови со користење на GraphQL



Изработиле

193181 – Петар Парталоски

193113 – Мартин Николов

Ментор

Проф. д-р Бобан Јоксимовски

## Содржина

GraphQL .....	3
За проектот .....	4
База на податоци .....	5
Апликативен дизајн .....	8
Имплементација на GraphQL во проектот .....	16
Заклучок .....	29

## GraphQL

**GraphQL** претставува едноставен прашален јазик за **Rest API** на дадена апликација. Развиен е од компанијата **Facebook** и истиот е слободно достапен за користење. Имено, овозможува корисникот да побара точно она што посакува за податоците од самата апликација преку однапред дефинирана шема за побарување, имплементира на серверската страна. Преку оваа шема, се подразбира што може секој прашалник да врати како одговор. Под корисник се подразбира секоја апликација, обичен корисник или било која друга засегната странка, што има потреба од комуникација со дадената апликација.

Ваквиот пристап овозможува побрзи и постабилни апликации со збогатено корисничко искуство бидејќи конекцијата не се прептоварува со вишок и непотребни податоци и целата контрола оди на корисник за побарување на податоците. Секој корисник има слобода да побара различен волумен на податоци. Со ова се овозможува голема флексибилност за приказ на податоци на дадениот корисник, поради тоа што, доколку има потреба да побара или да намали податоци, едноставно ги запишува или не истите во соодветниот прашалник, без никаква дополнителна интервенција од серверска страна.

За разлика од сите видови **HTTP** за **Rest API**, **GraphQL** овозможува само **POST** барања каде во телото на барањето е напишан самиот прашалник. Ги поддржува сите **CRUD** операции, каде барањето на податоци оди преку обичен прашалник, а операциите на измена преку соодветна мутација. Сите барања преку **GraphQL** завршуваат на едно исто урл наместо на повеќе при стандарно **Rest API**. Важно е да се напомене дека **GraphQL** преку едно барање, може да се преземат објекти кои воопшто не се корелирани меѓусебе, додека кај стандарден **Rest API** би се пратиле повеќе **HTTP** барања на разни крајни урл.

```

{
  "data": {
    "actors": [
      {
        "personId": 6,
        "name": "Will",
        "surname": "Smith",
        "imageUrl": "https://static.wikia.nocookie.net/id4/images/7/79/W",
        "dateOfBirth": "1968-09-25",
        "actors": [
          {
            "movie": {
              "movieId": 6,
              "title": "Paris pieds nus",
              "description": "Fiona го посетува Paris по прв пат с",
              "imageUrl": "https://m.media-amazon.com/images/M/MV5"
            }
          }
        ]
      }
    ]
  }
}

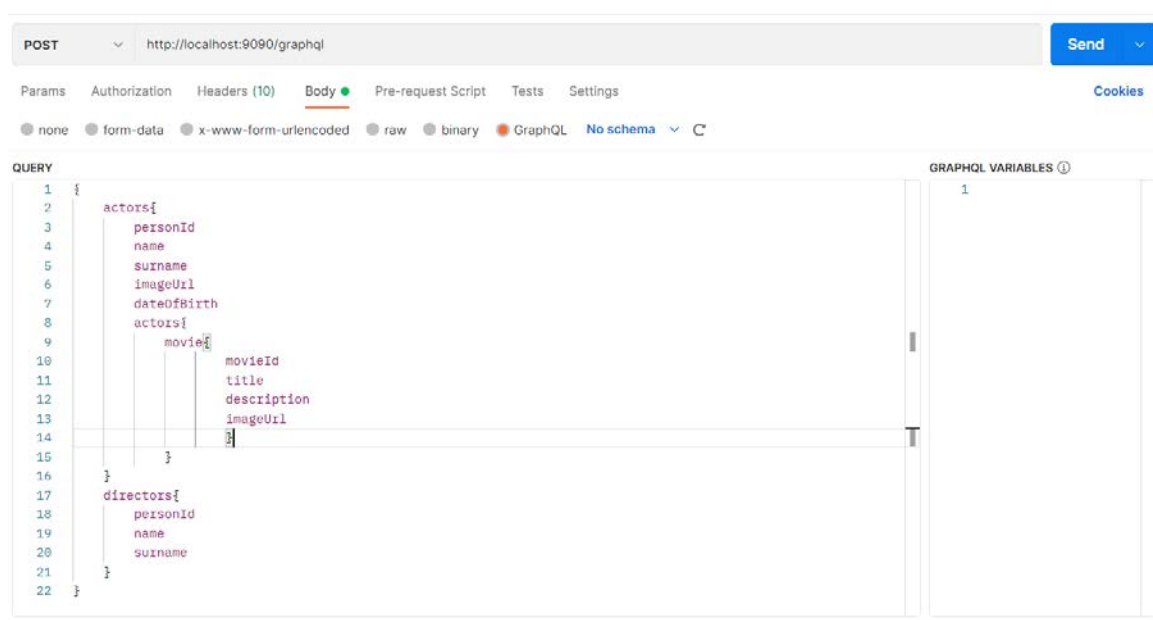
```

Се што побарува е дефинирање на строга шема во која се пишува што се може да биде пристапено и под кое име или пократко со користење на анотации врз дадените методи и својства на објектите. Проектот кој ние го имплементиравме, го користевме принципот со анотации за поконцизен и поточен код.

Слика 1. Пример прашалник за нашиот проект

Слика 2. Дел од одговорот на прашалникот проследен во Слика 1

## За проектот



Проектот претставува форум за филмови, актери и режисери каде корисниците ќе можат да отворат дискусија за посакуваниот филм или актер или режисер, да се спротистават или поддржат мислење

преку реплика на дадена дискусија, да остават рејтинг за посакуваниот ентитет. Исто така, имаат можност, да ги прегледаат рејтинзите и допаѓањата на останатите корисници што ќе им помогнат во процесот на носење одлука за гледање на следен филм бидејќи скоро секогаш се троши доста време во барање филм за гледање. Форумот е имплементиран на веќе креирана база, реализирана од тимот на проектот, по предметот Бази на податоци, под менторство од страна на вонр. проф. Вангел Ајановски.

**Проектот е темелно прикажан линкот долу, на гранката „graphql“**

**<https://github.com/Mato-77/WeDiscussMovies/tree/graphql>**

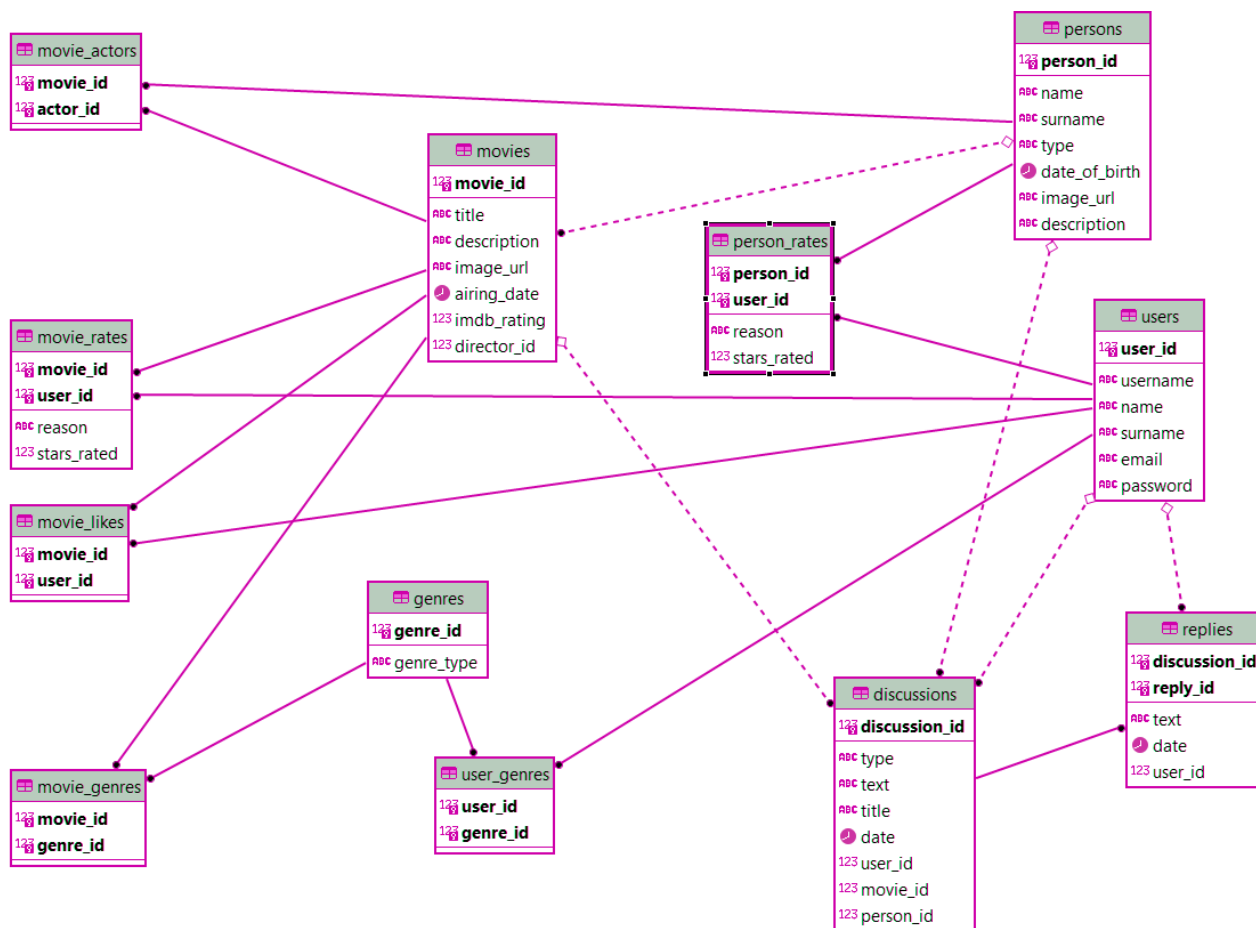
За успешно стартување на проектот, најпрвин е потребно извршување на тунел скриптата лоцирана во датотеката **src/tunnel/tunnel\_scripta**. Потребно откако ќе ја отворите скриптата, два пати да претиснете Ентер и ќе може проектот да го стартувате. Во спротивно, проектот нема да се стартува бидејќи конекцијата со базата нема да биде успешна.

## База на податоци

Базата на податоци за проектот се состои од 14 табели и истата преставува **PostgreSQL**. Приказ ЕР дијаграм на базата е даден подолу. За сите табели, освен оние со релациите, се користени вештачки сурогат клучеви, со цел полесно и побрзо споредување кога има потреба за пребарување во табелите. Филмовите, актерите и режисерите се претставени со табелите **movies** и **persons**, каде соодветно во табелата **persons** се чува **type** – карактер со вредност А што означува актер или вредност D што означува режисер за соодветниот запис. Сите атрибути за актер или режисер се задолжителни при внесување на нов запис, додека за филм само опционални се **imdb\_rating** и **director\_id** што референцира до соодветниот ред во табелата **persons** и насловот на филмот е задолжително да биде уникатен.

Атрибути за актер или режисер

- **person\_id** – број, вештачки генериран идентификатор од страна на базата
- **name** – текстуален податок, име
- **surname** – текстуален податок, презиме
- **type** – карактер, соодветен тип, вредност А или D
- **date\_of\_birth** – датум, датум на раѓање
- **image\_url** – текстуален податок, линк кон соодветна слика за приказ на актерот или режисерот
- **description** – текстуален податок, опис за дадениот актер или режисер



Слика 3. Приказ на EP дијаграм за дадената база

#### Атрибути за филм

- **movie\_id** – број, вештачки генериран идентификатор од страна на базата
- **title** – текстуален податок, наслов на филмот
- **description** – текстуален податок, опис на филмот
- **image\_url** – текстуален податок, линк за приказ на слика за филмот
- **airing\_date** - датум, премиера на дадениот филм
- **imdb\_rating** - број, рејтинг според IMDb
- **director\_id** – број, референца кон запис за режисер на дадениот филм, еден за даден филм

Корисниците на форумот се репрезентираат во табелата **users**. Сите атрибути се задолжителни, додека име и емаил претставени со **username** и **email** мораат да се и автентични.

#### Атрибути за корисник

- **user\_id** – број, вештачки генериран идентификатор од страна на базата
- **name** – текстуален податок, име на корисник
- **surname** – текстуален податок, презиме на корисник
- **username** - текстуален податок, корисничко име на корисник
- **email** – текстуален податок, е-пошта на корисник
- **password** – текстуален податок, лозинка на корисник, хеширана со алгоритам

Дискусиите се мапирани во табелата **discussions**. Дискусијата дали се однесува за филм или личност(актер или режисер), се препознава по атрибутот **type**, што има вредност М за филм или Р за личност соодветно. При вредност М, **person\_id** не се поставува, додека при вредност Р, **movie\_id** не се поставува. Опционални атрибути се **person\_id** и **movie\_id**, но во даден момент, согласно со вредноста на **type**, едниот мора да биде поставен, а другиот да нема вредност.

Атрибути за дискусија

- **discussion\_id** – број, вештачки генериран идентификатор од страна на базата
- **type** – карактер, Р за личност или М за филм
- **text** – текстуален податок, текст на дискусијата
- **title** - текстуален податок, наслов на дискусија
- **date** – датум, датум на објавување на дискусија
- **user\_id** – број, референца кон креатор на дискусија
- **movie\_id** – број, референца за филмот за кој се однесува дискусијата
- **person\_id** – број, референца за личноста за која се однесува дискусијата

Репликите се поставени во табелата **replies**. Сите атрибути се задолжителни.

Атрибути за реплика

- **reply\_id** – број, вештачки генериран идентификатор од страна на базата
- **discussion\_id** – број, референца за која дискусија се однесува репликата
- **text** – текстуален податок, текст на реплика
- **date** – датум, датум на објавување на реплика
- **user\_id** – број, референца кон креатор на реплика

Жанровите се поставени во табелата **genres**. Се состои од вештачки генериран идентификатор и текстуален податок што означува име на жанрот. Името на жанрот е задолжителен атрибут.

Табелите **movie\_likes**, **reply\_likes**, **discussion\_likes** се релации меѓу филм, реплика и дискусија соодветно со корисникот на кој му се има допаднато споменатите ентитети. Се составени само од идентификаторите за соодветните записи во табелите. Кардиналност на релациите изнесува многу-многу.

**Movie\_rates**, **person\_rates** – ги прикажуваат соодветните рејтинзи и причина за поради која дадениот корисник оставил рејтинг за соодветниот филм или личност, додека пак **user\_genres**, **movie\_genres**, ја илустраат релацијата за тоа кои жанрови му се допаѓаат на даден корисник и на кои жанрови припаѓа филмот соодветно. Кардиналност е исто и овде многу-многу.

Актерите на филмот се претставени преку релацијата **movie\_actors** на табелата **movies** со **persons**. Кардиналноста и овде е многу-многу.

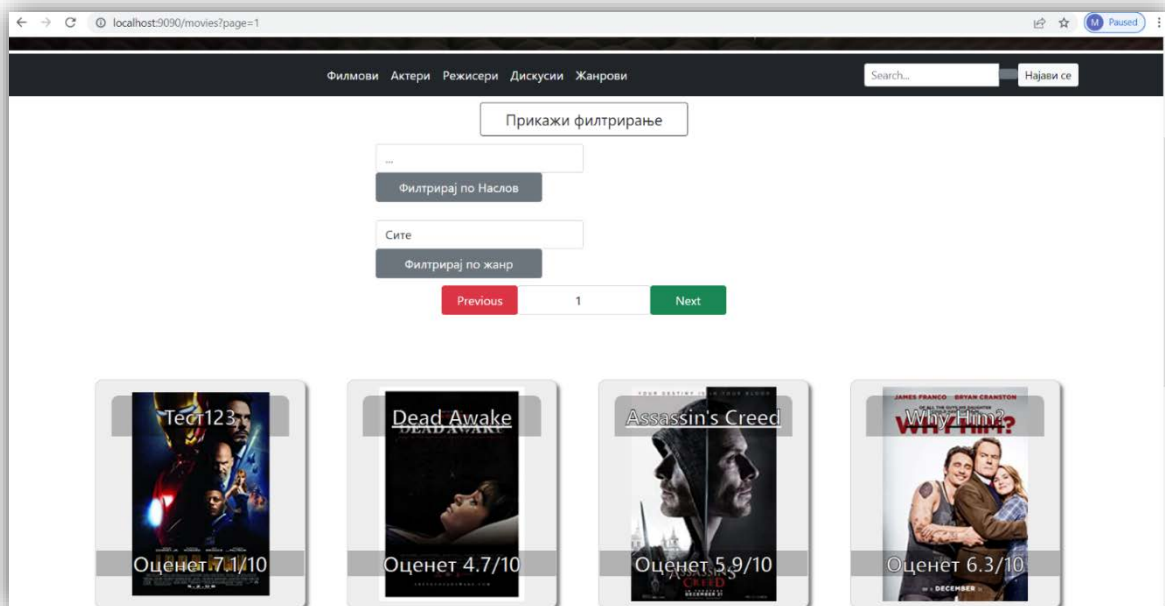
## Апликативен дизајн

Форумот има два типа на корисници:

- Ненајавен корисник – се достапни само опциите за прегледување на податоци
- Најавен корисник – се достапни сите опции односно прегледување, додавање, уредување, бришење, допаѓање и оставање рејтинг на сите ентитети соодветно со табелите и релациите.

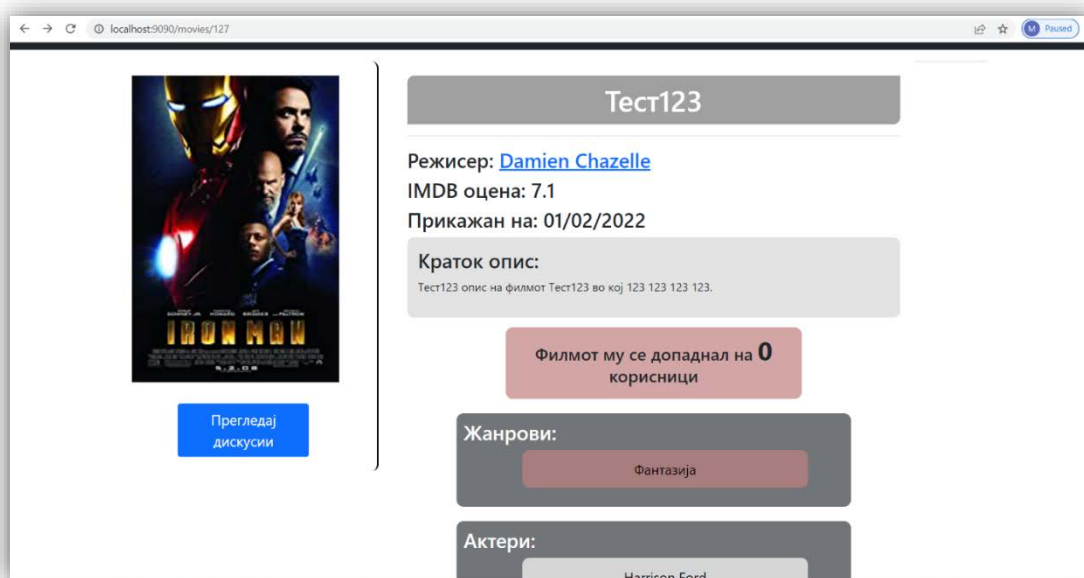
Форумот претставува **Spring boot** апликација каде се искористени функционалностите на **Thymeleaf**, **Spring security**, **hibernate** и **jQuery** за манипулација со **DOM** објектите и соодветните **ajax** повици.

Почетна страница на форумот е страницата со филмови каде се прикажани филмовите со страничење и опции за филтрирање на истите според наслов или жанр. При клик на даден филм, се истакнуваат информациите за дадениот филм.



Слика 4. Приказ на почетна страница со филмови

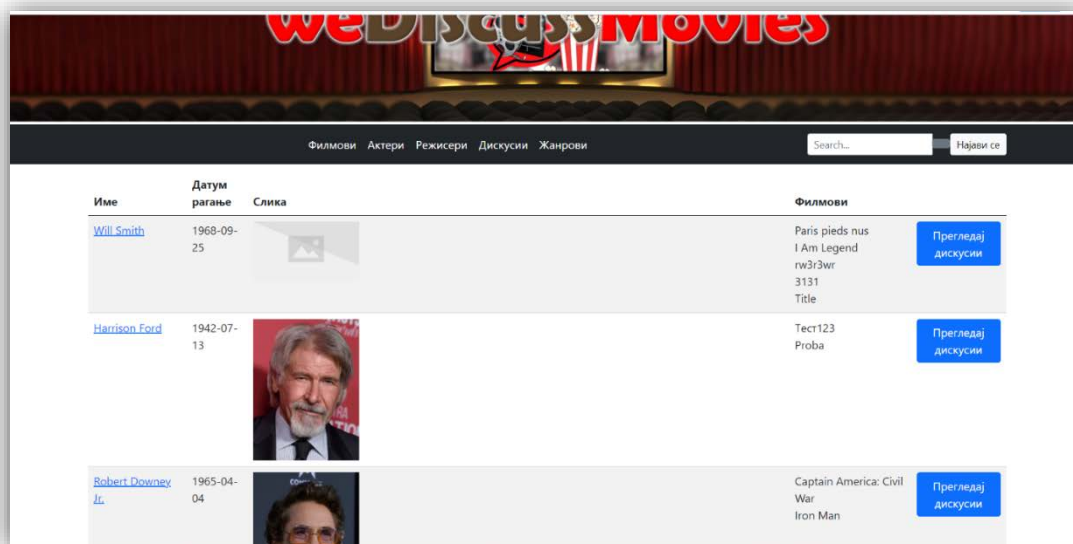




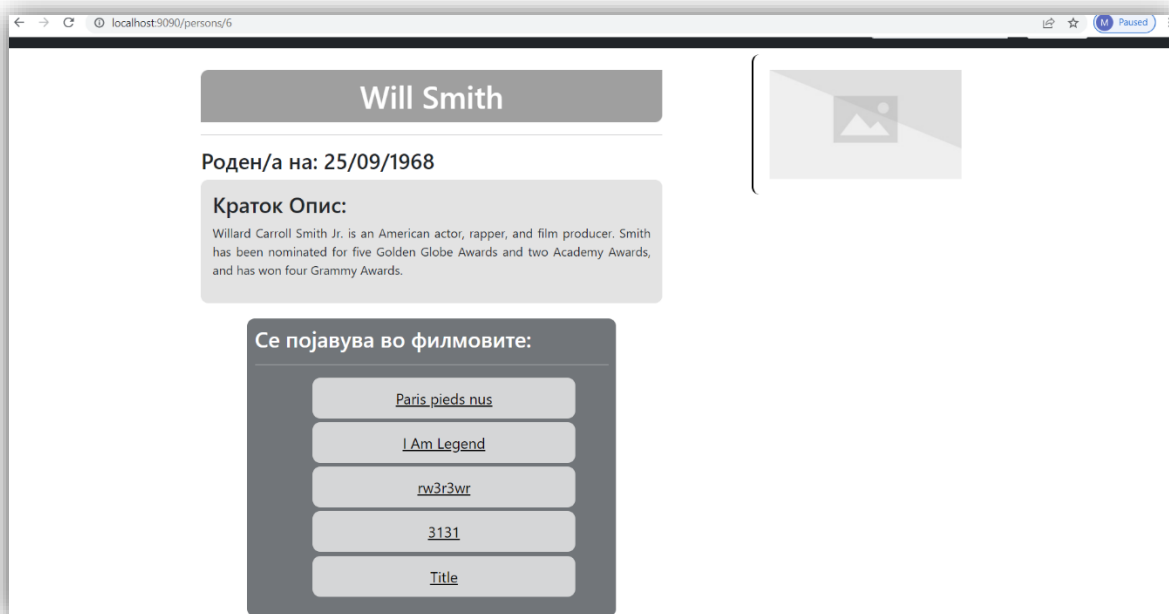
Слика 5. Приказ на дел од информациите на даден филм

Соодветно копчето Прегледај дискусии – го навибира корисникот до дискусиите за дадениот филм, доколку не постојат, при клик на копчето, се појавува празна страница.

Актерите се прикажани на Актери од навигациско мени, а режисерите на Режисери од навигациско мени. При клик на прегледај дискусии се прикажуваат дискусиите на дадениот актер или режисер. Форматот на приказ дискусиите и ист како и кај филмови. Името на актерот соодветно пренасочува кон приказ за дадениот актер. Сличен е приказот и на режисерите.

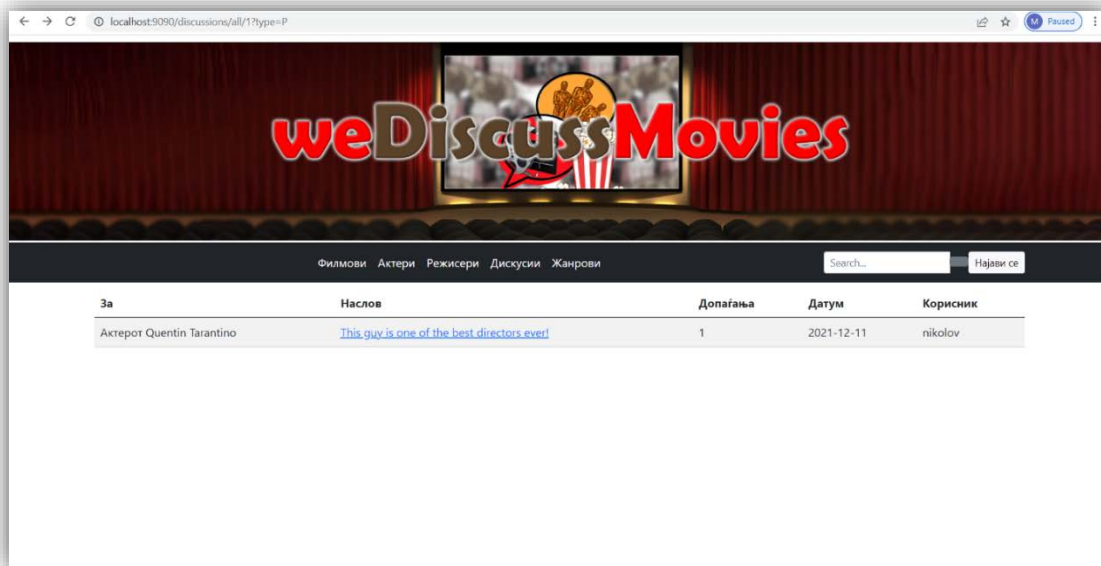


Слика 6. Приказ на актери



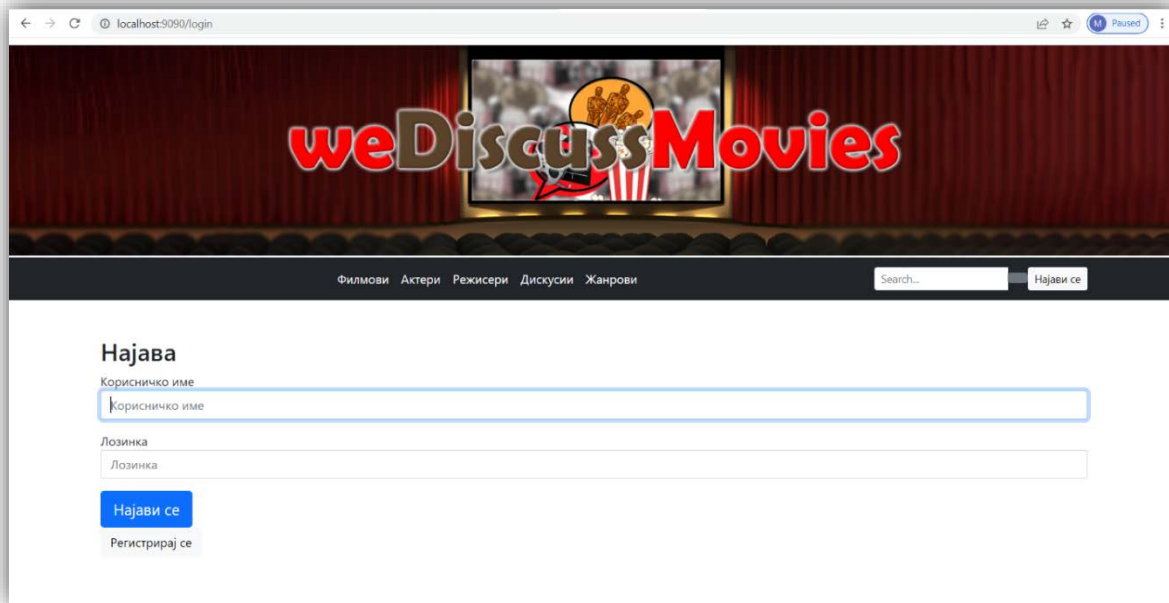
Слика 7. Приказ на дел од информации за даден актер

При клик на насловот на дискусијата се навибира на конкретен приказ на истата дискусија. Жанровите се прикажани со името на жанрот и соодветно бројот на допаѓања за истиот.



Слика 8. Приказ дискусии за даден режисер

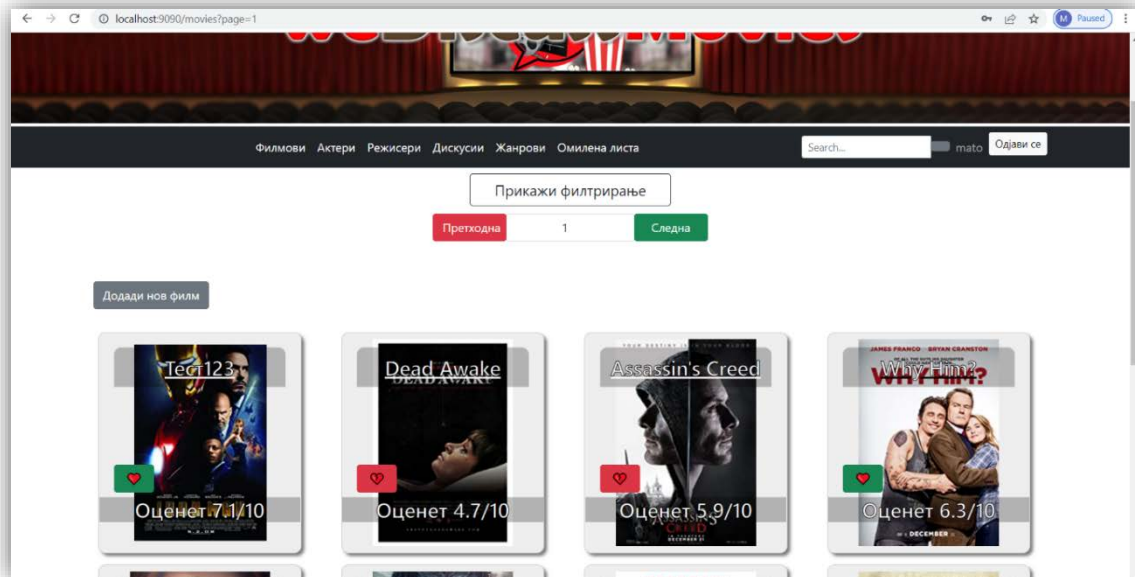
При клик на копчето Најави се, се прикажува соодветно формата за најава на корисник со корисничко име и лозинка, како и опцијата за регистрација. Доколку има проблем со автентикација на корисникот, се појавува известување дека не е успешна најавата. По успешна најава, корисникот се насочува кон страницата со приказ на филмовите.



The screenshot shows a web browser window with the address bar displaying 'localhost:9090/login'. The website's header features a dark background with the 'weDiscussMovies' logo in the center, which includes a movie screen and popcorn. Navigation links for 'Филмови', 'Актери', 'Режисери', 'Дискусии', and 'Жанрови' are on the left, and a search bar and a 'Најави се' button are on the right. The main content area is white and titled 'Најава'. It contains two input fields: 'Корисничко име' (Username) and 'Лозинка' (Password). Below these fields are two buttons: a blue 'Најави се' (Login) button and a grey 'Регистрирај се' (Register) button.

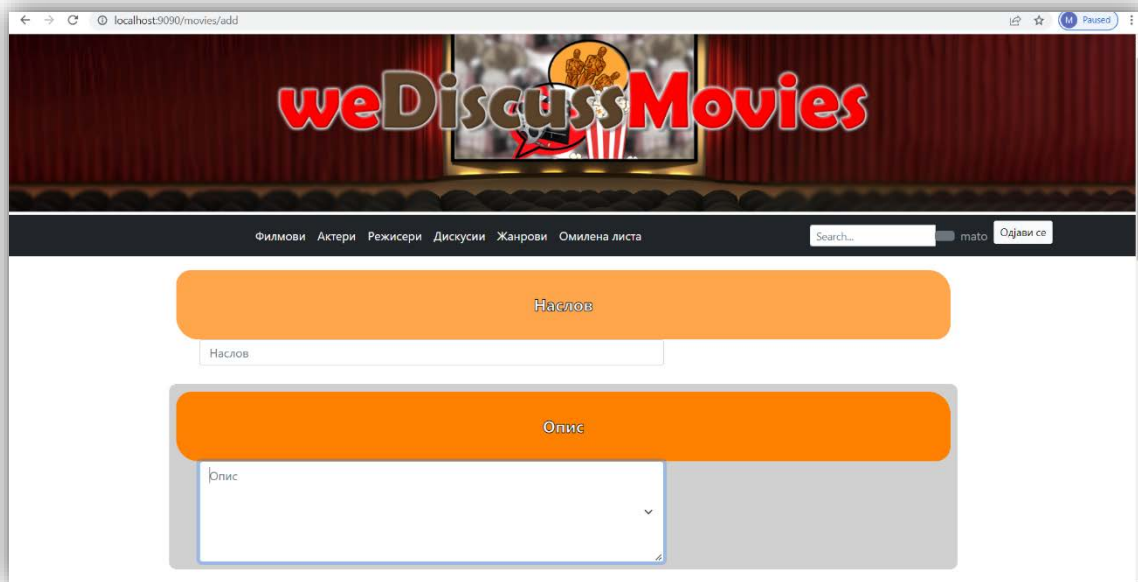
Слика 9. Приказ на форма за најава

Досега приказите на погледите на страниците беа кога корисникот е ненајавен и истите прикажуваат само дел од страниците. Кога корисникот ќе се најави, се прикажуваат и копчињата за соодветните **CRUD** и останатите операции кои се дозволени за записите од соодветните табели. Иконата срце со зелена позадина означува дека корисникот го нема допаднато филмот, додека иконата скршено срце со црвена позадина, посочува дека корисникот му се има допаднато филмот. При клик на иконите, се додава или брише допаѓањето за дадениот ентитет и иконите се заменуваат една со друга. Се појавува и соодветно копче Додади нов филм, што корисникот го насочува кон формата за додавање на филм. Дополнително, во навигациското мени се додаде референца кон филмовите кои се во омилена листа на корисникот преку Омилена листа.



Слика 10. Приказ на страницата филмови кога корисник е автентичиран

Формата за додавање на филм ги содржи полињата со кои се репрезентираат соодветните атрибути на филмот. Најдолу се прикажани копчињата за додавање и откажување на акцијата додавање и пренасочување кон страницата филмови. При успешно додавање или откажување, корисникот се враќа на страницата филмови. Додадениот филм се наоѓа на последната страница од страничењето бидејќи на серверска страна со секоја страница се земаат по 12 филмови секвенцијално.



Слика 11. Приказ на форма за додавање на филм

Страничењето е реализирано така што се земаат сите идентификатори на филмовите и се земаат оние чијшто идентификатор се наоѓа во интервалот претходна страница \* 12 и сегашна страница \* 12. Поставени се и валидации во кодот за страницата да биде во дозволениот ранг.

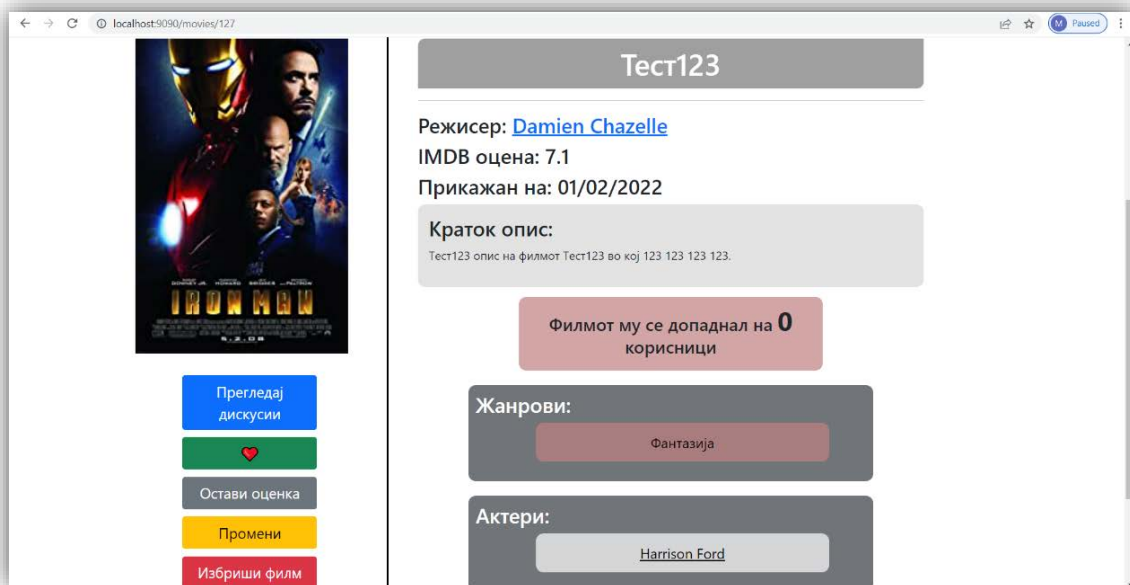
```
public class PageFrontMovies {
    public static List<Movie> getPagedMovies(String page, MovieService movieService, Model model){
        List<Integer> movieIds = movieService.listAllIds();
        List<Movie> movies = new ArrayList<>();
        if(page==null || Integer.parseInt(page) <= 0)
            page="1";
        int pageToLoad = Integer.parseInt(page);

        int from = (pageToLoad-1)*12;
        int to = pageToLoad*12;

        if (from>movieIds.size()){
            page = "1";
            pageToLoad = Integer.parseInt(page);
            from = (pageToLoad-1)*12;
            to = pageToLoad*12;
        }
        else if(to>movieIds.size()){
            to = movieIds.size();
        }
        movieIds = movieIds.subList(from, to);

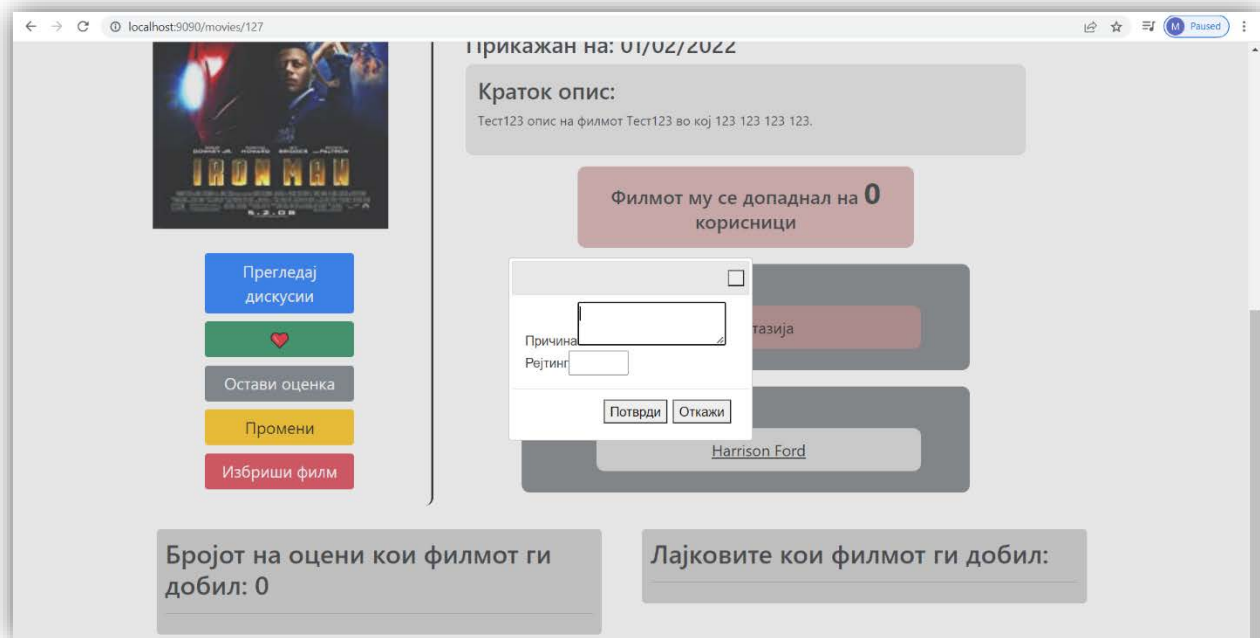
        for(Integer id: movieIds){
            //Trying to improve load times by not pulling all things from the database, excluding un-needed stuff
            //
            //movies.add(movieService.findById(id));
            movies.add(movieService.findById(id));
        }
        model.addAttribute( attributeNamed: "page", page);
        return movies;
    }
}
```

Слика 12. Приказ на кодот за страничење



Слика 13. Приказ на даден филм кога корисник е најавен

Иконата со срце ја има истата намена како и претходно. Кликот на Остави оцена, отвара модален прозорец каде го корисникот е потребно да внесе рејтинг и причина зошто го остава тој рејтинг. При успешно оставање рејтинг се менува копчето Остави оцена и соодветно се зголемува бројот на оцени на филмот.

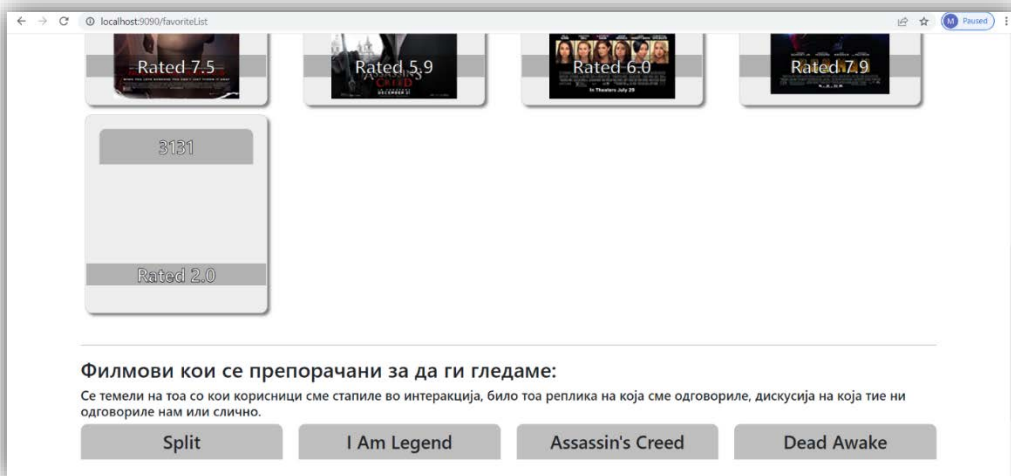


Слика 14. Приказ на модален прозорец за додавање на рејтинг и причина за филмот

Соодветно формите за додавање на актер и режисер, како и нивно уредување, имаат сличен приказ со нивните атрибути кои се потребни, но и акциите за оцена и бришење се со истите функционалности како и кај филмот.

При преглед на филмови од омилена листа на даден корисник, на крај се дадени филмовите кои се предлог врз база на корисниците со кои стапил во интеракција во последните 2 месеци и корисници кои имаат оставено број на реплики поголем од просечниот број на сите реплики. Имено, според тие корисници, се добиваат најдобро оценетите филмови од табелата **movie\_rates**. Се преземаат само насловите на филмовите и прашалникот прима го прима параметарот што претставува конкретниот идентификатор за дадениот корисник, на кого ќе се предложуваат филмовите. **MovieSuggest** претставува интерфејс во кој е енкапсулиран самиот наслов на филмот што се враќа.

Слика 15. Приказ на форма за уредување на веќе постоечки филм



Слика 16. Прашалник за побарување на предлог филмови

```

@Query(nativeQuery = true,value = "select title from(\n" +
    "\t select m.title, sum(mv.starsRated) as total\n" +
    "\t from project.replies r \n" +
    "\t join project.discussions d on r.discussion_id = d.discussion_id\n" +
    "\t join project.replies r2 on r2.discussion_id = d.discussion_id and r2.user_id != ?1 \n" +
    "\t join project.users u on u.user_id = r2.user_id and \n" +
    "\t \n" +
    "\t \t \t \n" +
    "\t \t \t select count(r3.reply_id)\n" +
    "\t \t \t from project.replies r3 \n" +
    "\t \t \t group by u.user_id \n" +
    "\t \t \t ) \n" +
    "\t \t \t select (cast(count(*) as float )) / (select cast(count(*) as float) from project.users)\n" +
    "\t \t \t from project.replies \n" +
    "\t \t \t ) \n" +
    "\t \t \n" +
    "\t join project.movie_rates mv on mv.user_id = u.user_id \n" +
    "\t join project.movies m on m.movie_id = mv.movie_id\n" +
    "\t where r.user_id = ?1 and \n" +
    "\t r.date between current_date - interval '2 months' and current_date\n" +
    "\t group by m.title\n" +
    "\t order by sum(mv.starsRated) desc \n" +
    "\t limit 15\n" +
    "\t ) as tabela")
List<MovieSuggest> proposeMovie(Integer id);

```

Слика 17. Приказ на дел од страницата за омилена листа на даден корисник



## Имплементација на GraphQL во проектот

Во проектот **GraphQL** е имплементирано со помош на анотации. Како што споменато погоре, истото е направено, со цел да се запази форматот на почист и поконцизен код, што нема до води до поголем број грешки на апликацијата. Во секој од моделите што репрезентира даден ентитет од базата, е искористена анотацијата **@GraphQLQuery** на секое својство на моделот. Имено, оваа анотација прима два параметри, име на својство како ќе се побарува и опис што истото значи. Дополнително, е употребена и анотацијата **@GraphQLNonNull** со која се посочува, дека даденото својство во моделот е задолжително односно мора да има вредност.

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Id
@Column(name = "discussion_id")
@GraphQLQuery(name = "id",description = "Идентификатор")
private int discussionId;
@Basic
@Column(name = "type")
@GraphQLNonNull
@GraphQLQuery(name = "type",description = "Тип")
private Character type;
@Basic
@Column(name = "text")
@GraphQLNonNull
@GraphQLQuery(name = "text",description = "Текст")
private String text;
@Basic
@Column(name = "title")
@GraphQLNonNull
@GraphQLQuery(name = "title",description = "Наслов")
private String title;
@Basic
@Column(name = "date")
@GraphQLNonNull
@GraphQLQuery(name = "date",description = "Датум на креирање")
private LocalDate date;
```

Слика 18. Приказ на искористени анотации за својства во моделот за дискусии



Во имплементациите на сервисните логики, искористени се анотациите `@GraphQLQuery` и `@GraphQLMutation`, заедно за аргументите кои ги прима методот преку анотација `@GraphQLArgument`, каде се наведува името на параметарот како да биде застапен во прашалникот. Анотацијата за мутација е искористена секаде онаму, каде има потреба за промена на податоците во базата односно за нивно додавање, бришење и уредување. Обичните прашалници само за преглед на податоци, се репрезентирани преку анотацијата за прашалник. Во прилог, е прикажан дел од имплементација на сервисниот модел за филм.

```
@Override
@Transactional
@GraphQLMutation(name = "editMovie")
public Movie edit(@GraphQLArgument(name = "id") Integer movieId,
                  @GraphQLArgument(name = "title") String title,
                  @GraphQLArgument(name = "description") String description,
                  @GraphQLArgument(name = "image") String imageUrl,
                  @GraphQLArgument(name = "airingDate") @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate airingDate,
                  @GraphQLArgument(name = "rating") Double rating,
                  @GraphQLArgument(name = "director") Integer directorId,
                  @GraphQLArgument(name = "actorIds") List<Integer> actorIds,
                  @GraphQLArgument(name = "genreIds") List<Integer> genreIds) {

    Movie movie = this.findById(movieId);
    Person director = getDirector(directorId);

    movie.setTitle(title);
    movie.setDescription(description);
    movie.setImageUrl(imageUrl);
    movie.setAiringDate(airingDate);
    movie.setImdbRating(rating);
    movie.setDirector(director);

    this.movieActorsRepository.deleteAllByMovie(movie);
    this.movieGenresRepository.deleteAllByMovie(movie);

    movie = this.movieRepository.save(movie);
    this.addActorsAndGenresForMovie(movie, actorIds, genreIds);

    return movie;
}
```

Слика 19. Приказ на методата за уредување на филм во сервисната логика

Методата е анотирана како трансакција со што, на менаџерот за трансакции, се укажува дека во рамките на овој метод, ќе бидат извршени повеќе промени во базата и истите да се извршат во дадената трансакција. Имено, најпрвин се пребарува во базата дали постои запис за идентификаторот проследен како прв аргумент, и доколку постои, се ажурираат соодветните нови вредности за дадениот филм, во спротивно, се фрла исклучок што означува дека филм со таков идентификатор не постои. Потоа, се бришат старите записи во табелите за актери и режисери и се додаваат новите записи за дадениот филм. Прикажани се соодветните анотации за **GraphQL** прашалници и форматот на датумот да биде во YYYY-MM-DD, што означува година, месец и ден.

Прикажано е и името на мутацијата како треба да биде напишана во прашалникот кога се повикува. Подолу, е прикажано повикот, од **JavaScript** датотеката како е направен повикот за мутацијата на уредување на даден филм.

За тестирање на **GraphQL** барањата, се направени посебни темплејти кои започнуваат со префиксот `test`. Направени се темплејти за прикажување на динамичка содржина за филмови и личности врз база на одбраните полиња на корисникот за тие податоци. Во овие темплејти, се имплементира исто барањата за автентикација и согласно, доколку корисник е автентизиран, се појавуваат соодветните копчиња за менување на состојбата во базата. Исто така, направени се и за додавање на филм и личност, каде соодветно, доколку станува збор за уредување, на почеток се земаат податоците за тој филм или личност со **GraphQL** барање. Се земаат и останатите податоци во овие датотеки со **GraphQL** барања односно жанровите, актерите, директорите и филмовите за внес во формите. Истите се мапирани на следниве урл <http://localhost:9090/graphql/person/actors> за актери, <http://localhost:9090/graphql/person/directors> за режисери, и за филмови <http://localhost:9090/graphql/movie> каде соодветно се прикажуваат полињата и каде корисникот одбира што сака да повлече од податоците. Напомена, најпрвин е потребно да се причека да се повлечат почетните податоци, па потоа, да се прави интеракција со избирање на филтрите за податоци бидејќи до тој момент, нема истите да функционираат од причина што се влечат податоците и DOM моделот не е креиран целосно. Друга напомена, која треба да се спомне е дека, идентификаторот е потребен за сите операции за менување на состојбата, доколку истиот не е повлечен, нема да може да се извршат тие операции. Доколку некои податоци не се повлечени, на некои места ќе се покажува дека истите не се дефинирани.

Прашалникот за мутација започнува со клучниот збор **mutation** и потоа во телото на овој објект се проследува името на мутацијата, каде во заграда, се проследуваат како парови клуч-вредност одделени со `:`, се пишуваат аргументите кои се потребни за функционалност на таа операција. Соодветно, има дополнителни проверки за тоа дали корисникот внел податоци во формата, со цел, прашалникот да се конструира врз основа на она што корисникот, побарал да биде внесено. На крај, во зависност од тоа дали станува збор за додавање или уредување, се проследува барањето до единствениот урл задолжен за овие **GraphQL** барања.

```

let query = 'mutation {'
if (id)
  query += ' editMovie( id: ' + $(".granka").attr( name: "granka") + " , "
else
  query += ' saveMovie('

query = query+"title: \"" + title + "\" , description: \"" + description + "\" , airingDate: \""+ date
+"\" , image: \""+ image + "\" , rating: " + rating

if (director.length > 0)
  query+=" , director: " + director
if (actors.length > 0)
  query+=" , " + " actorIds: [" + actors + "]"
if (genres.length > 0)
  query+=" , genreIds: [" + genres + "]"
query+=" ) { movieId title } }"
console.log(JSON.stringify( value: {query:query}))
if (!id)
  callAjax(query, type: "add", text: "додаден!")
else
  callAjax(query, type: "add", text: "променет!")

```

Слика 20. Приказ на градење на прашалник за мутација на филм

```

function callAjax(query,type,text){
  $.ajax( url: {
    url: "/graphql",
    content: "application/json",
    data: JSON.stringify( value: {
      query:query
    } ),
    contentType: "application/json",
    method: "POST",
    accept: "application/json",
    success:function (data){
      console.log(data)
      console.log("ajaxSucess")
      if (type === 'display')
        doneDisplay(data)
      else if (type === 'add') {
        if (data.data.saveMovie)
          add(data.data.saveMovie, text)
        else
          add(data.data.editMovie, text)
      }
      else if (type === 'edit')
        edit(data.data.movie)
    }
  })
}

```

Слика 21. Приказ на Ajax повик за GraphQL барања

Искористена е предноста на **GraphQL**, па така кога, се уредува филм или личност, во едно исто барање се повлекуваат и сите филмови, режисери и актери во зависност од формата за додавање.

```
function displayInfoMovie(id) {  
  console.log(id)  
  let q = '{ movie(id: ' + id + '){ movieId title description imageUrl airingDate' +  
    ' imdbRating actors{ person{ personId name surname } } director{ personId name surname }' +  
    ' genres{ genre{ genreType genreId } } }'  
  q+=' actors{ \n' +  
    '   personId name surname } \n' +  
    '   directors{ personId name surname } \n' +  
    '   genres{ \n' +  
    '     genreId genreType } }'  
  console.log(q)  
  callAjax(q, type: 'edit')  
}
```

Слика 22. Приказ на **GraphQL** прашалник за филм и различни домени на податоци

Од приказот на слика 22, може да се увиде, дека покрај што се влечат податоците за дадениот филм, се побаруваат и податоците за сите жанрови, актери и режисери.

```
function displayBody() {  
  callAjax( query: '{ actors{ '+  
    ' personId name surname } '+  
    'directors{ personId name surname }' +  
    'genres{ ' +  
    ' genreId genreType } ' +  
    '}', type: 'display')  
}
```

Слика 23. Приказ на **GraphQL** прашалник за личност и различни домени на податоци

Доколку станува збор само за додавање, кога се вчитува формата, се праќаат барањата за различните домени, без идентификатор за филм.

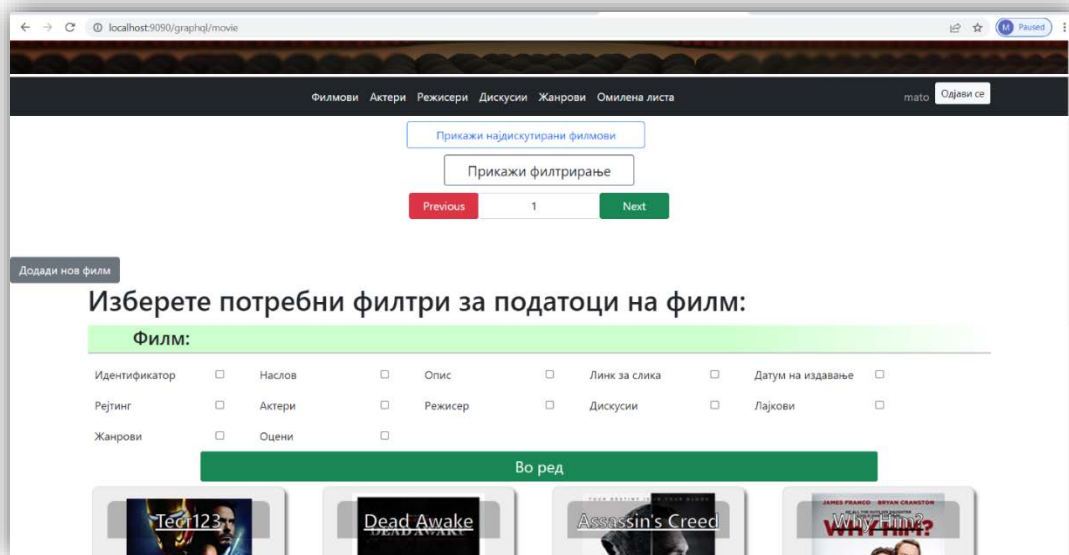
```

if (typePerson == 'D') {
  q = '{ person(id: ' + id + '){ personId name surname imageUrl dateOfBirth type description movies{ movieId title } }'
  q += ' moviesByType(type: \"D\"){ \n' +
    ' movieId title } }'
}
else {
  q = '{ person(id: ' + id + '){ personId name surname imageUrl dateOfBirth type description actors{ movie{ movieId title } } }'
  q += '\n moviesByType(type: \"A\"){ \n' +
    ' movieId title } }'
}
}

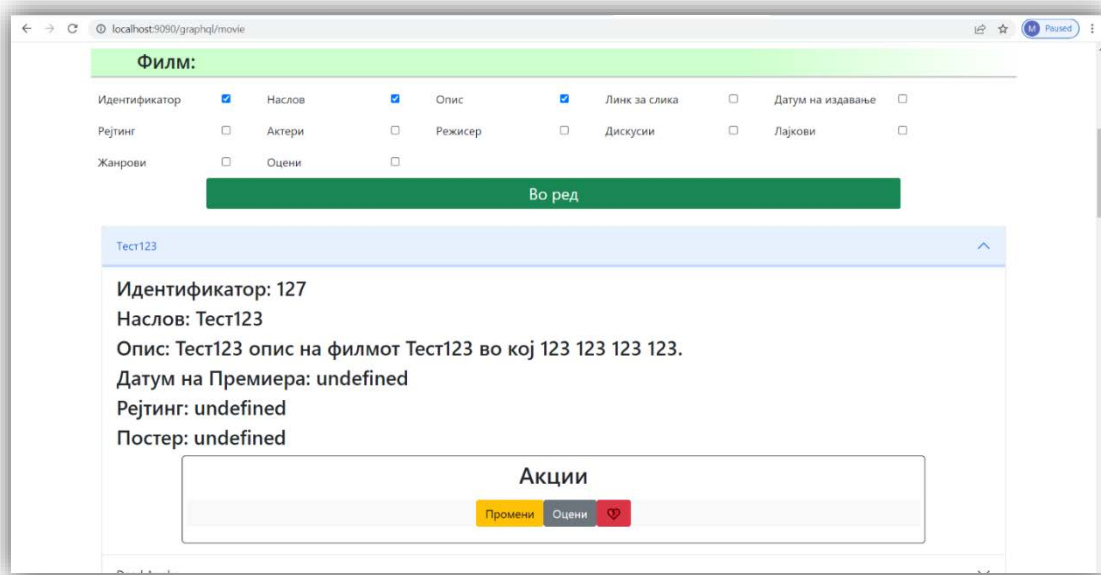
```

Слика 24. Приказ на **GraphQL** за додавање на филм

Ова е почетниот приказ на автентизиран корисник што би сакал да добие различна претстава на податоците за филм. Имено, потребно е посакуваните податоци да се штиклираат и при клик на копчето Во ред, истите ќе бидат поставени на местото на иницијалните филмови каде се сега поставени.



Слика 25. Приказ на страница за динамички **GraphQL** барања за филм



Слика 26. Приказ на дел од повлечените податоци за филм

Во делот прикажи Најдискутирани филмови, се покажува извештај за секоја година кој филм имал најголем збир на дискусии и реплики, поставени во таа година. Годишите ја репрезентираат секоја година кога имало поставено дискусија.

Најдискутирани филмови низ годините:

Година	Наслов	Бр. дискусии
2017	Paris pieds nus	1
2021	I Am Legend	12
2022	I Am Legend	10

Слика 27. Приказ на најдискутирани филмови

Прашалникот се извршува на тој начин, така што, најпрвин ги влече годините од дискусиите, кога биле поставени, па потоа, за секоја година ги спојува репликите и дискусиите и го зема оној филм, што имал најголем број на дискусии и реплики.

Во следниот дел, ќе се фокусираме на барањата кои се прават од страна на клиентот, или – што точно се прави кога ги селектираме сите оние полиња на сите ентитети кои би можеле да се содржат во базата и на клик на копчето, во случајот со идентификатор „okay-button“, при клик на коа копче, се повикува функцијата која ние сме ја дефинирале во фајлот <resources/static/js/graphqlmovie.js>

Функцијата која таму се повикува при клик на копчето, преку слушање на event, кое во позадина е имплементирано со помош на jQuery, библиотека за JavaScript.

Во следниот дел, ние ќе ви објасниме како се прави комплексното барање кое е детално барање на филмови, филтрирање на кои податоци ние (или клиентот) ги сакаме да ги добиеме и тие истите се влечат од некој „Rest Controller“ кој враќа податоци, тоа барање ние го испраќаме и потоа се справуваме со податоците кои ни се испраќаат од серверот.

При клик на копчето кое го наведов горе, се повикуваат следните делови на код, кои ќе ги објаснувам за секој код што се прави:

```
$("#okay-button").click(function () {  
  if ((!filters['actors'] &&  
    !filters['likes'] &&  
    !filters['genres'] &&  
    !filters['rates'] &&  
    !filters['discussions'] &&  
    !filters['director']) && (  
    !filters['movieId'] &&  
    !filters['movieDesc'] &&  
    !filters['movieTitle'] &&  
    !filters['movieImageUrl'] &&  
    !filters['movieAiringDate'] &&  
    !filters['imdbRating']  
  ))  
  return
```

Уште на почетокот на функцијата која лови за клик на копчето, ние веќе веднаш ги проверуваме атрибутите за кои ние треба да се погрижиме дека не се празни, бидејќи ако би биле празни, ние само би трошеле време на серверот и би давале барања, без да добиваме некои битни информации назад, т.е би добивале празни делови на JSON форматиран резултат – податоци.

Со цел да се избегне трошење на време и влегување длабоко во функцијата, ние ако видиме дека сите оние полиња се празни, веќе веднаш се враќаме од функцијата, и идеално би било да се известат клиентите што се случува со помош на некој си alert, а истото и се прави.

Наредно, доколку сеуште се наоѓаме во функцијата, ќе стигнеме до парчето код кое е поставено тука.

Целта на тоа делче код е да види кои дадени филтри се штиклирани, а кои не, при што – како што можете да видите на сликата, доколку филтерот е штиклиран, на пример во случајов да го земеме филтерот „movieId“ ако тој е штиклиран, тогаш во нашето квери или прашање, би сакале да го побараме тој податок, истото важи и за останатите атрибути кои еден ентитет Movie може да ги има (title, description, imageUrl, airingDate, imdbRating), тука се почетоците на градењето на кверито, и истото се продолжува и во наредните парчиња на функцијата.

```
let string = "{ movies{" + '\n'
if (filters['movieId'])
  string += "movieId" + '\n'
if (filters['movieTitle'])
  string += "title" + '\n'
if (filters['movieDesc'])
  string += "description" + '\n'
if (filters['movieImageUrl'])
  string += 'imageUrl' + '\n'
if (filters['movieAiringDate'])
  string += 'airingDate' + '\n'
if (filters['imdbRating'])
  string += 'imdbRating' + '\n'
```

```
if (filters['actors'] && (filters['actorId'] ||
  filters['actorName'] || filters['actorSurname']
  || filters['actorImage'] || filters['actorDesc']
  || filters['actorBirth'])) {
  string += 'actors{person{' + '\n'
  if (filters['actorId'])
    string += 'personId' + '\n'
  if (filters['actorName'])
    string += 'name' + '\n'
  if (filters['actorSurname'])
    string += 'surname' + '\n'
  if (filters['actorImage'])
    string += 'imageUrl' + '\n';
  if (filters['actorDesc'])
    string += 'description' + '\n'
  if (filters['actorBirth'])
    string += 'dateOfBirth' + '\n'
  string += '}' + '\n'
}
```

Во овој дел, соодветно гледаме дали филтрите кои би можеле да се полиња во една редица за ентитетот Person кој е во многу-многу релација со ентитетот Movie, опишан со релацијата movie\_actors, во овој ентитет, може да имаме податоци од типовите – (personId, name, surname, imageUrl, description, dateOfBirth).

Пред воопшто да провериме за секој атрибут филтер дали е селектиран, ќе видиме дали има воопшто некој атрибут поврзан со дадената релација кој е селектиран, бидејќи ако нема, не треба да имаме уште едно вгнездување, бидејќи како што може да видите, ако има атрибут барем еден кој е селектиран, се додава вгнездување.



Истото на некој начин се прави за ентитетот исто како претходно – Person, кој може да ги има истите атрибути – бидејќи е истиот тип на ентитет, само се разликува по вредноста која се наоѓа во полето type, која ние ја користиме за да разликуваме актери и режисери во нашата база на податоци.

Разликата тука е што еден филм може да има точно еден режисер или да нема режисер, при што можеме да референцираме кон редица во табелата Persons директно од редица во табелата Movies, т.е секоја редица од Movies референцира кон null или кон некоја постоечка редица од табелата Persons, ентитет кој има вредност на type еднаква на „D“ – Director.

```
if (filters['movieGenres']) {
    string += "genres{ genre{ genreType } }" + '\n'
}
```

Тука сме веќе стигнати до дискусиите на еден филм, па соодветно ако кликнато е копчето кое означува дека ние сакаме да имаме дискусии и ако имаме барем едно поле кое означува колона од табелата „Discussions“, ќе влеземе во овој дел на кодот, т.е ќе имаме вгнездување.

За секое поле кое може да е колона во Discussions, ние го внесуваме неговото име во нашето „query“.

Секоја дискусија може да има атрибути од типот на – (discussionId, text, date, title)

```
if (filters['director'] && (filters['director'] ||
    filters['directorName'] || filters['directorSurname']
    || filters['directorImage'] || filters['directorDesc']
    || filters['directorBirth'])) {
    string += 'director{' + '\n'

    if (filters['director'])
        string += 'personId' + '\n'
    if (filters['directorName'])
        string += 'name' + '\n'
    if (filters['directorSurname'])
        string += 'surname' + '\n'
    if (filters['directorImage'])
        string += 'imageUrl' + '\n';
    if (filters['directorDesc'])
        string += 'description' + '\n'
    if (filters['directorBirth'])
        string += 'dateOfBirth' + '\n'

    string += '}' + '\n'
}
```

Во ова кратко делче код, правиме само проверка дали клиентот на front-end селектирал дали сака жанрот да се прикаже во кој еден филм прираѓа.

```
if (filters['discussions'] && (filters['discussionId'] ||
    filters['discussionText'] || filters['discussionDate'] ||
    filters['discussionTitle'] || filters['discussionReplies']
    || filters['discussionUser'])) {
    string += 'discussions{' + '\n'
    if (filters['discussionId'])
        string += 'discussionId' + '\n'
    if (filters['discussionText'])
        string += 'text' + '\n'
    if (filters['discussionDate'])
        string += 'date' + '\n'
    if (filters['discussionTitle'])
        string += 'title' + '\n'
```

```

if (filters['discussionUser'] && (filters['discussion-userId'] ||
  filters['discussion-username'] || filters['discussion-userSurname'] ||
  filters['discussion-userUserName'])) {
  string += 'user{' + '\n'
  if (filters['discussion-userId'])
    string += 'userId' + '\n'
  if (filters['discussion-userUserName'])
    string += 'username' + '\n'
  if (filters['discussion-username'])
    string += 'name' + '\n'
  if (filters['discussion-userSurname'])
    string += 'surname' + '\n'
  string += '}' + '\n'
}

```

Тука, сеуште се наоѓаме во делот кој е наменет за влечење податоци на една дискусија, бидејќи секој ентитет „Discussions“ може да има врски со ентитетот „Replies“, секој „Discussion“ има нула или повеќе реплики, а секоја реплика има точно една дискусија на која одговара, што значи таа дискусија е референцирана директно во некоја колона на ентитетот, не ни треба релација.

Од секоја ентитет „Reply“, ние може да ги побараме следните податоци – (text, date).

А истовремено, секоја реплика е поставена од некој корисник, што значи дека може исто да ги побараме и информациите за корисникот, слично како и кај секоја дискусија, секоја - реплика си има точно еден корисник, при што знаеме дека можеме тука директно да се референцира корисникот во секоја редица во табелата Replies, во некоја дадена колона – userId.

```

if (filters['rates'] && (filters['rates-grade']
  || filters['rates-reason'] || filters['rates-show-user'])) {
  string += 'rates{' + '\n'
  if (filters['rates-grade'])
    string += 'starsRated' + '\n'
  if (filters['rates-reason'])
    string += 'reason' + '\n'
}

```

Во овој дел од кодот, ќе гледаме да извлечеме информации за корисникот кој ја поставил дадената дискусија во која ако сме влезени во овој дел од кодот, за неа сме извлекле барем еден дел од кодот.

Тука ќе се вадат податоци од ентитетот „Users“ кои може да има пристапливи податоци од типот на (userId, username, name, surname), а е во директна релација со ентитетот Discussions, бидејќи секоја дискусија мора да е поставена од некој корисник, и тој може да е точно еден корисник.

```

if (filters['discussionReplies'] && (filters['replyText']
  || filters['replyDate'] || filters['replyUser'])) {
  string += 'replies{'
  if (filters['replyText'])
    string += 'text' + '\n'
  if (filters['replyDate'])
    string += 'date' + '\n'
  if (filters['replyUser'] && (filters['replyUserId']
    || filters['replyUserName'])) {
    string += 'user{' + '\n'
    if (filters['replyUserId'])
      string += 'userId' + '\n'
    if (filters['replyUserName'])
      string += 'username' + '\n'
    string += '}'
  }
}

```

Исто, секој филм може да е оценет од многу корисници, а секоја оцена да има вредност која ќе преставува колку е оценет филмот, од кој е оценет филмот, и едно поле кое ќе биде причина за дадената оцена.

Од секој ентитет „Rates“, можеме да ги побараме податоците од колоните (starsRated, reason).

Како што спомнавме, секоја оцена мора да е од некој корисник, и исто како кај реплика и дискусија, може да се побараат податоците од ентитетот „Users“ - (userId, username, name, surname).

Доколку не е штиклирано копчето кое означува дека ние сакаме да добиеме информации за даден корисник или нема ни едно друго поле што е штиклирано, овој дел од кодот не се извршува.

```
if (filters['rates-show-user'] && (filters['rates-userId'] || filters['rates-userUserName'])) {
  string += 'user{' + '\n'
  if (filters['rates-userId'])
    string += 'userId' + '\n'
  if (filters['rates-userUserName'])
    string += 'username' + '\n'
  string += '}' + '\n'
}
```

```
string += '}' }'
ajaxCall( button: null, string, type: "fetch")
```

Откако ќе завршиме со сите парчиња кои горе беа наведени, ќе дојдеме до ова делче код, кое прво го затвора „query“-то со тие две загради, па прави повик кон функцијата `ajaxCall`, во која како што можете да видите, како еден од аргументите е нашето „query“, со типот, кој е поставен како трет аргумент, ние фактички кажуваме што точно сакаме да правиме кога ќе стигнеме до функцијата, т.е во овој случај ние сакаме да „fetch“-уваме податоци, и истото го ставаме како аргумент, со цел да се знае што да се прави со податоците кога истите и ќе пристигнат.

Тоа што се прави со податоците кога тие ќе пристигнат и кога ќе имаме „fetch“ како вредност на аргументот „type“ е повик на функцијата `displayData`, функција која има цел да се ослободи од старите податоци кои ни се прикажани во тип на картички, и да ни ги даде податоците кои сме ги одбрале, во случајов за поинтуитивен дизајн и прегледност, ние користиме „accordion“ шема, каде при клик на насловите на филмовите, бидејќи само насловот е видлив, ни се отвара поле под насловот, кое ги содржи сите вредности кои сме ги одбрале, и ако тие имале вредност.

Шемата „accordion“ е шема која е достапна од страна на „bootstrap“ библиотеката од верзија 5.0 нагоре, и што е најбитно е доста прегледна и лесна е за имплементација.

Тоа што се прави во функцијата која е задолжена за полнење на страната со податоците кои ние ги добиваме со горенаведената функција која се извршува во позадина, е следниот код:

```
for (let item of data.data.movies) {
  let tr = `<div class='accordion-item' style='overflow: hidden'></div>`

  let divHeader = `<h2 class='accordion-header' id='heading'+ item.movieId +'>` +
    `<button class='accordion-button collapsed' type='button' data-bs-toggle='collapse' ` +
    `data-bs-target='#collapse'+item.movieId+'` +
    `aria-expanded='false' aria-controls='collapse'+item.movieId+'>` +
    item.title +
    `</button>` +
    `</h2>`
  let divBody = `<div id='collapse'+ item.movieId +'` +
    `class='accordion-collapse collapse' ` +
    `aria-labelledby='heading'+ item.movieId +` +
    `data-bs-parent='#tbody-table'>` +
    `</div>`

  let divBodyAdd = `<div class='accordion-body'></div>`

  $(divBodyAdd).append(`<h3>Идентификатор: ` + item.movieId + `</h3>`)
  $(divBodyAdd).append(`<h3>Наслов: ` + item.title + `</h3>`)
  $(divBodyAdd).append(`<h3>Опис: ` + item.description + `</h3>`)
  $(divBodyAdd).append(`<h3>Датум на Премиера: ` + item.airingDate + `</h3>`)
  $(divBodyAdd).append(`<h3>Рејтинг: ` + item.imdbRating + `</h3>`)
  $(divBodyAdd).append(`<h3>Постер: ` + item.imageUrl + `</h3>`)
```

Најпрво, започнуваме со некој си циклус, кои ги итерира сите филмови кои сме ги добиле при повик на функцијата `ajaxCall`, за секој филм се креира некој HTML елемент – во случајов „div“, секој член на „accordion“ шемата во „bootstrap“ се состои од „header“ и „body“, „header“-от претставува она што е видливо кога ние не сме имале сеуште кликнато на филмот и не е табот отворен, најчесто насловот – како во нашиот случај насловот на филмот, а во елементот „body“ ни е поставено сето тоа кое што сакаме да го гледаме при клик на насловот и ширење на div елементот, во случајов, body во најосновно сценарио би се содржел од – (Идентификатор, Наслов, Опис, Датум на премиера, Рејтинг и Линк до Постер).

Деловите „header“ и „body“ исто треба да имаат соодветни HTML селектори кои нив ќе ги идентификуваат и ќе знае кодот како да се справи со клик на „header“, т.е кој таб треба да го отвори, од табовите кои во позадина само ни се скриени.

Сите овие елементи, еден по еден, по редоследот во кој што дошле се вметнуваат во елементот кој е обичен div елемент со идентификатор „tbody-table“, кој првично ги содржи сите картички на филмови, кога сеуште ние не сме направиле повик, при иницијален load на страната `/graphql/movies`.

## Заклучок

Како секој програмер, особено програмер кој е заинтересиран за бази на податоци, влечење на податоци или веб програмирање, било кој би се заинтересирал и за GraphQL ако би ги знаел придобивките на технологијата, знаењето дека можеме да добиеме се што сакаме во секој даден момент, дека корисникот може да добие се тоа што го интересира и што сака да го добие, дека не мора да чека дополнително време за да базата врати податоци за кои на корисникот воопшто не му е гајле и не е заинтересиран за нив е доста скапоцено, времето е скапоцено, бидејќи чекање за да се вчитаат податоците од страна на базата би било фрустрирачко за било кој, а особено за клиентите, кои од фрустрација би можеле да ја напуштат вашата апликација или страна, и никогаш повторно да не сакаат да се вратат.

Заради тоа, ние сметаме дека било кој што прави апликација треба да се заинтересира за тоа што може да се направи со цел да се направи клиентот да чека пократко време, GraphQL е една од опциите за која што знаеме и сме ја пробале, и можеме со сигурност да кажеме дека иако ни требаше извесно време за да ја имплементираме и дополнително време за да ја тестираме, имплементирањето на оваа технологија е со сигурност вредно, вреди да се направи, па и да се потрошат долги периоди на анализирање на базата и гледање во код, истото на крај ќе покаже резултати, резултати кои ќе ја направат вашата страна супериорна над оние страници кои не нудат такви функционалности, а истото ќе биде приметено од страна на корисниците, кои ако не искусиле вакво нешто на некоја друга страна претходно, не ќе сакаат да ја напуштат вашата страна и да заминат кај вашите можни конкуренти на пазарот.

Знаејќи дека ние имаме доминација врз нашите клиенти на пазарот е нешто што е многу клучно во тоа дали ние ќе имаме успешна страница или не, и сето тоа зависи од нешто толку мало како што е дел од секундата – а некогаш и повеќе секунди, кои корисникот би ги поминал чекајќи, чекајќи да нешто се случи наместо да листа по вашата веб страна или апликација.

Како што можеме да видиме по алгоритмите кои популарните социјални медиуми ги користат, гледаме дека корисниците сакаат да гледаат тоа што ги интересира, а истото може да се примени и тука – не мораме да нудиме податоци кои корисниците не ги сакаат, не мораме да ги силиме на чекање на добивање на сите оние (за нив) бескорисни податоци, треба да им дозволиме да го добиваат она што го сакаат, и ние да го поттикнеме истото, да им ставиме навика на корисниците да го посетуваат нашиот сајт, да им дадеме причина за истото.

Тоа можеме ние да го направиме со помош на оваа технологија, која можеме да ја искористиме за точно тоа што досега го говоревме – добивање на точно тоа што го сакаме, а за да го направиме тоа, тука ќе ни помогне точно оваа технологија - GraphQL.

Поради сето тоа, ние како тим, би му го препорачале на секој кој сака да ја подобри својата апликација, да проба да го имплементира ова API, со цел да проба да се издигне себеси или својата апликација барем малку погоре на маркетингот, иако можеби базата на некој од вас би била комплицирана, ние сметаме дека времето кое сте го потрошиле вие имплементирајќи ја оваа технологија вистина вреди, и е подобро вие да потрошите време работејќи, отколку вашите потенцијални клиенти и корисници – чекајќи, бидејќи придобивките се доста големи и може да ја направат вашата апликација нешто посебно, нешто кое вашите корисници досега не го сретнале, поради кое можеби би ве одбрале вас наспроти вашите конкуренти.