**Елаборат за семинарска работа по предметот Веб сервиси**

**Тема: Евиденција на гледани филмови и серии**

**Изработено од:**

**-Стефан Ќоропановски**

**-Филип Додевски**

**Листа на искористени dependencies и jars**

* JaxWs-Api – Jakarta EE API, неопходно за создавање на SOAP базирани веб сервиси
* JaxWs-Rt – Runtime за JaxWS интерфејсот
* Javax.WS.Rs-Api – Jakarta EE API, неопходно за создавање и нудење на спецификации за REST базирани веб сервиси
* Javax-Servlet-Api – Jakarta servlet, кој како софтверска компонента ги проширува можностите на deployment серверот, се имплементира како web container кој служи за хостирање на веб апликации во еден веб сервер
* Jersey-Container-Servlet – Dependency, кој нуди специфична имплементација на rest servlet интерфејс, базиран на jersey framework спецификации
* Jersey-Media-Moxy – Помошен dependency кој служи за едноставно и ефикасно претворање на java POJO класи во JSON
* Jackson-Databind – Dependency кој нуди data binding функционалности за генерални цели користејќи tree-model на Jackson-Data-Processor, изградено врз Streaming API и Jackson анотации за конфигурација
* Jackson-Dataformat-Xml – Екстензија на Jackson кој како компонента служи за пишување и читање на XML енкодирани податоци
* Org.Json – Мал dependency кој служи за парсирање на JSON документи во Java класи и обратно
* Jersery-Client – Dependency кој нуди неопходни класи, интерфејси и спецификации за правење на rest client кој би консумирал REST сервис чии request-и се сервисирани од Jersey servlet
* Sun.Xml.Ws-Servlet – Нуди готов сервлет кој служи за deployment на SOAP сервис, и опслужување на request-и насочени кон него

**Структура и организација на семинарската работа**

Најпрво и најосновно семинарската е поделена на 2 дела. Првиот дел го сочинува најголемиот дел од проектот, тој се наоѓа во папката FilmReviewsProject. Во себе го содржи web.xml кој како датотека служи за конфигурација за web serverot на кој ќе биде deploy-ната веб апликацијата. Потоа ја содржи имплементацијата на SOAP сервисот, Rest сервисот како и клиентскиот код за Rest сервисот и секако целиот BackEnd интерфејс каде се манипулира со податоците. Целиот овој проект и неговата структура се базира на Maven организација конкретно на WebApp archetype кој служи за едноставно организирање и изградба на веб сервиси. Другиот дел од проектот се наоѓа во папката FilmsSoapWsdl кој е посебен проект исто така со Maven организација базиран на архетипот QuickStart, каде преку командата wsimport со користење на wsdl за SOAP сервисот кој е дефиниран на URL-то: <http://localhost:8080/filmreview/soapapi?wsdl> успеавме да генерираме клиентски код за консумирање на нашиот SOAP сервис. Клиентскиот код за SOAP сервисот го направивме во посебен проект, бидејќи класите и директориумите кои се генерираат преку WSImport се многу обемни и редундантни и ќе создадеа хаос во нашиот веќе обемен и хаотичен главен проект.

**Структура на FilmReviewsProject**

Најпрво ќе започнам со .java класата PasswordHasher кој служи за хеширање и обезбедување на акаунтите на регистрираните корисници. Оваа класа содржи функции кои се користат за генерирање на обезбеден password за корисниците, користејќи го SHA-1 алгоритамот со големина на клуч од 256 бајти и 10000 алгоритамски итерации. Исто така се користи и salt за дополнително осигурување на passwordot, како salt се користи username-от на корисникот. Значи на лозинката која што ќе ја внеси корисникот при регистрирање му се конкатенира самото корисничко име и потоа овој текст се хешира со SHA-1 и резултатот од ова се чува во датабазата како корисничка лозинка. Во оваа класа има исто така и функција која се користи за верификација на корисничката лозинка при негово логирање со што повторно се хешира неговата лозинка претходно конкатенирана со корисничкото име и се споредува со оној хеш кој што претходно бил зачуван во датабазата. Потоа останатите ресурси и класи се распоредени во посебни директориуми, сите модели се чуваат во директориумот со име Models, клиентскиот код за REST сервисот во папката Clients, во Resources се наоѓаат сите главни компоненти и ресурси за REST сервисот заедно со custom Exceptions кои ние ги дефиниравме со цел подобро да го информираме корисникот за проблемите кои може да се јават при користење на самиот REST сервис, и ги заменуваат оние предефинирани exceptions кои ги пушта default-ната конфигурација на Jersey сервлетот. Слично во папката SoapResources се наоѓата имплементацијата и ресурсите на SOAP сервисот заедно со сите custom Exceptions дефинирани одделно за него. Конечно во папката Utilities се наоѓа податочниот BackEnd каде во позадина се манипулира со податоците односно се извлекуваат и зачувуваат во датабаза и се споредуваат со аргументи кои корисникот ги внесува и се враќа некаков податок како резултат на ваквата споредна назад до Resources и SoapResources класите.

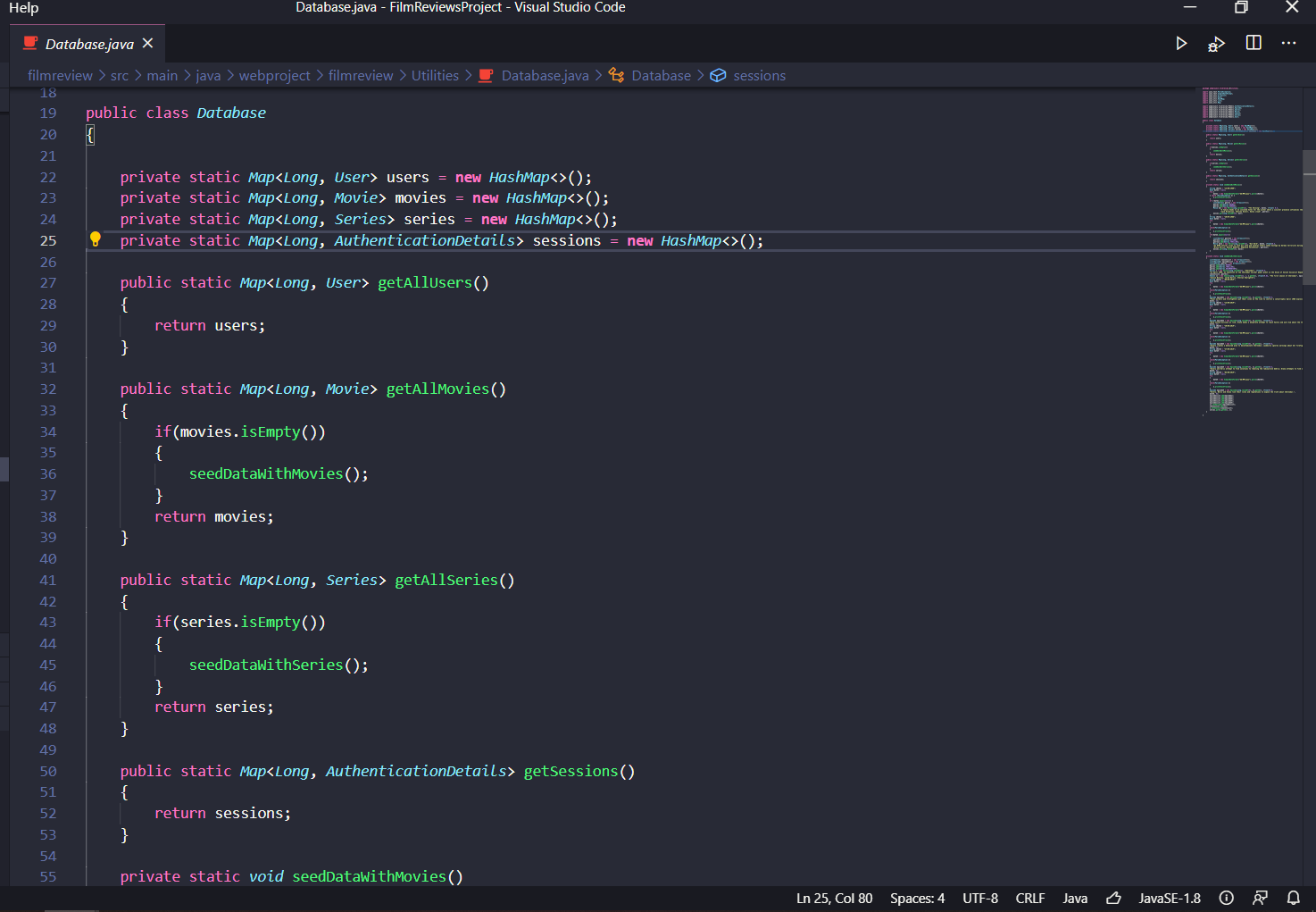
**Структура на FilmsSoapWsdl**

Во овој проект во папките webproject/filmreview/soapresources се наоѓаат сите модели, класи и ресурси генерирани преку WSImport, а во другиот дел на папки како што се filmsproject/soapwsdl се наоѓа клиентскиот код за консумирање на SOAP сервисот, каде имаме 4 главни класи. Едната е SoapMoviesClientGetter каде имаме дефинирано клиент кој прави GET барања до сервисот за филмови, друга е SoapMoviesClientPoster каде се прават POST, PUT и DELETE барања за филмовите. Потоа аналогно за серии ги имаме и SoapSeriesClientGetter и SoapSeriesClientPoster. Направени се посебни класи за get и за post/put/delete бидејќи имаше некакви error-и поврзани со имињата на импортите, односно дупликатни имиња.

**Управување и автентификација на корисници**

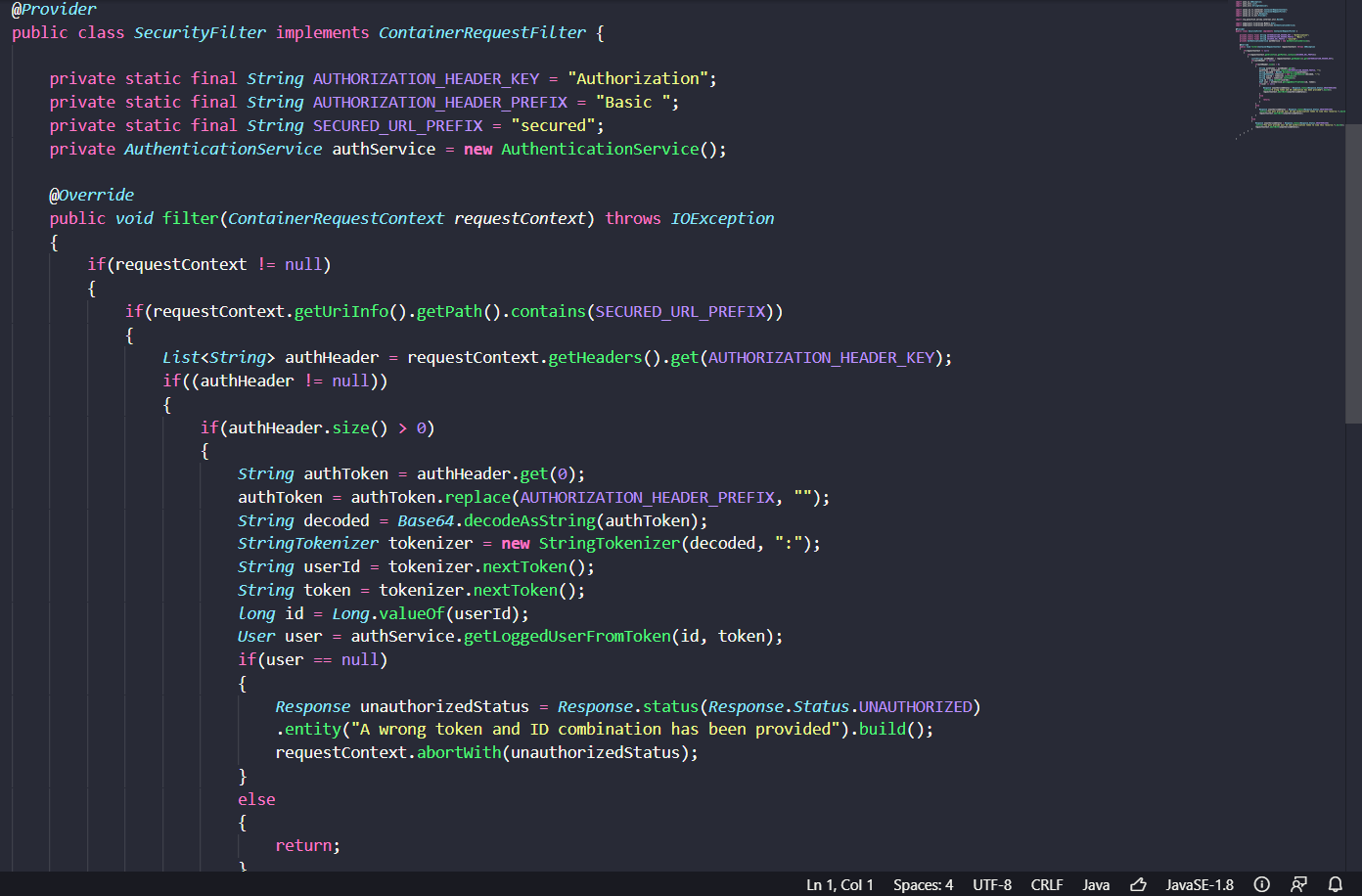
Самиот веб сервис кој што го направивме во себе содржи систем за регистрирање и логирање на корисници. За консумирање на SOAP сервисот нема потреба од автентификација ама, само за правење на GET барања, за останатите барања треба во барањето како аргумент да се понуди и администраторски token. Што значи тие барања не се наменети за обичен корисник освен, за читање на податоци. За Rest сервисот треба корисникот да биде регистриран и логиран, и тука може да прави секакви барања но, во тој сервис корисникот манипулира со податоци кои што му припѓаат нему односно прави листи на филмови/серии кои што сака да ги гледа и подоцна истите може да ги маркира како изгледани со внесување на датум на гледање. Во SOAP сервисот се прават записи за филмови и серии, и тоа се податоци кои што се достапни глобално во целата датабаза, но како што претходно напоменав обичен корисник може само да чита од нив и притоа не му ни треба автентификација за тоа. А администраторот кој си има свој token може да прави освен читање и запишување на податоци во тој сегмент.

За секој корисник се чува ID, корисничко име, лозинка (хеширана) и листа на филмови и листа на серии кои што ги има на своја листа за гледање ставено каде што истите може да стојат во состојба на неизгледани (ги има во план да ги гледа во иднина) или изгледани (внесен е датум на нивно гледање). Ресурсите и пристапните точки за логирање и регистрирање се управувани и менаџирани како како дел од REST сервисот, бидејќи за консумирање на истиот потребан е таквата автентификација. Главна точка каде што се спојуваат сите податоци со кои се работи целата семинарска односно записите и од SOAP делот и од REST делот е класата Database.java која се наоѓа во Utilities папката.



**Слика 1 – Приказ на Database класата**

Значи може да се воочи дека за чување на податоците користиме хеш мапи, една за регистрирани корисници, една за филмови, една за серии и уште една за сесии кои се создаваат при логирање. Овие хеш мапи имаат една точка од која потекнуваат и кога некој ресурс има потреба за управување со податоците во нив истите се повикуваат од оваа класа. Користењето на хеш мапа е згодно за нашиот веб сервис бидејќи откако сервисот е deploy-уван на веб сервер сите податоци кои се внесуваат си остануваат и се перзизтентни се додека веб сервисот е активен во веб серверот, откако веб серверот се исклучи сите промени кои што биле направени се бришат, но тоа е доволно за да ви го покажеме принципот на кој што работи нашиот веб сервис и општо концептите врз кои што е граден, користењето на вистинска SQL база е секако неопходно кај комерцијален веб сервис, но за нашиот случај каде што треба само да ви покажеме колку знаеме и умееме да работиме со веб сервиси овој начин на чување на податоци е потполно доволен и не прави вистинска разлика. REST ресурсите каде што е дефинирано url-то и кодот за комуницирање со корисници каде што се управува со нивно логирање и регистрирање, а секако и нивно одлогирање се токму соодветно класите RegisterResource.java, LoginResource.java и LogoutResource.java кои се дел од папката Resources. A backend кодот на чија што подршка зависат овие ресурси е класата AuthenticationService.java која што се инстанција во секој од овие ресурси и се наоѓа во Utilities папката, а која што пак директно ги зема хеш мапите од претходно спомнатата Database.java. Во овој backend код AuthenticationService се регистрираат корисници (притоа се прават проверки и дали некое корисничко име е зафатено или не), се логираат корисници (се прават сесии, а претходно се проверува дали е регистриран корисникот) и се одлогриаат корисници (се бришат сесии, претходно водејќи сметка дека истите тие сесии биле веќе запишани во датабазата). Системот за логирање кој што го имаме дизајнирано се базира на издавање на токени. Односно секоја сесија се состои од ID на корисникот и токенот кој што му бил издаден при логирањето (се враќа во body-то на response-от). Значи во овој бекенд код има функции за генерирање на токен, проверка дали некој токен е валиден или не, бришење на токен, извлекување на логиран корисник преку токен и слично. Начинот на кој се генерира токенот е следен. Прво корисникот откако ќе внесе корисничко име и лозинка се проверува дали е регистриран, доколку е се извлекува неговиота хеширана лозинка од датабазата, потоа се извлекува неговото ID и истото се конкатенира со неговото корисничко име. Потоа хешираната лозинка повторно се хешира со SHA-1 алкоритамот со 256 бајтен клуч и 10000 итерации каде што сега како salt се користи конкатенацијата од ID-то и корисничкото име. Потоа овој хеш се енкодира со Base64 и резултатот од овој процес е всушност токенот на корисникот кој потоа заедно со неговиот ID се враќа назад како response, а во меѓувреме и се прави запис за сесијата во датабазата. Да напоменам дека за внес на погрешна лозинка и корисничко име секако имаме customized response кој што ќе му даде до знаење на корисникот што згрешил. Откако корисникот поминал низ овој автентификациски процес, може потоа да пристапи до оној дел од REST ресурсите кои се однесуваат за логирани корисници. За сите овие ресурси заедничко е тоа што во URL-то го содржат делот “/secured/\*”. Односно имаме дефинирано безбедносен филтер кој што ги обезбедува сите ресурси кои го содржат овој дел во нивното URL.

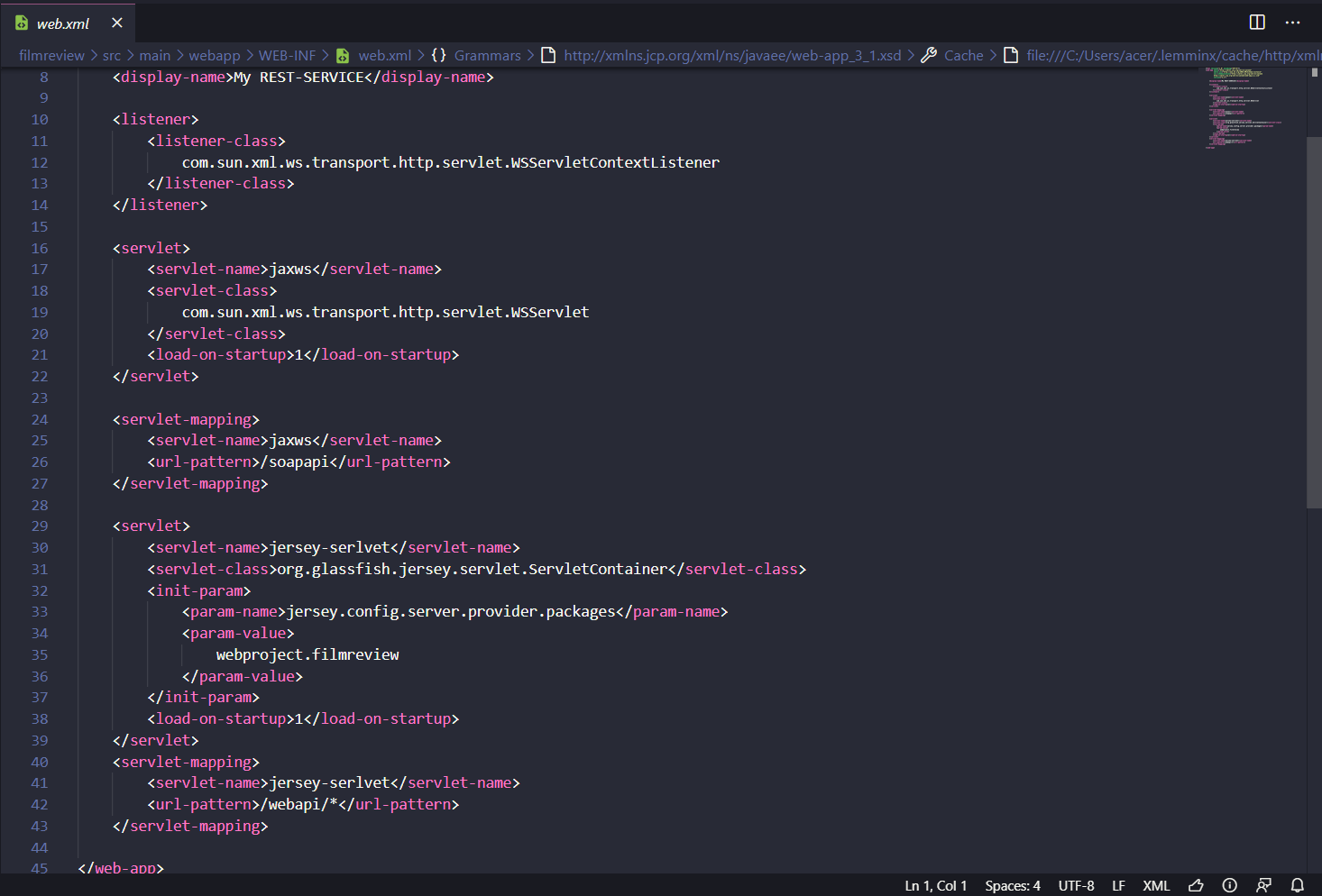


Слика 2 – Приказ на класата на безбедносниот филтер (се наоѓа во Resources папката)

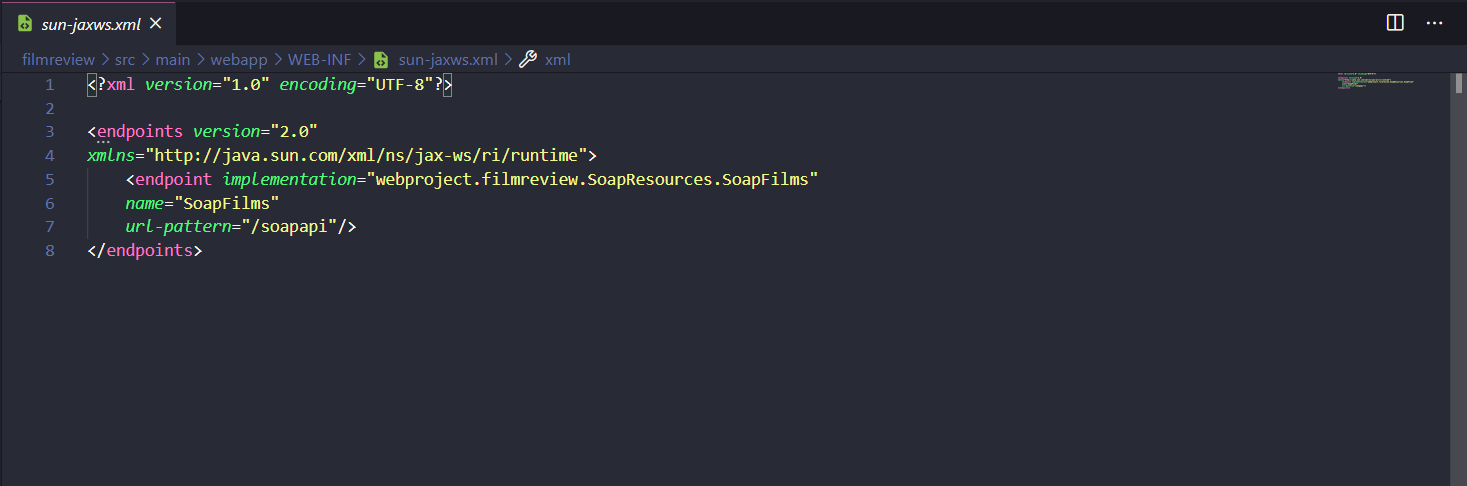
Од покажаното можеме да заклучиме дека откако ќе го добие корисникот неговиот токен, за сите request-и кои што ќе ги прави во останатиот дел од REST сервисот каде што ќе може да си управува со листи на филмови и серии ќе мора да додаде и Authorization Header од типот Basic Auth, каде што како username го пружи неговиот ID, а како password неговиот токен, доколку овие податоци не се валидни ќе добие NotAuthorizedException. Главниот дел на обезбедени rest ресурси се двете класи WatchTrackerMovieResource.java и WatchTrackerSeriesResource.java. Овие два ресурси имаат мноштво на функции во нив соодтветно за правење на GET, POST, PUT и DELETE request-и насочени кон веб сервисот. Секоја функција си има свој дефиниран url и метод комбинација. Во сите rest специфицирани функции во нашиот веб сервис функциите примаат податоци во JSON и враќаат податоци во JSON формат.

**Дефиниција на веб servlet-и во конфигурацискитите датотеки**

Пред да започнам да ви објаснувам како фунцкионираат rest ресурсите каде секој корисник си менаџира со своите листи на гледани филмови и серии, морам да посочам внимание на конфигурациските дадотеки web.xml и sun-jaxws.xml кои се наоѓаат во папката webapp/WEB-INF.



Слика – Приказ на web.xml дадотеката



Слика – Приказ на sun-jaxws.xml дадотеката

Како сервлет за интерцептирање и менаџирање со корисничките барања насочени кон нашиот rest сервис го користиме сервлетот со име jersey-servlet кој според самото име се гледа дека е имплементација на Jersey framework-от. Додека пак како servlet кој ќе ги опслужува барањата насочени кон нашиот SOAP сервис го користиме servlet-от со име jaxws. Дополнително за soap сервлетот имаме дефинирано и WS Servlet Context listener кој го парсиса помошниот конфигурациски фајл sun-jaxws.xml кој пак го нотифицира listener-от дека request-ите кои го содржат url-то /soapapi/\* мора да му бидат предадени и управувани на soap веб сервисот кој што се наоѓа во package-от со име webproject.filmreview.SoapResources и тоа е всушност јава класа со име SoapFilms која е анотирана како @WebService, на овој начин рутирањето е завршено. Додека пак за rest servlet-от кој што е готова Jersey имплементација, потребно е само да се специфицира во кој package се наоѓа jar фајлот и package-от каде што се наоѓаат сите rest ресурси со @Path анотација кои што во случајов се наоѓаат во package-от webproject.filmreview, а тоа всушност се сите јава класи во Resources папката. Со помош на ваквата конфигурација овозможено е истовремено преку .war фајлот да се deploy-ува и rest сервисот и soap сервисот на веб серверот Tomcat, со таа разлика што rest сервисот го содржи /webapi/\* во неговото url, а soap сервисот го содржи /soapapi/\*.

**Управување со листи за гледање на филмови**

Главниот ресурс за извршување на ваквата функционалност за правење на листи со гледани филмови и листи со филмови кои што се во план да бидат изгледани е токму WatchTrackerMovieResource.java. Додека пак бекенд кодот каде што се извршуваат позадинските операции поврзани со датабазата на кој што овој ресурс се потпира е токму јава класата WatchTrackerService.java која што заеднички ја користи овој ресурс, а исто така и аналогниот ресурс за серии WatchTrackerSeriesResource.java. Во ресурсот за филмови дефинирани се многу функции кои му стојат на располагање на корисникот за менаџирање на разно разни негови request-и. Првата е getWatchingMovies која што според името се знае е getRequest кој што враќа листа на филмови (изгледани и неизгледани) кои корисникот ги има ставено претходно во истата. Според QueryParam кои ги прима како параметри во URL-то оваа функција нуди и опција за филтрирање на филмовите и тоа истовремено може преку повеќе филтри доколку за сите е специфицирана некаква вредност што не е null од страна на корисникот. Доколку нема понудени параметри се враќа целата листа, доколку има се враќа листа на филмови кои ги задоволуваат условите зададени преку параметрите и тоа може според повеќе услови истовремено. Првиот параметар е contains, што значи филмот мора да ја содржи вредноста на овој параметар во неговото име, наредно е rating што значи филмот мора да има рејтинг поголема или еднаква на вредноста на овој параметар, потоа има genre што значи филмот мора во неговата листа на жанрови (кои се енумерација) да ја содржи вредноста на овој жанр. Моделот WatchesMovie е оној кој го сврзува корисникот со филмот кој што го гледа (изгледан и неизгледан), и преку него, односно листата од него која што ја има секој корисник дознаваме кој филмови ги гледа и враќаме листа на моделот Movie како одговор и тоа во JSON формат. Друга функција е getFinishedMovies, која што ги враќа сите филмови како листа кои корисникот ги има маркирано како изгледани. Потоа имаме getFinishedMovieById, кока враќа изгледан филм, но филтриран според неговото ID. GetWatchingMovieById враќа филм кој што не е маркиран како изгледан но, филтриран повторно според неговото ID. AddMovieToWatch е POST request кој што прима ID на филм и доколку претходно не бил истиот додаден на листата за гледање ќе го додаде, но доколку претходно бил додаден или истиот не постои ќе даде exception кој што ние лично го дефиниравме, со цел да ги избегнеме default-ните jersey exception-и кои не пружат конкретни информации за тоа што згрешил корисникот. Наредно го имаме setMovieAsFinished кој што е PUT request каде што корисникот дава како аргумент ID на филм и датум кој како String мора да биде во строг формат од типот dd/mm/yyyy, инаку во датабазата овој String се претвора во податок од тип Date и како таков се зачувува, инаку ќе добие exception. Истиот датум исто така мора да биде подоцна од самиот release date на филмот инаку ќе има повторно exception. На крај имаме и DELETE барање кое отсранува филм од листата на филмови за гледање доколку го има, ако го нема се враќа exception. Начинот преку кој препознаваме за кој корисник станува збор е помошната функција getUserIdFromHeader која што од Authorization header-от го извлекува ID-то на корисникот, а безбедносниот филтер претходно проверил дали неговата содржина е валидна. Додека пак начинот до кој пристапуваме до самото авторизациско заглавие е според @Context анотацијата која се става пред променлива од типот HttpHeaders кој што е променлива која стои како влезен аргумент во секоја од овие функции и потоа преку неа се доаѓа до било кое заглавие кое стои во request-от.

**Управување со листи за гледање на серии**

Аналогниот rest ресурс за оваа функционалност е WatchTrackerSeriesResource.java, каде што главниот бекенд кој му нуди подршка и функционалност на овој ресурс е повторно WatchTrackerService.java класата, затоа и истата е инстанцирана во овој ресурс. Бројот на функции во овој ресурс за разлика од оној за филмови е многу поголем и посложен бидејќи освен што се менаџираат сериите, се менаџираат и нивните сезони и епизоди. Бидејќи секоја серија си содржи листа на сезони секоја со свој ID и реден број, но секоја сезона си има и своја листа со епизоди кои исто така си имаат свој ID и реден број. Аналогно како и кај филмовите и тука имаме функција за земање на сите серии кои се ставени на watch list и тоа е функцијата getAllWatchingSeries, и има можност за истите да бидат филтрирани според тоа што содржи нивното име, според нивниот рејтинг и според нивниот жанр, значи ова е на ист принцип како и кај филмовите. Потоа имаме функција за земање на серија која е на листата за гледање но, исфилтрирано според нејзиниот ID, потоа функција за враќање на листа на сите изгледани серии, и функција за земање на изгледана серија според нејзиниот ID. Кога се враќа една серија истата се враќа заедно со сите нејзини сезони кои се ставени на листата за гледање, согласно со епизодите од тие сезони кои што се ставени на листата за гледање согласно до таму до кај што ги има ставено корисникот. Потоа имаме функција која ни враќа листа на сите изгледани сезони од дадена серија заедно со нивните епизоди, и исто така посебна функција за земање на сите изгледани епизоди од дадена сезона од дадена серија (како аргументи се става ID-то на серијата и сезоната) и функција која ни дава епизодa која е изгледана, секако исфилтрирана претходно според нејзиниот ID. Имињата на функциите не би ги посочувал бидејќи кога ќе го разгледате самиот фајл ќе заприметите по самите имиња на функциите и нивните аргументи како и нивниот деклариран метод веднаш може да се препознае што е тоа што функцијата го прави односно нивните имиња се интуитивни. Да напоменам дека од GET функциите преостанува да напоменам уште дека има функции кои земаат сезона заедно со сите нејзини епизоди според нејзиниот ID и епизода исфилтрирана според нејзиниот ID, ова се две одделни функции кои би вратиле соодветно сезона (изгледана или неизгледана) и епизода (изгледана или неизгледана). Наредно би продолжил со сите POST функции чија што улога секако е ставање на листата за гледање. Процедурата е следна пред да ставиме епизода на листата за гледање (како аргумент се пушта нејзиниот ID, ID на сезоната и ID на серијата) кое што е менаџирано со посебна функција, претходно мораме да ја имаме ставено сезоната на листата за гледање повторно посебна функција која го прима ID-то на сезоната и на серијата, но сега пак пред да ставиме сезона во листата за гледање најпрво и најосновно мора серијата да била ставена на листата за гледање секако при тоа имаме посебна функција која го прима ID-то на серијата. Доколку ваквиот редослед и процедура на повици не е запазен на корисникот ќе му се даде соодветен одговор во тип на exception или response кој што ќе му каже што има згрешено, заклучок е дека неможе да се стави сезона на листа за гледање без претходно да била серијата ставена во листата и неможе епизода да биде ставена во листата за гледање без сезоната прво да била ставена. Овој концепт е реализиран преку 3 соодветни POST функции. Редниот број на сезоните и епизодите во рамките на серијата немора да биде запазен при нивно ставање на листа за гледање, бидејќи сакаме да му дадеме слобода на корисникот пример доколку сака прво да ја изгледа некоја епизода која што не е првата епизода, може како таква да ја стави на листата за гледање таа друга епизода. Сличен е концептот на PUT request-и за ова имаме 3 функции повторно една за маркирање на епизода како изгледана, една за маркирање на сезона како изгледана и една за маркирање на серија како изгледана. Во секоја функција се пушта соодветно ID кое што се бара дали тоа би било на епизода, сезона или серија или на сите 3 истовремено или пак само на 2 од 3-те зависи до функцијата, друг аргумент е секако датумот на завршување кој исто како и кај филмовите се раководи според истиот концепт и облик, и доколку тој облик не е запазен корисникот ќе си добие соодветен одговор во облик на грешка. Тука оди обратен редослед од претходно, неможе една серија да биде маркирана како изгледана доколку сите нејзини сезони претходно не биле маркирани како изгледани, исто така неможе една сезона да биде маркирана како изгледана доколку сите нејзини епизоди не се маркирани како изгледани. Редоследот на маркирање на епизодите како изгледани не е битен но, датумот на нивно гледање мора да биде после датумот на нивно пуштање во јавноста на телевизија (release date), во спротивно ќе има грешка. Потоа датумот на маркирање на една сезона како изгледана мора да биде после датумот на гледање на неговата последно маркирана како изгледана епизода, и слично датумот на маркирање на една серија како изгледана мора да биде после датумот на маркирање на последната сезона која била маркирана како изгледана. Од GET функциите имам уште 2 како бонус, едната од нив ја враќа последната епизода од дадена серија која што е маркирана како изгледана според датумот на маркирање (getLastWatchedEpisodeFromSeriesByDate), а другата ја враќа последната епизода од дадена серија која што е маркирана како изгледана но, не според датумот на маркирање него според нејзниот реден број во рамките на сезоната (getLastWatchedEpisodeFromSeries). Во двете функции како аргумент се прима само ID-то на серијата. Конечно да напоменам дека имаме уште 3 функции и овие се дел од оние чиј метод е DELETE, односно имаме функција за бришење на една епизода од листа на гледање, потоа функција за бришење на сезона од листа на гледање и конечко за функција за бришење на серија од листа за гледање. При што корисникот ако сака директно да избрише од листа на гледање цела серија истата ќе биде избришана заедно со сите нејзини сезони, а слично и за сезона истата ќе се избриши заедно со сите нејзини епизоди. Ова беше отприлика накратко објаснето како работи овој REST сервис, тексуално опишано звучи хаотично но, при презентирање би била многу појасна големата слика или при читање на самиот код.

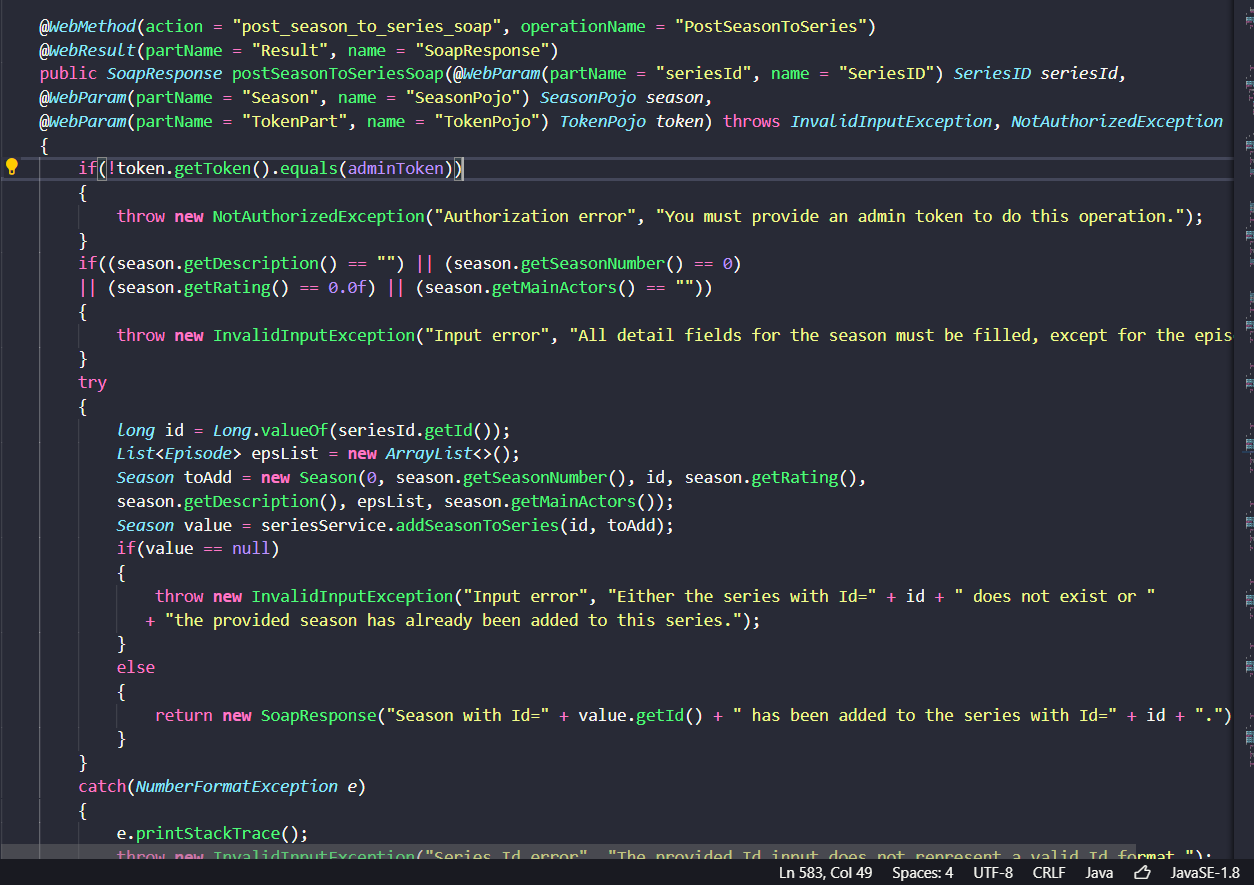
Слики не пуштив од овој дел на кодот конкретно бидејќи класите се многу обемни и во една слика едвај може една фукнција само да се стави. Можда пропуштив да спомнам дека самите exceptions кои може да се јават при повикот на ваквите функции се доста customized со посебен модел ErrorMessage кој се става како ентитет во response-от или пак чисто пуштање на некаков exception дали тој би бил некој како што се NotFoundException или NotAcceptableException кои што не се exception-и како дел од Tomcat него се дел од самото Jax-RS api со што е овозможенo поголемo нивнo прилагодување. Исто така околу формат на датумот бидејќи неговиот внес и формат е доста строг имам дефинирано и посебен модел на exception наречен DateFormatException кој наследува од RunTimeException кој што си има и свој DateFormatExceptionMapper кој што мора да ја има @Provider анотацијата слично како и безбедносниот филтер со цел Jersey сервлетот да знае како да пристапи до нив и да ги користи кога ќе се јави потреба од нив. Ова е методологијата која што мора да биде запазена за пуштање на прилагодени exceptions кај rest сервиси кои ја следат Jersey имплементацијата. За разлика од таму кај SOAP пуштање на прилагоден exception е многу поедноставно бидејќи немора да имаат свој Mapper, а исто така наследуваат од Exception наместо од RunTimeException. Но, недостаток кај обичен Exception е тоа што компајлерот бара од нас дополнително да реагираме врз нив односно се од типот на checked exceptions, кој што си имаат своја catch декларација.

**REST клиенти**

Клиентскиот код за консумирање на нашиот rest сервис се наоѓа во папката Clients и има посебен јава клиент за филмови и посебен јава клиент за серии. Клиентите се прилагодени според соодветната имплементација на самиот rest сервис односно според спецификациите на Jersey, порадо тоа се користат посебни dependencies и јава интерфејси за таа цел. Ставени се веќе соодветен број на функции во овие клиенти меѓутоа не сите функции кои што се на располагање бидејќи се многу на број, а ќе видите и дека самиот код за пишување на клиенти е многу обемен и редундантен и ставени се само некои поважни и функции колку да ви покажеме дека знаеме да ги пишуваме REST Jersey клиентите и да ги користиме класите и интерфејсите потребни за нивно пишување, како и додавање на заглавија во самите http барања. Во секоја класа има и декларација на main метод на кој откако ќе му стегнете Run во терминал ќе ви се изпише json одговорот конвертиран во обичен текст. Исто така однапред да знаете дека имаме направено и seedData методи во Database.java класата каде што иницијално датабазата се исполнува со 2 филмови, 1 серија со 1 сезона и 5 епизоди, како и 1 корисник со корисничко име zoran и лозинка 123, кој што е деклариран и внесен во AuthenticationService.java класата, овие иницијални податоци се користат за презентирање токму на горенаведените rest клиенти, каде што се претставени функции кои прават барања од сите 4 основни типови на http методи.

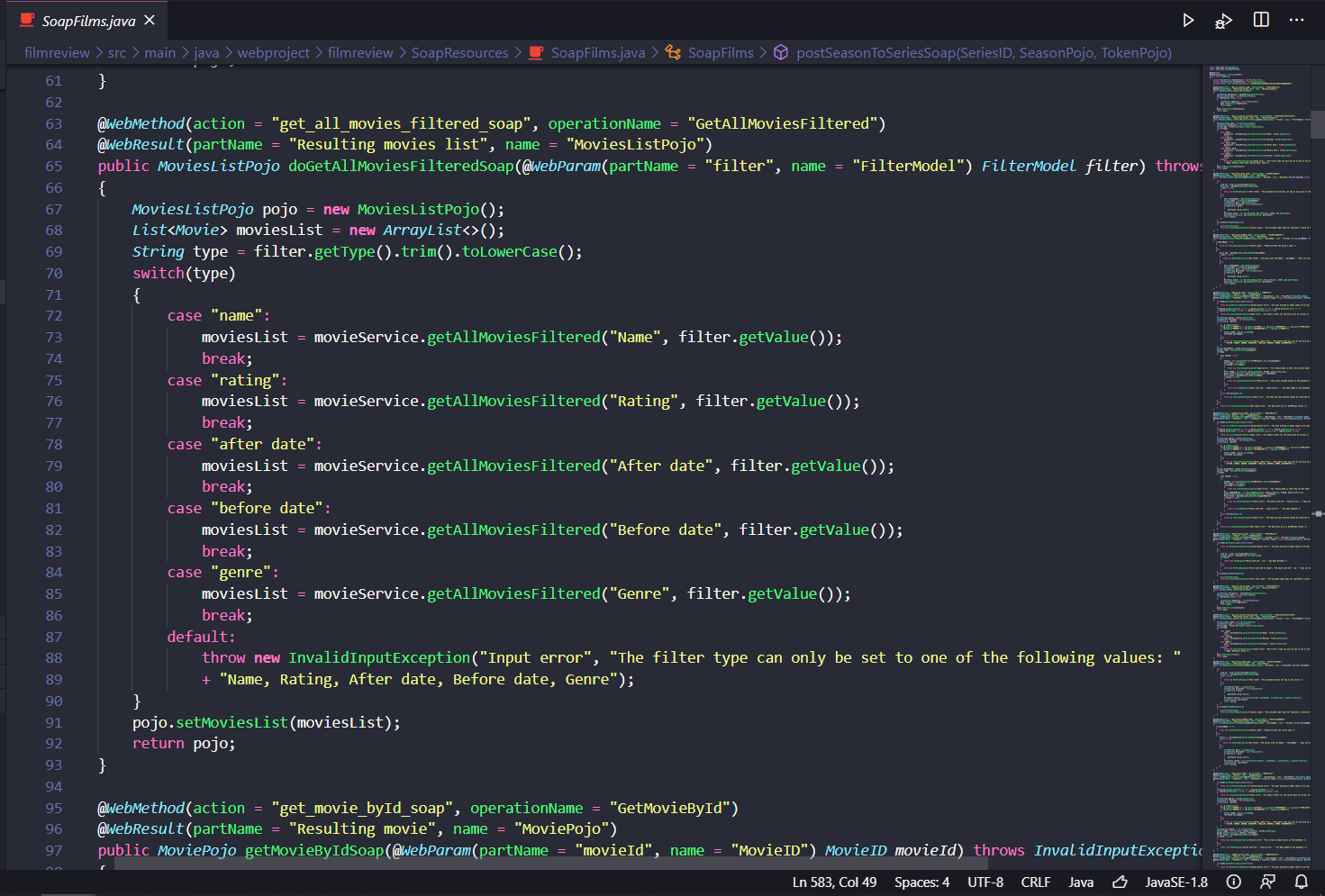
**Имплементација и структура на SOAP сервисот**

Имплементацијата на SOAP сервисот и сите негови помошни класи се наоѓаат во папката SoapResources. Самиот сервис е класата SoapFilms, но името на сервисот во неговиот wsdl стои како SoapFilmsService што е многу соодветно. Овој сервис се потпира на 2 одренени backend помошни класи каде се манипулира со податоците а тоа се соодветно MoviesService и SeriesService кои се дел од папката Utilities. Во овој сервис се користат доста голем број на Jax-B анотации за аргументите на методите, за резултатот што го враќаат аргументите, за самите методи, како и за самите модели кои што се специфично дефинирани за користење во овој сервис. Сето горенаведено е напревено со цел да има поголем customization на самиот wsdl и истиот да биде почитлив и разбирлив за обичен корисник. Ова лесно може да се воочи кога истиот би го тестирале на пример во SoapUI сите аргументи и response-и се доста интуитивни во нивната структура како и exception-ите кои што се враќаат при некоја грешка. Еве ви конкретни анотации кај еден од методите.



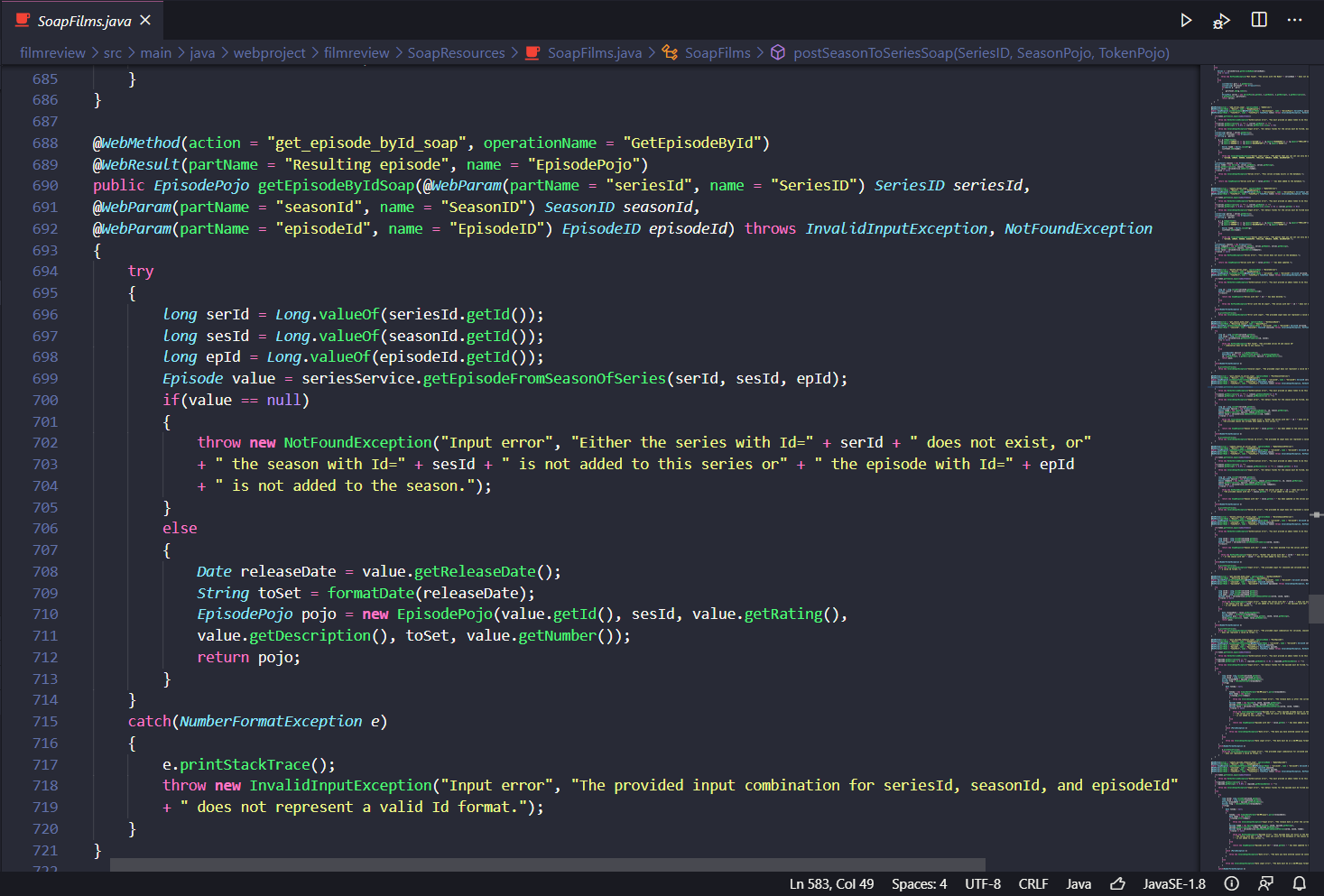
Слика 5 – Приказ на одреден метод од SOAPFilms сервисот (seriesService е инстанца на помошната бекенд класа SeriesService.java)

Во овој сервис сите методи кои се однесуваат на GET операција се достапни за било кој корисник па дури и за некој гостин кој не е автентифициран. Додека пак за сите операции поврзани со POST, PUT и DELETE операција потребено е да се како аргумент и администраторски токен. Од GET методите имаме метод кој ни ги враќа сите филмови во датабазата, метод кој ни враќа филм филтриран според неговото ID, метод кој ни враќа филм филтриран според неговото име и метод кој враќа листа на филмови филтрирани според еден филтер кој може да биде од типот филтер за текст кој треба во името да го содржи филмот, филтер за колкав да му биде минималниот рејтинг, филтер за жанр кој мора да го има во неговата листа на жанрови, филтер за release date да му биде пред некој датум и филтер за release date да му биде после некој датум.



Слика 6 – Приказ на методот за филтрирање на филмови (ист тип на метод има и за серии)

Потоа имаме метод за додавање на филм во датабазата, каде што пред да се додаде филмот се проверува дали истиот веќе го има во датабазата и дали податоците кои се пружат како аргумент се во валиден формат и доколку е се во ред ќе се додаде, ако не ќе има exception. Би ја ставил сликата од овој метод но нема целиот код да го собере во една слика. Потоа имаме и PUT метод за update на некој филм и на крај и delete метод за бришење на филмот. Во истиот фајл секако ги имаме и методите за управување со сериите, GET методи за земање на сите серии, метод за земање на листа на серии според филтер, метод за земање на една серија филтрирана според ID и метод за земање на серија според нејзиното име филтрирана. Потоа имаме метод за земање на сезона од серија и метод за земање на епизода од дадена сезона од дадена серија. Целиот клиентски код за soap сервисот се наоѓа во проектот FilmsSoapWsdl конкретно во папките filmsproject/soapwsdl, кои што пак се папки кои се наоѓаат во папките src/main/java соодветно.



Слика 7 – Приказ на методот за преземање на епизода од дадена сезона од дадена серија

Да напоменам кога се враќа одредена серија се враќа целата со се сите нејзини сезони и епизоди, а кога се враќа дадена сезона се враќа истата со се сите нејзини епизоди. Сега слично како и кај филмовите за секоја серија, сезона и епизода имаме соодветни POST, PUT и DELETE методи. Притоа кога се додава серија прво се додава серијата, и таа ќе биде празна без сезони, па потоа со посебен метод се додаваат сезони во таа серија, тие сезони ќе се на почеток празни без епизоди, па потоа со посебен метод се додаваат епизоди во тие сезони од одредени серии и така дабатазата се полни со податоци. Кога се прави PUT (update) на некоја серија може да се променат сите детали за серијата освен нејзините сезони. Затоа имаме посебен метод кој прави update на сезони и слично може сите детали да се променат освен листата со епизоди, и затоа на крај имаме посебен метод за update на епизоди, при што секако мора да се стават соодветни аргументи за ID. И уште ги имаме DELETE методите, имаме посебен метод за бришење на серија заедно со се сите нејзини сезони и епизоди, метод за бришење на сезона заедно со се сите нејзини епизоди и метод за бришење на една епизода. Да напоменам дека при додавање на сезона или епизода мора да се запаси и редниот број нејзин бидејќи ако во листата претходно се најде некоја сезона или епизода со ист реден број тогаш ќе се појави грешка и корисникот ќе биде известен. Ова е отприлика целиот SOAP сервис кој служи за полнење на базата со филмови и серии, како што претходно наведов, преку генерираниот wsdl, во посебен проект со користење на WSImport е генериран клиентски код за консумирање на SOAP сервисот. Во него има 4 класи секоја со свои методи за консумирање на сервисот и main програма на која кога ќе му направите run во терминал ќе ви се испише резултатот. Во овој клиентски код опфатени се сите методи кои ги има soap сервисот освен get методите за земање на листа на филмови и серии преку филтери, нема некоја специфичност кај тие методи едноставно сметавме дека немора да ги има, а секако кога ќе ви го презентираме проектот на Zoom сите методи кои што не се наведени во клиентскиот код, доколку сакате ќе ви ги претставиме преку SoapUI или Postman.