Developing, Deploying, Publishing and Finding Your Own Web Service – Proof of Working Code

Nicholas Cowling - 17072242

Contents

0 - Criteria Completed

Component	**Amount Completed**
Criteria 1 - Access to the data using simple http web service calls (10 marks)	100% (Full implementation working.)
Criteria 2 - Options to return the data in text, json (the default) or xml (10 marks)	100% (Clear and understandable usage of formatting using libraries and able to cope with range of data objects)
Criteria 3 - Google App Engine or Microsoft Azure to implement the application on a remote cloud based server. (20 marks)	100% (Fully tested and demo'd CRUD operations on cloud server)
Criteria 4 - A WSDL description of the interface to the web service (5 marks)	100% (Full WSDL implemented and documented)
Criteria 5 - Access to the data using REST type interaction (10 marks)	100% (Full implementation working. Data transactions sent in correct RESTful way for all CRUD operations)
Criteria 6 - An Ajax based web front end to retrieve the data and display in a suitable format using library based routines for an enhanced user interface (15 marks)	100% (Excellent use of standard Ajax library and UI functionality)
Criteria 7 - Critical analysis of your work and the techniques you have used. (30 marks)	100% (Full discussion of SE techniques used and DPs with comparative evaluation.) – In a separate word document called "Critical Analysis".

1.0 - Criteria 1 - Simple HTTP Web Service Calls

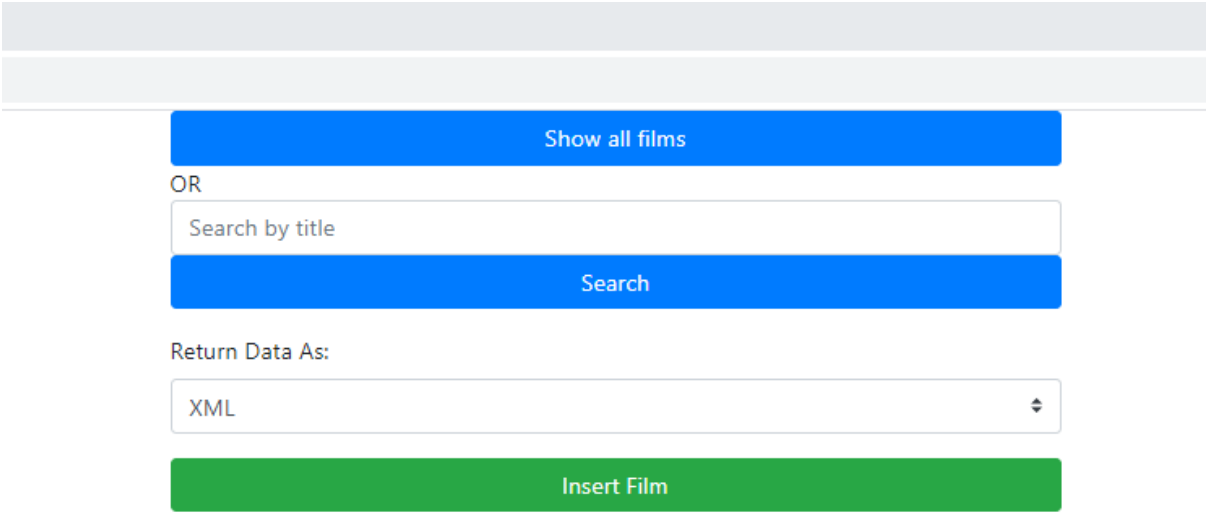
Criteria 1 can be accessed via the FilmHTTPWebService project, containing the backend (four packages, hibernate configuration files), the front-end (JavaScript / jQuery files, index.html) and the required jar files. The packages within this project are:

- Controller – Containing the individual Servlets, enabling HTTP requests to perform CRUD operations
- Dao – Containing the different types of data accessor objects (Traditional JDBC, Hibernate), the interface for these, and the factory class.
- Model – Containing the model classes: Film.java, FilmStore.java (for JAXB marshalling into a collection of films as XML), and Response.java (an object used to return a response to the user). The Film.java class "models" the film table in the database.
- Utilities – Containing the utility classes: FilmUtils.java, ResponseUtils.java. These are ultimately methods that are used in more than one servlet, promoting code-reuse.

Below is a table displaying the available methods that can be invoked, and the parameters required:

Key / Headers	**Values**	**HTTP Methods**	**Servlets that consume the key**
Key: format	xml, text, json (the default). This determines what format to return the data in.	GET	/getAllFilms /getFilmByID /getFilmsByTitle
Key: filmID	The ID of the film (10001, 10039, etc…)	GET - /getFilmByID DELETE - / deleteFilm	/getFilmByID /deleteFilm
Key: filmname	The search string for the title of the film	GET	/getFilmsByTitle
Header: Content-Type	Application/json, Application/xml. A buffered reader is used to grab the raw data, and then a conditional based on the content-type is used.	POST - /insertFilm PUT - /updateFilm	/insertFilm /updateFilm

When this project has been imported and is ran on Tomcat, navigate to this url: http://localhost:8080/FilmHTTPWebService/. Here you will be able to operate the CRUD system via a front-end that looks like this:



1.1 – Create Film (JSON)

After sending a JSON create request, the eclipse console will output the following:

```
Received request for inserting a film into the database!
Creating a Film object from data: application/json
0#Elf#2003#Jon Favreau#Will Ferrell, Zooey Deschanel#This movie is sweet and funny, though it can't quite seem to make up its min
Successful inserted, check db: 1
Request finished
{"response":"200 : ok","message":"Film has successfully been inserted (This response was sent in JSON)"}
```

To confirm it has been inserted into the database, MySQL Workbench is used:

```
1 •  SELECT * FROM films WHERE films.title = 'ELF'
2
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

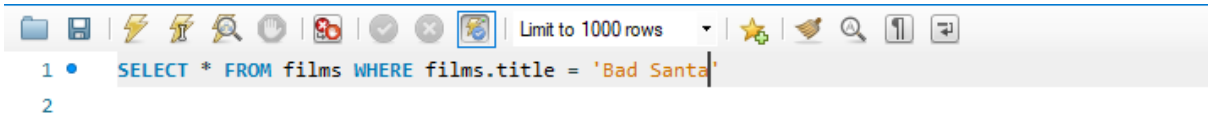
id	title	year	director	stars	review
11335	Elf	2003	Jon Favreau	Will Ferrell, Zooey Deschanel	This movie is sweet and funny, though it can't quite see…
NULL	NULL	NULL	NULL	NULL	NULL

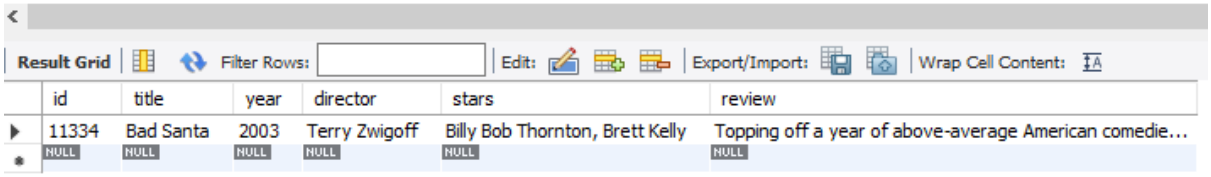
1.2 – Create Film (XML)

After sending an XML create request, the eclipse console will output the following:

```
Received request for inserting a film into the database!
Creating a Film object from data: application/xml
0#Bad Santa#2003#Terry Zwigoff#Billy Bob Thornton, Brett Kelly#Topping off a year of above-average American comedies - Dodgeball and Anch
Successful inserted, check db: 1
Request finished
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<responseXML>
    <response>200 : ok</response>
    <message>Film has successfully been inserted (This response was sent in XML)</message>
</responseXML>
```

To confirm it has been inserted into the database, MySQL Workbench is used:



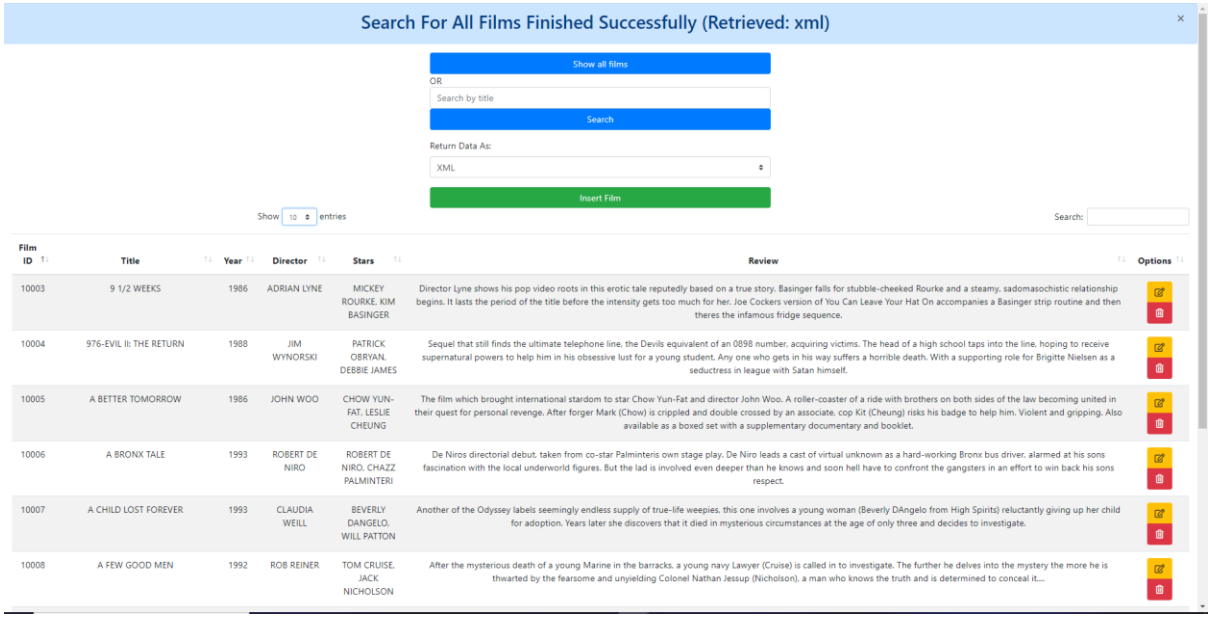


1.3 – Retrieve All Films

After requesting to retrieve all films, eclipse will display the following messages in console:

```
Get All Films request retrieved!
Chosen format type for all films is: xml
Films have been formatted
```

The front-end will then display all the films in a table:

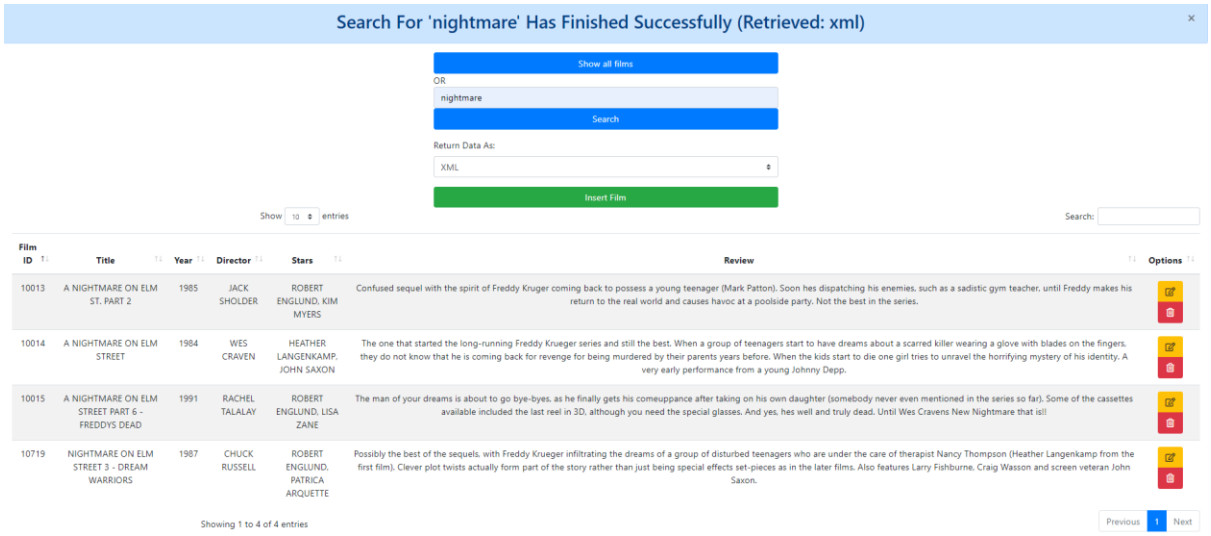


1.4 – Retrieve Films By Title

After requesting to retrieve films by search name relating to the title, eclipse will display the following messages in console:

```
Received request for retrieving all films by title!
Format type requested is: xml and the search name is: nightmare
Films have been formatted
```

The front-end will then display the films that were found:

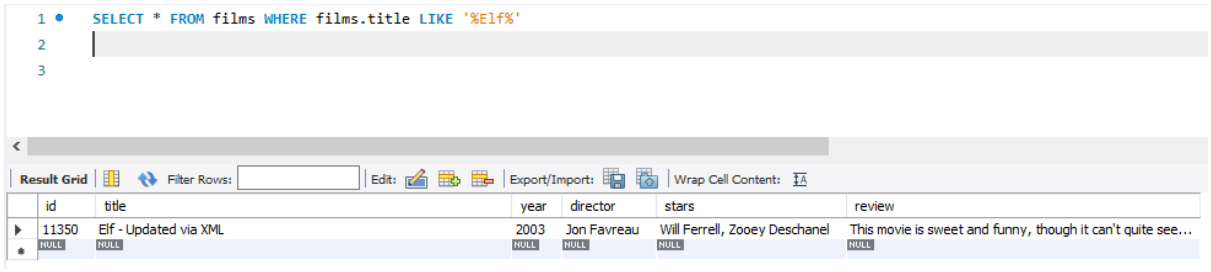


1.5 – Update A Film (XML)

After sending a request to update the film via XML, the eclipse console will display:

```
Received request to update film!
Updating a Film object from data: application/xml
11350#Elf - Updated via XML#2003#Jon Favreau#Will Ferrell, Zooey Deschanel#This movie is sweet and funny, though it
Successful updated, check db: 1
Request finished
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<responseXML>
    <response>200 : ok</response>
    <message>Film has successfully been updated (This response was sent in XML)</message>
</responseXML>
```

To confirm it has been updated in the database, MySQL Workbench is used:



1.6 – Update A Film (JSON)

After sending a request to update the film via JSON, the eclipse console will display:

```
Received request to update film!
Updating a Film object from data: application/json
11334#Bad Santa - Updated via JSON#2003#Terry Zwigoff#Billy Bob Thornton, Brett Kelly#Topping off a year c
Successful updated, check db: 1
Request finished
{"response":"200 : ok","message":"Film has successfully been updated (This response was sent in JSON)"}
```

To confirm it has been updated in the database, MySQL Workbench is used:

```
1 •  SELECT * FROM films WHERE films.title LIKE '%Bad Santa%'
2
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	title	year	director	stars	review
11334	Bad Santa - Updated via JSON	2003	Terry Zwigoff	Billy Bob Thornton, Brett Kelly	Topping off a year of above-average American comedie…
NULL	NULL	NULL	NULL	NULL	NULL

1.7 – Deleting A Film

After sending a request to delete a film, the eclipse console will display the following:

```
Delete request retrieved!
Film requested to be deleted: 11335
Successful delete, check db: 1
{"response":"200 : ok","message":"Film: 11335 has successfully been deleted"}
```

To confirm the Film with ID “11335” was deleted from the database, MySQL workbench is used:

Limit to 1000 rows

```
1 •  SELECT * FROM films WHERE films.id = 11335
2
```

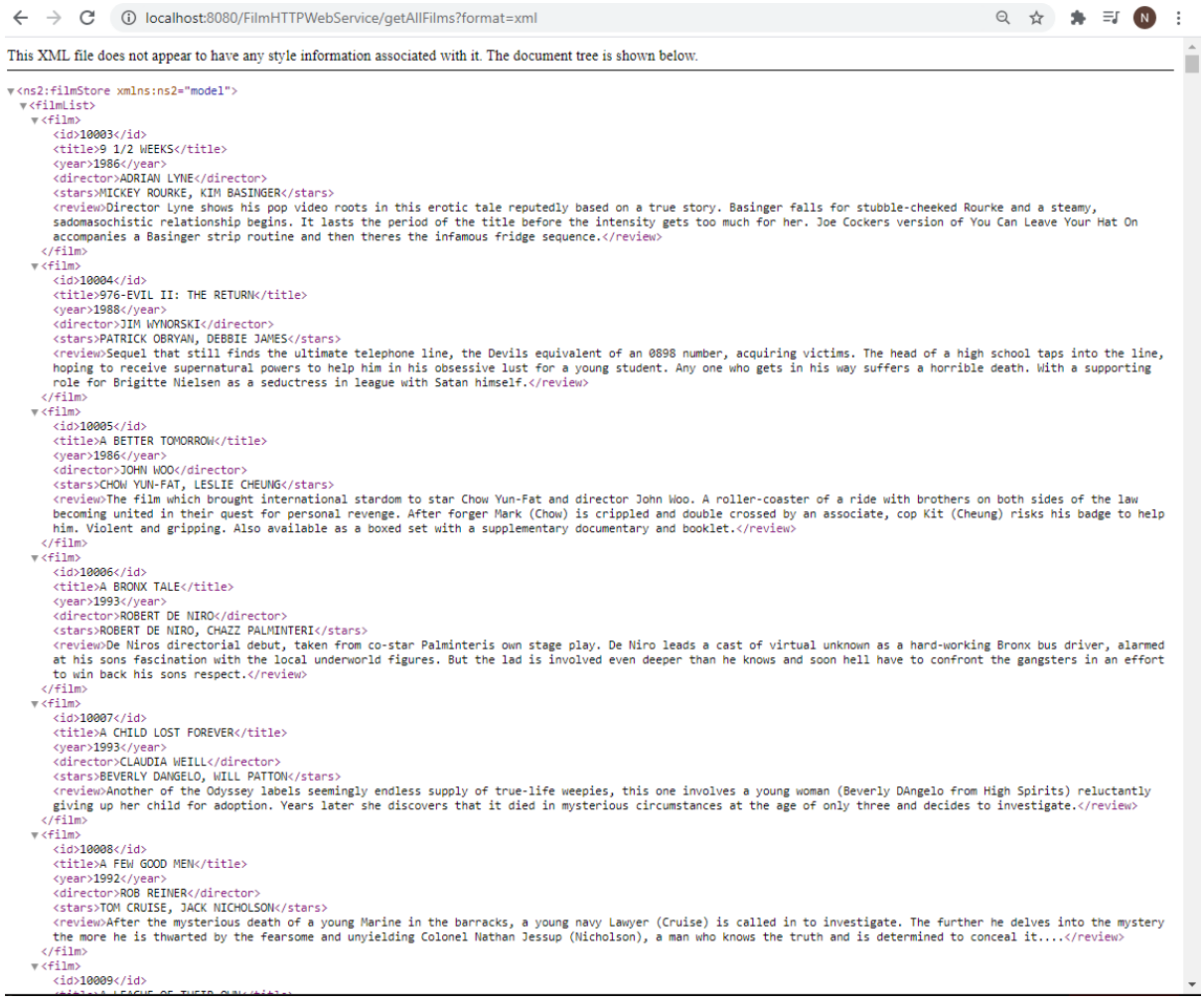
Result Grid | Filter Rows: | Edit: | Export/Import:

id	title	year	director	stars	review
NULL	NULL	NULL	NULL	NULL	NULL

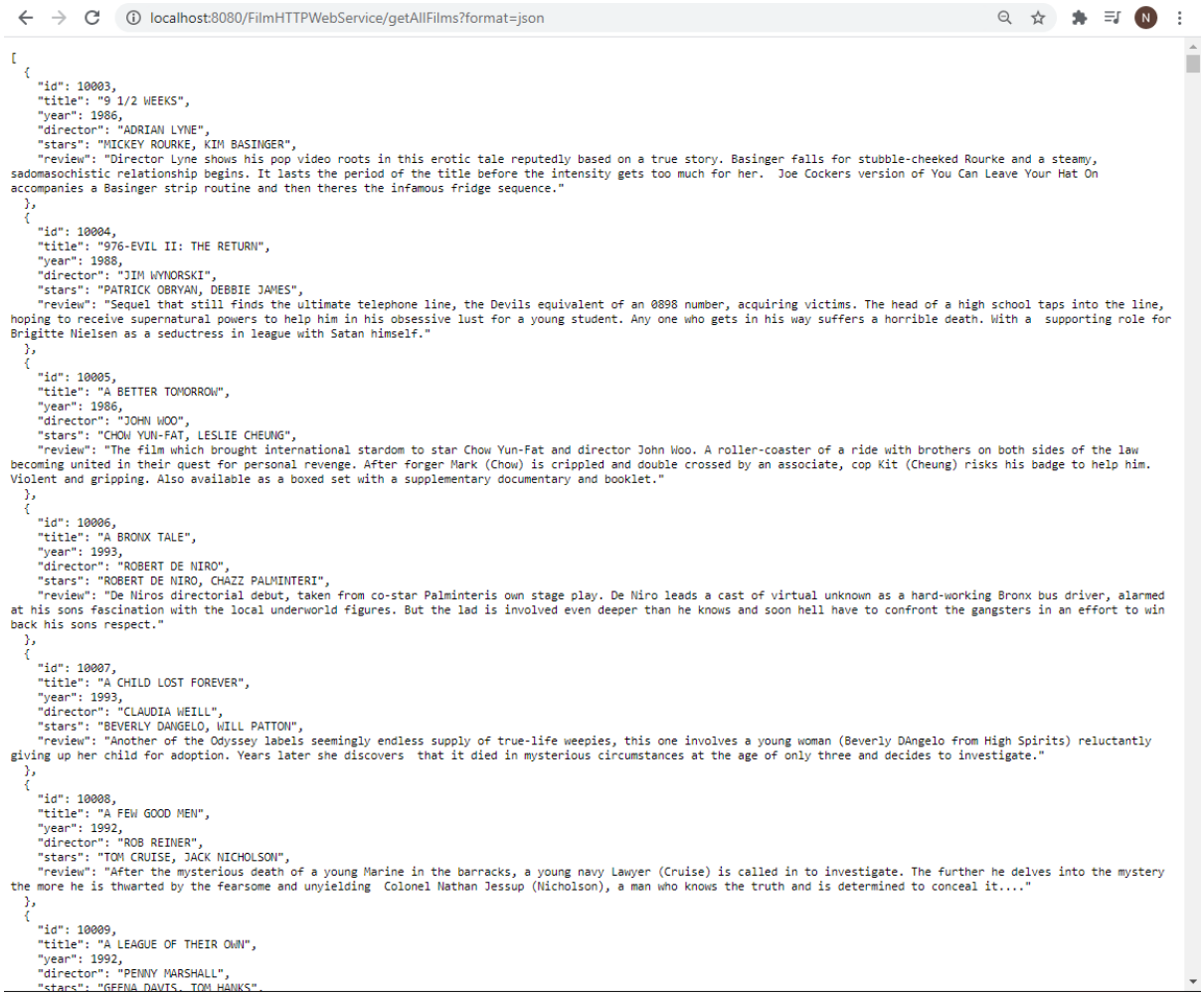
2.0 - Criteria 2 – Returning Data in Different Formats

For the HTTP project that uses servlets, JAXB and Gson are used to explicitly format the films into XML / JSON, and an overridden toString method was used for the formatting to TEXT. These solutions can cope with a range of data objects. For the RESTful project, JAXB is still used for annotations, but the formatting is implicitly done for us via spring annotations.

2.1 – Returning as XML

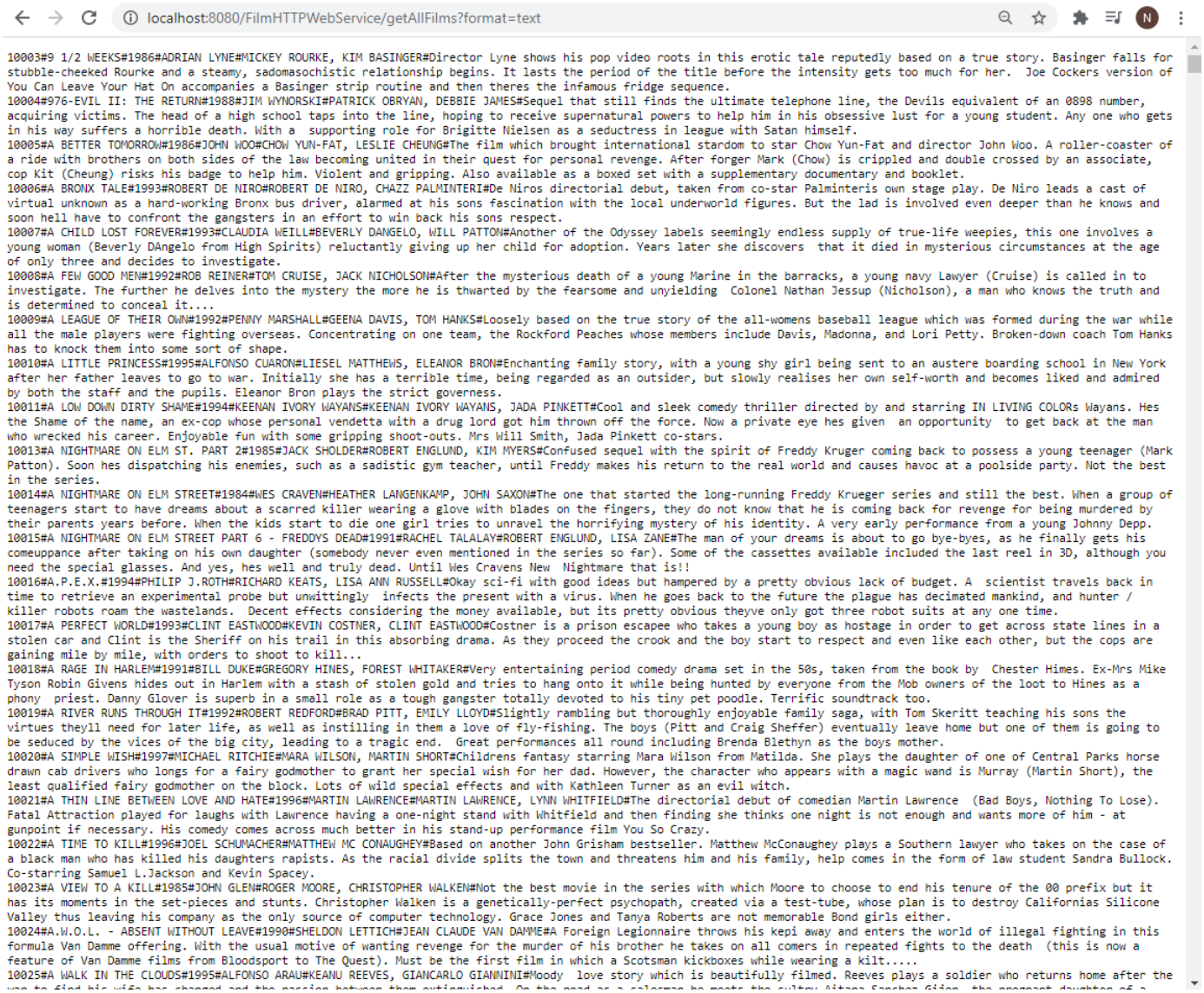


2.2 – Returning as JSON (the default if non-specified)




```
[
  {
    "id": 10003,
    "title": "9 1/2 WEEKS",
    "year": 1986,
    "director": "ADRIAN LYNE",
    "stars": "MICKEY ROURKE, KIM BASINGER",
    "review": "Director Lyne shows his pop video roots in this erotic tale reputedly based on a true story. Basinger falls for stubble-cheeked Rourke and a steamy, sadomasochistic relationship begins. It lasts the period of the title before the intensity gets too much for her.  Joe Cockers version of You Can Leave Your Hat On accompanies a Basinger strip routine and then theres the infamous fridge sequence."
  },
  {
    "id": 10004,
    "title": "976-EVIL II: THE RETURN",
    "year": 1988,
    "director": "JIM WYNORSKI",
    "stars": "PATRICK OBRYAN, DEBBIE JAMES",
    "review": "Sequel that still finds the ultimate telephone line, the Devils equivalent of an 0898 number, acquiring victims. The head of a high school taps into the line, hoping to receive supernatural powers to help him in his obsessive lust for a young student. Any one who gets in his way suffers a horrible death. With a  supporting role for Brigitte Nielsen as a seductress in league with Satan himself."
  },
  {
    "id": 10005,
    "title": "A BETTER TOMORROW",
    "year": 1986,
    "director": "JOHN WOO",
    "stars": "CHOW YUN-FAT, LESLIE CHEUNG",
    "review": "The film which brought international stardom to star Chow Yun-Fat and director John Woo. A roller-coaster of a ride with brothers on both sides of the law becoming united in their quest for personal revenge. After forger Mark (Chow) is crippled and double crossed by an associate, cop Kit (Cheung) risks his badge to help him. Violent and gripping. Also available as a boxed set with a supplementary documentary and booklet."
  },
  {
    "id": 10006,
    "title": "A BRONX TALE",
    "year": 1993,
    "director": "ROBERT DE NIRO",
    "stars": "ROBERT DE NIRO, CHAZZ PALMINTERI",
    "review": "De Niros directorial debut, taken from co-star Palminteris own stage play. De Niro leads a cast of virtual unknown as a hard-working Bronx bus driver, alarmed at his sons fascination with the local underworld figures. But the lad is involved even deeper than he knows and soon hell have to confront the gangsters in an effort to win back his sons respect."
  },
  {
    "id": 10007,
    "title": "A CHILD LOST FOREVER",
    "year": 1993,
    "director": "CLAUDIA WEILL",
    "stars": "BEVERLY DANGELO, WILL PATTON",
    "review": "Another of the Odyssey labels seemingly endless supply of true-life weepies, this one involves a young woman (Beverly DAngelo from High Spirits) reluctantly giving up her child for adoption. Years later she discovers  that it died in mysterious circumstances at the age of only three and decides to investigate."
  },
  {
    "id": 10008,
    "title": "A FEW GOOD MEN",
    "year": 1992,
    "director": "ROB REINER",
    "stars": "TOM CRUISE, JACK NICHOLSON",
    "review": "After the mysterious death of a young Marine in the barracks, a young navy Lawyer (Cruise) is called in to investigate. The further he delves into the mystery the more he is thwarted by the fearsome and unyielding  Colonel Nathan Jessup (Nicholson), a man who knows the truth and is determined to conceal it...."
  },
  {
    "id": 10009,
    "title": "A LEAGUE OF THEIR OWN",
    "year": 1992,
    "director": "PENNY MARSHALL",
    "stars": "GEENA DAVIS, TOM HANKS",
```

2.3 – Returning as TEXT



3.0 - Criteria 3 & Criteria 6 – Google App-Engine With Cloud Database & AJAX Based Web Front-end

For Criteria 3, I copied the "FilmHTTPWebService" project code over to a new Google App-Engine project "FilmCloudWebService", and configured the Data Accessor Object configuration (Hibernate in this case) to target the Cloud SQL database I set up.

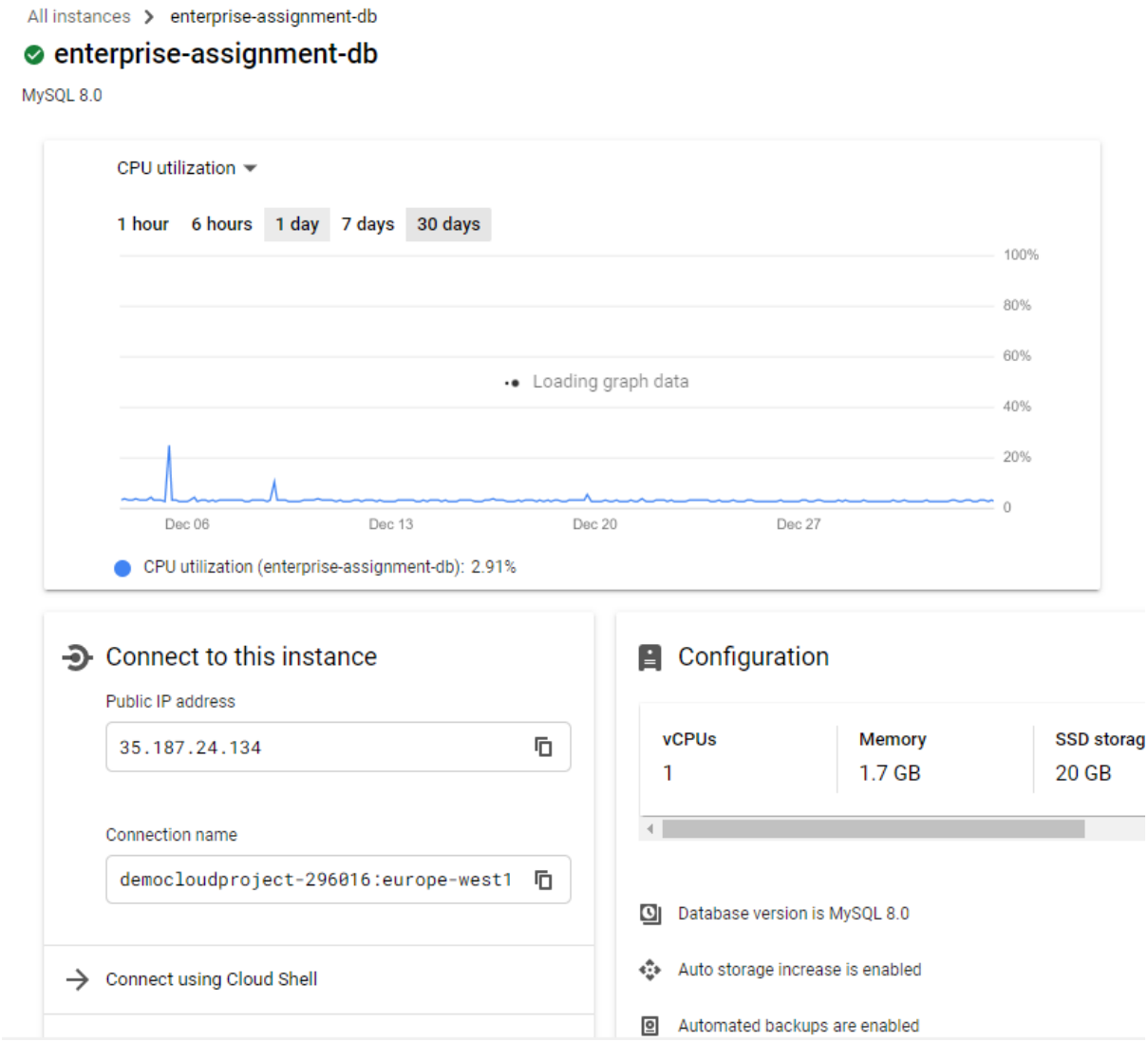
The front-end used for the Cloud in the "FilmCloudWebService" project is the same for "FilmHTTPWebService" and "FilmRESTfulWebService" projects.

Demonstrating the front-end in this section and confirming results with MySQL Workbench will show that the Cloud CRUD operations fully work (satisfying criteria 3), and will also evidence the use of UI libraries. jQuery was used for AJAX calls and wherever else possible and most convenient to do so.

For the front-end, DataTables & bootstrap is used to generate and style the dynamic film table.

The link to the live cloud version is: https://democloudproject-296016.ew.r.appspot.com/

Below is proof of the Google Cloud database from Google console:

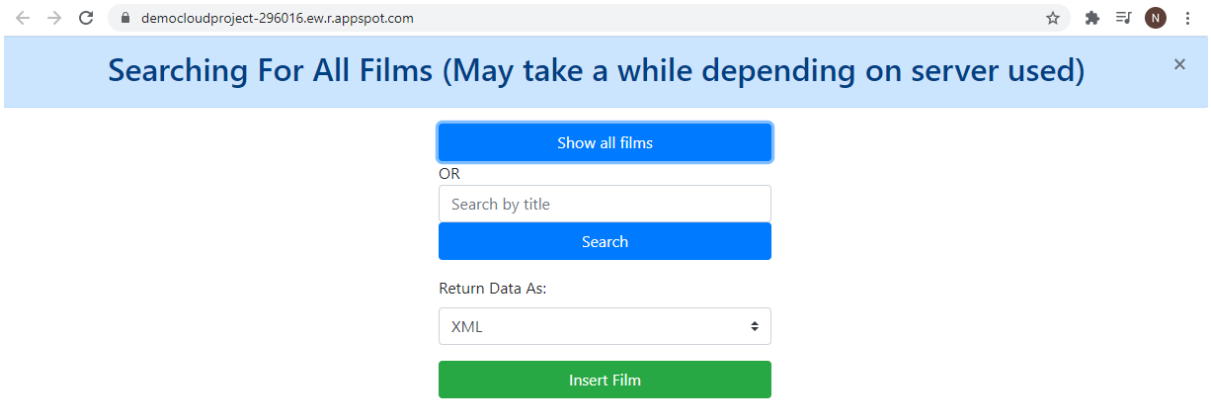


Below shows the hibernate configuration used when deploying the Google App Engine to the Cloud, so it will target the Cloud Database.

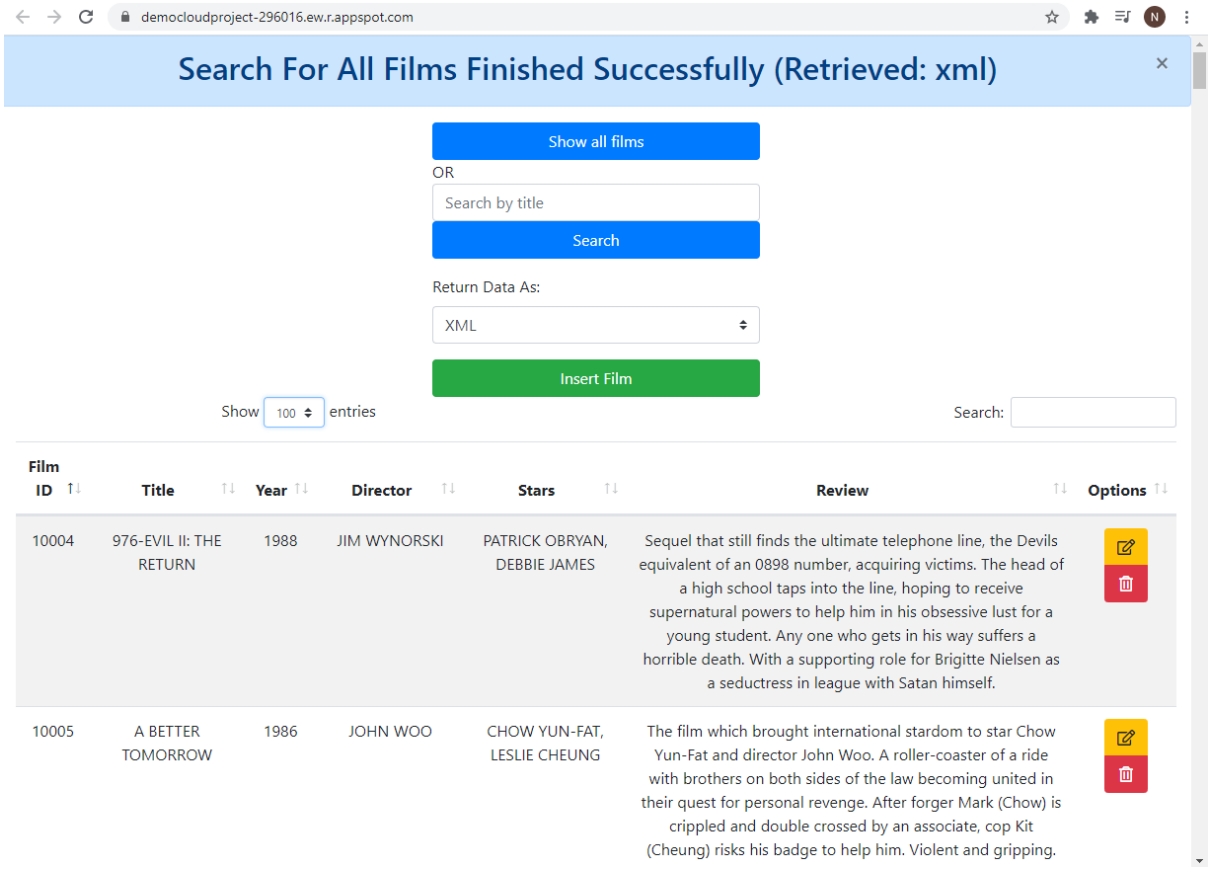
```
<property name="hibernate.connection.url">jdbc:mysql:///</property>
<property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">admin</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.default_schema">filmdb</property>
<property name="hibernate.connection.socketFactory">com.google.cloud.sql.mysql.SocketFactory</property>
<property name="hibernate.connection.cloudSqlInstance">democloudproject-296016:europe-west1:enterprise-assignment-db</property>
```

3.1 – Get All Films

When “Show All Films” is pressed, a loading div appears whilst the ajax operation takes place and waits for the server to respond. The first request may take some time as I have selected the cheapest database option on Cloud SQL.



When the AJAX operation finishes, the film table appears, and the alert div updates to a successful response.

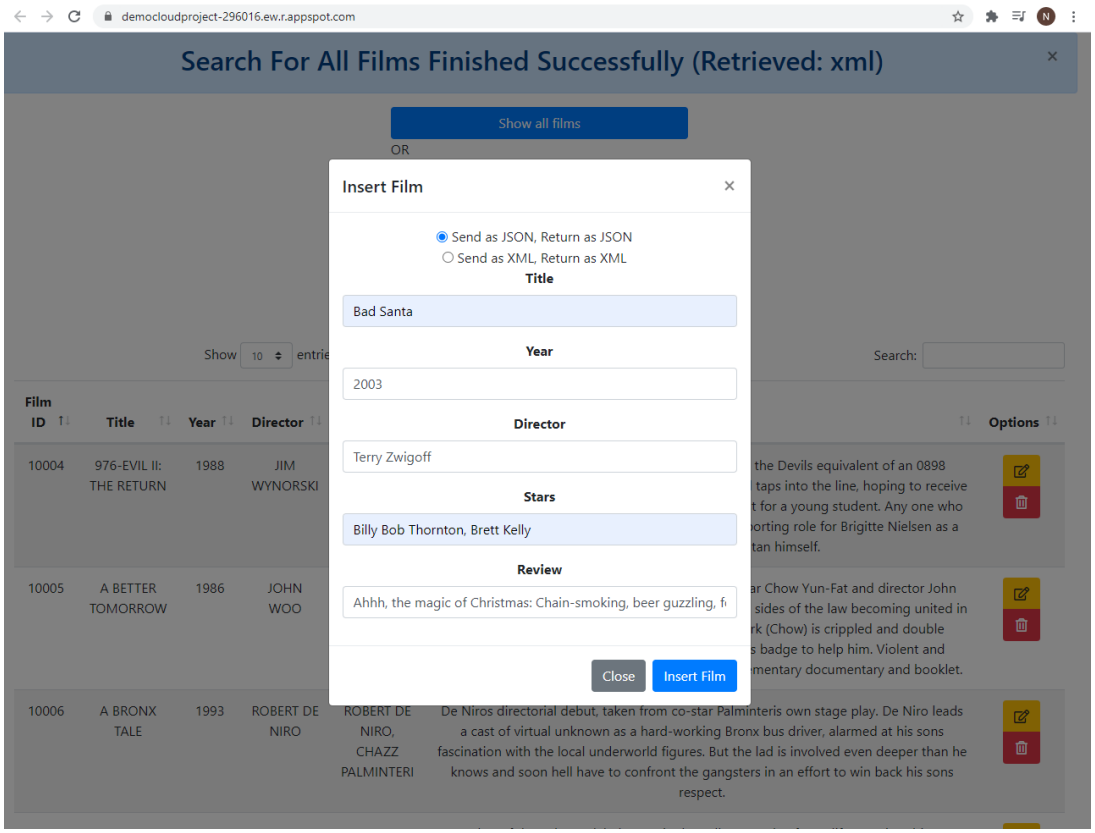


3.2 – Get Films By Title

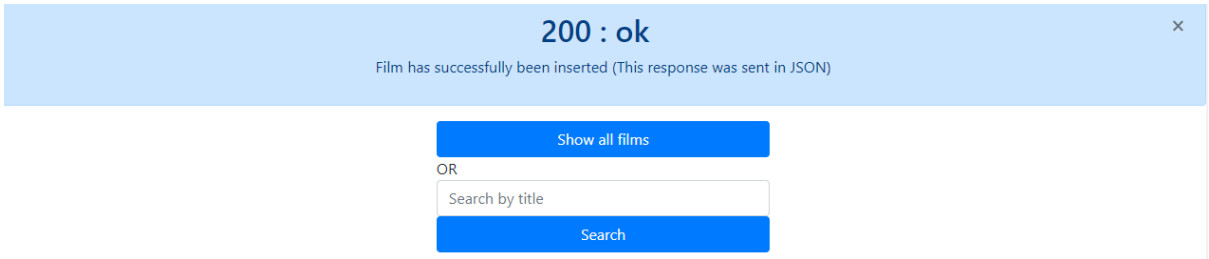
The below image shows the result of searching for “Nightmare on elm street 3” (There is also a loading div that appears during the ajax call):

3.3 – Create Film

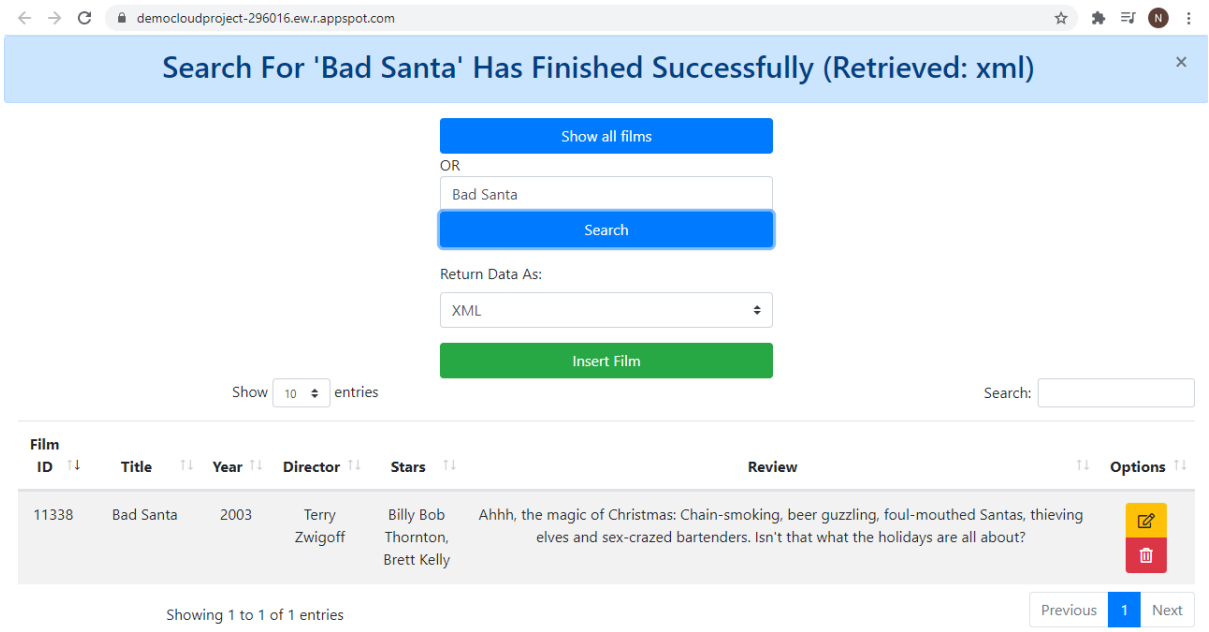
When “Insert Film” is pressed, a bootstrap modal pops up with a form for the user to fill in and submit.



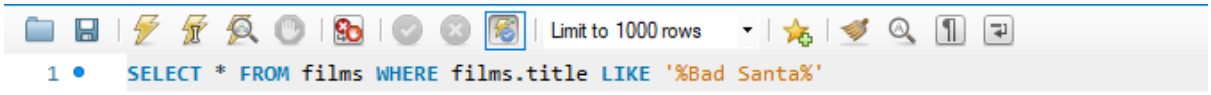
When “Insert Film” has been pressed again, the modal box should close, and a response will be displayed at the top of the screen based on whether it was a success or fail (the server sends back a response in XML / JSON format depending on how you sent the film data).

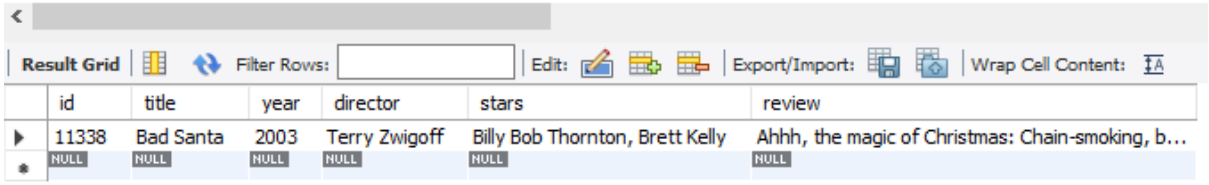


When searching for the inserted film by title, it now displays in the table:



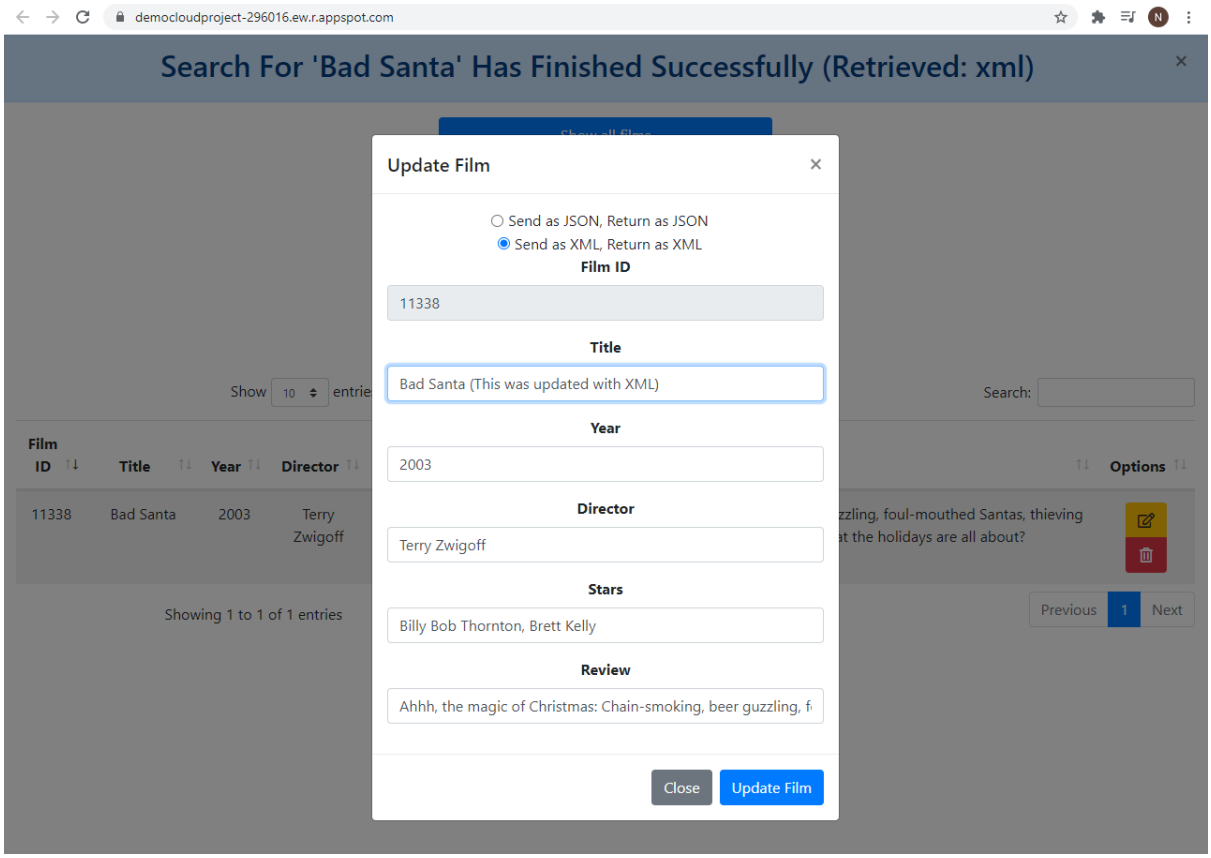
To confirm with MySQL Workbench:



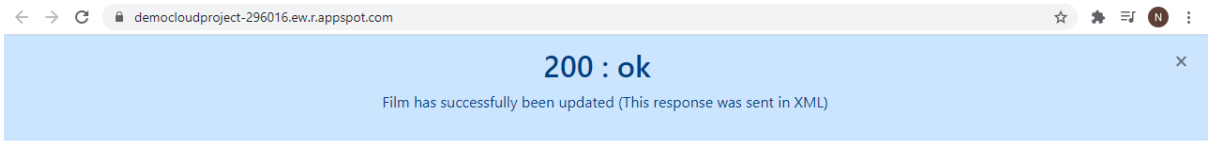


3.4 – Update Film

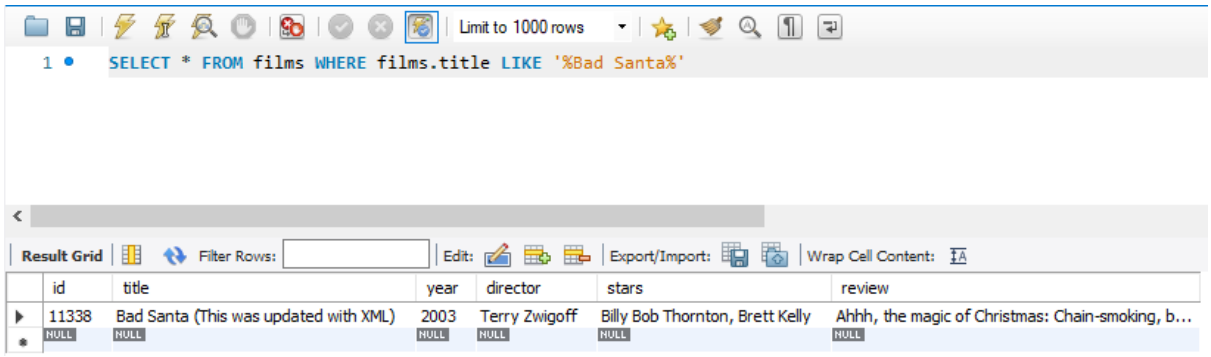
When the update button is pressed on a film row, an update film modal will pop up, with the existing data of that film already set in the input boxes for the user.



When “Update Film” is pressed again on the modal, it will close and a response will be shown at the top of the screen with a message based on whether it succeeded or not.

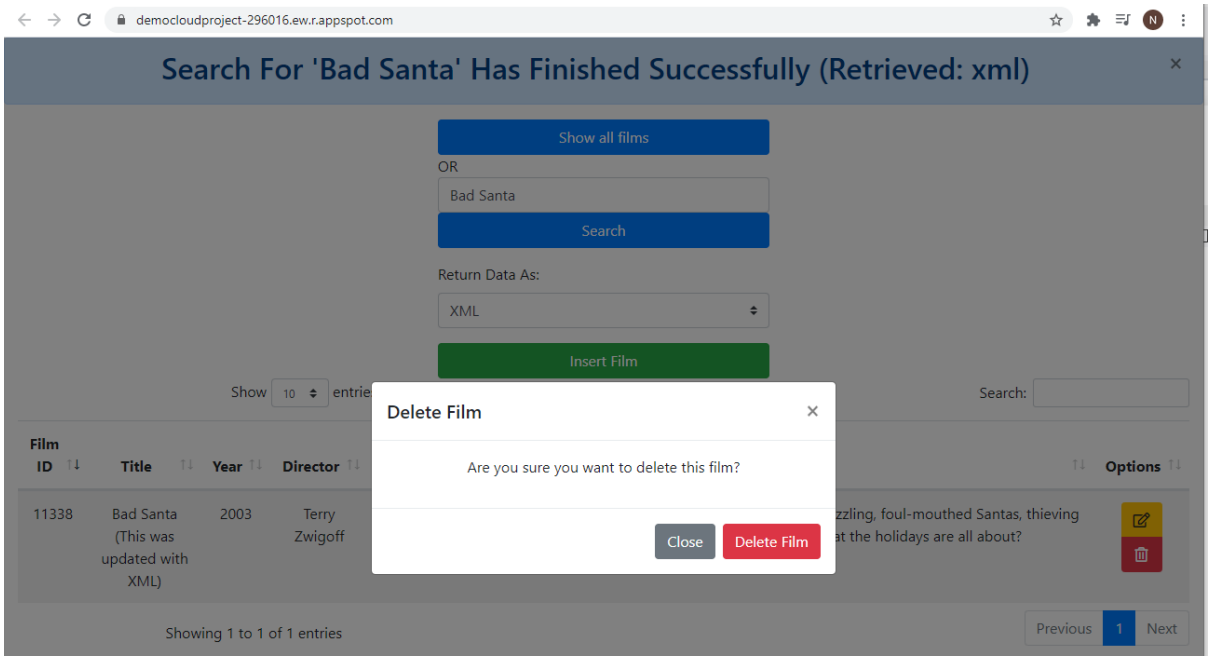


To confirm this update worked using MySQL Workbench:

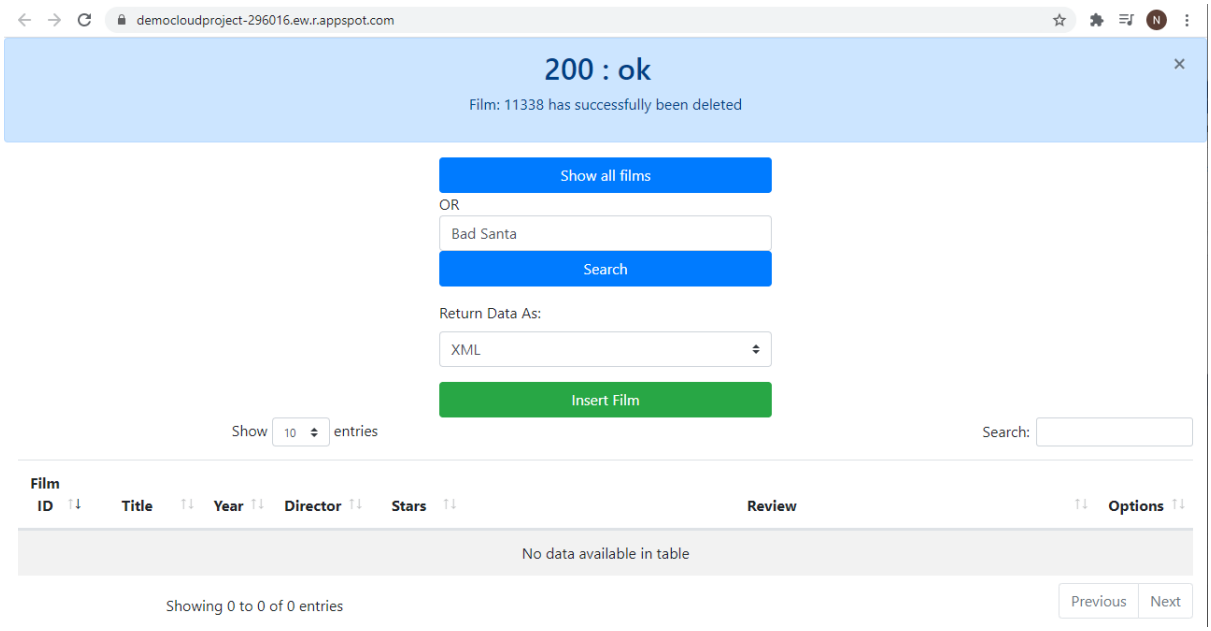


3.5 – Delete Film

When the delete button is pressed on the film row, a modal will pop up to confirm whether the user wishes to permanently delete this film.



When the delete button is pressed, a response will display at the top of the screen based on whether it was a success or not. If it was a success, the table will remove the row automatically.



To confirm this delete worked using MySQL Workbench:

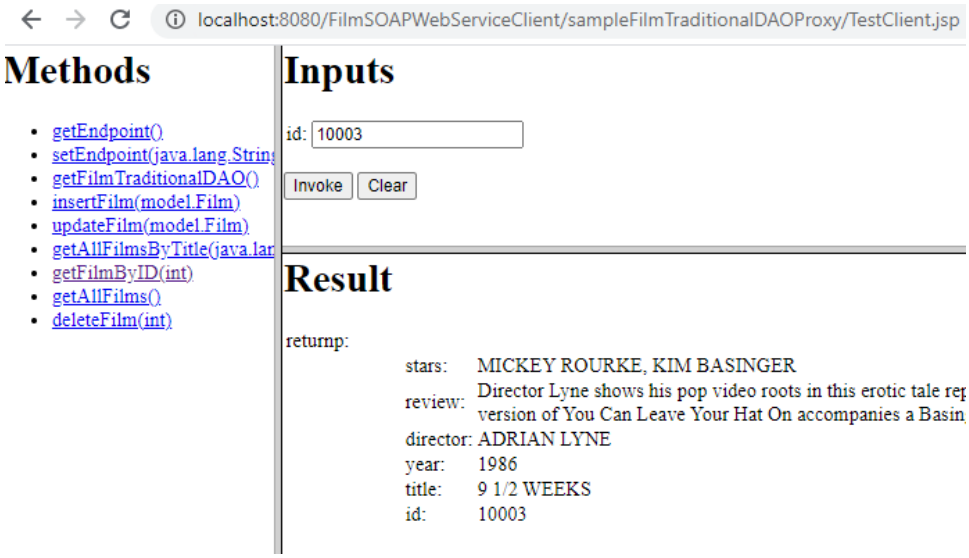
4.0 – Criteria 4 - WSDL File and SOAP Client

The WSDL file can be seen below:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://dao" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://dao" xmlns:int
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://dao" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://model"/>
   <element name="insertFilm">
    <complexType>
     <sequence>
      <element name="film" type="tns1:Film"/>
     </sequence>
    </complexType>
   </element>
   <element name="insertFilmResponse">
    <complexType>
     <sequence>
      <element name="insertFilmReturn" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="updateFilm">
    <complexType>
     <sequence>
      <element name="film" type="tns1:Film"/>
     </sequence>
    </complexType>
   </element>
   <element name="updateFilmResponse">
    <complexType>
     <sequence>
      <element name="updateFilmReturn" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="getAllFilmsByTitle">
    <complexType>
     <sequence>
      <element name="searchName" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="getAllFilmsByTitleResponse">
    <complexType>
     <sequence>
      <element maxOccurs="unbounded" name="getAllFilmsByTitleReturn" type="tns1:Film"/>
     </sequence>
    </complexType>
   </element>
   <element name="getFilmByID">
    <complexType>
     <sequence>
      <element name="id" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="getFilmByIDResponse">
    <complexType>
```

Design Source

The SOAP Client can be seen below:



5.0 – Criteria 5 – RESTful server, With REST type interaction

The project "FilmRESTfulWebService" satisfies this criteria.

This project's package layout is the same as "FilmHTTPWebService", however – since it uses Spring, there are a few differences:

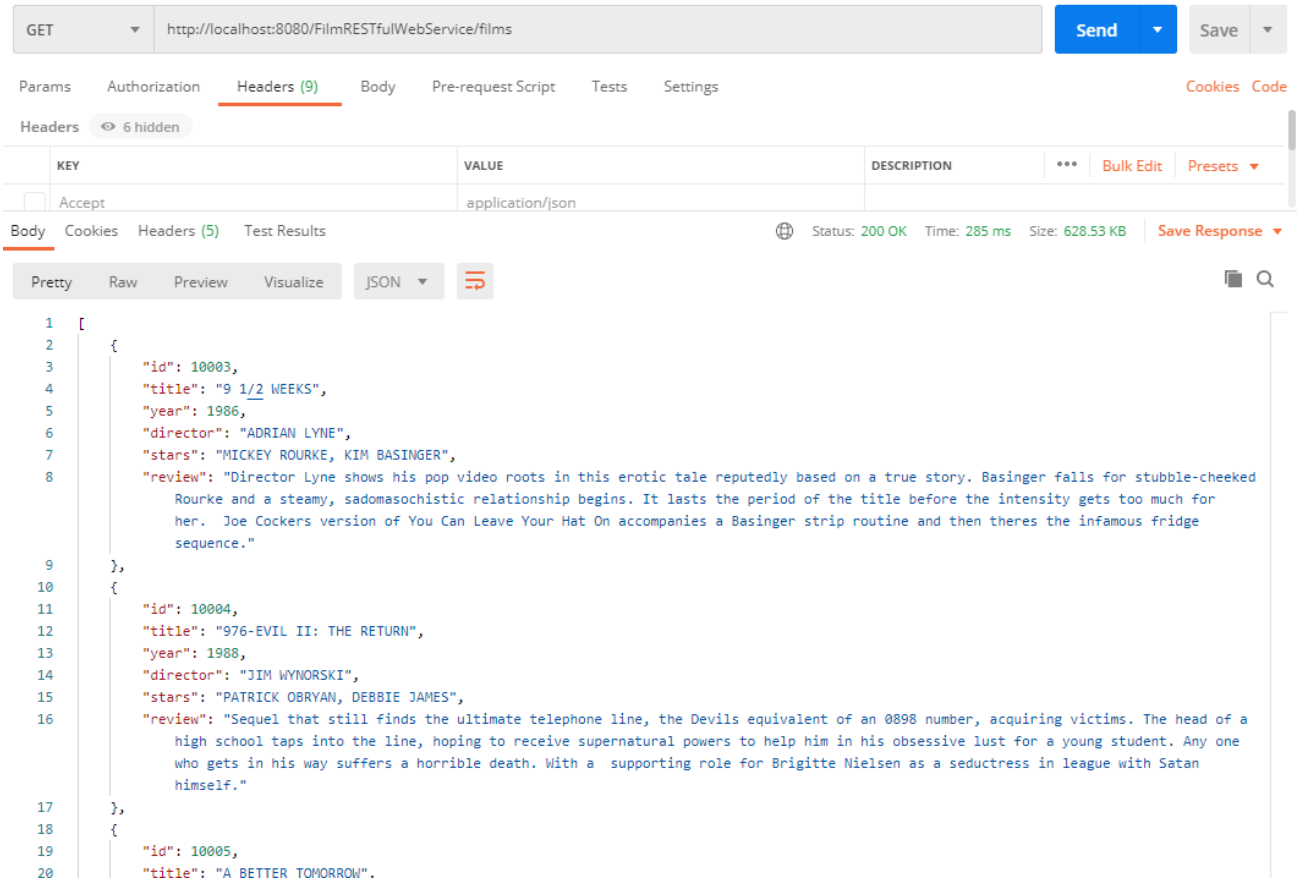
- There is one single class within the controller package (our controller class)
- The dao package contains only the interface and the hibernate DAO, rather than the traditional DAO & factory class too (it was already demonstrated in the aforementioned project)
- The utilities package contains only the ResponseUtils class and now has only one method. This is because Spring automatically parses JSON and does the marshalling / unmarshalling of XML for us, methods that parse formats are no longer required
- The model package however is the exact same

When you have imported this project and ran it on tomcat, navigate to the URL for a front-end: http://localhost:8080/FilmRESTfulWebService/

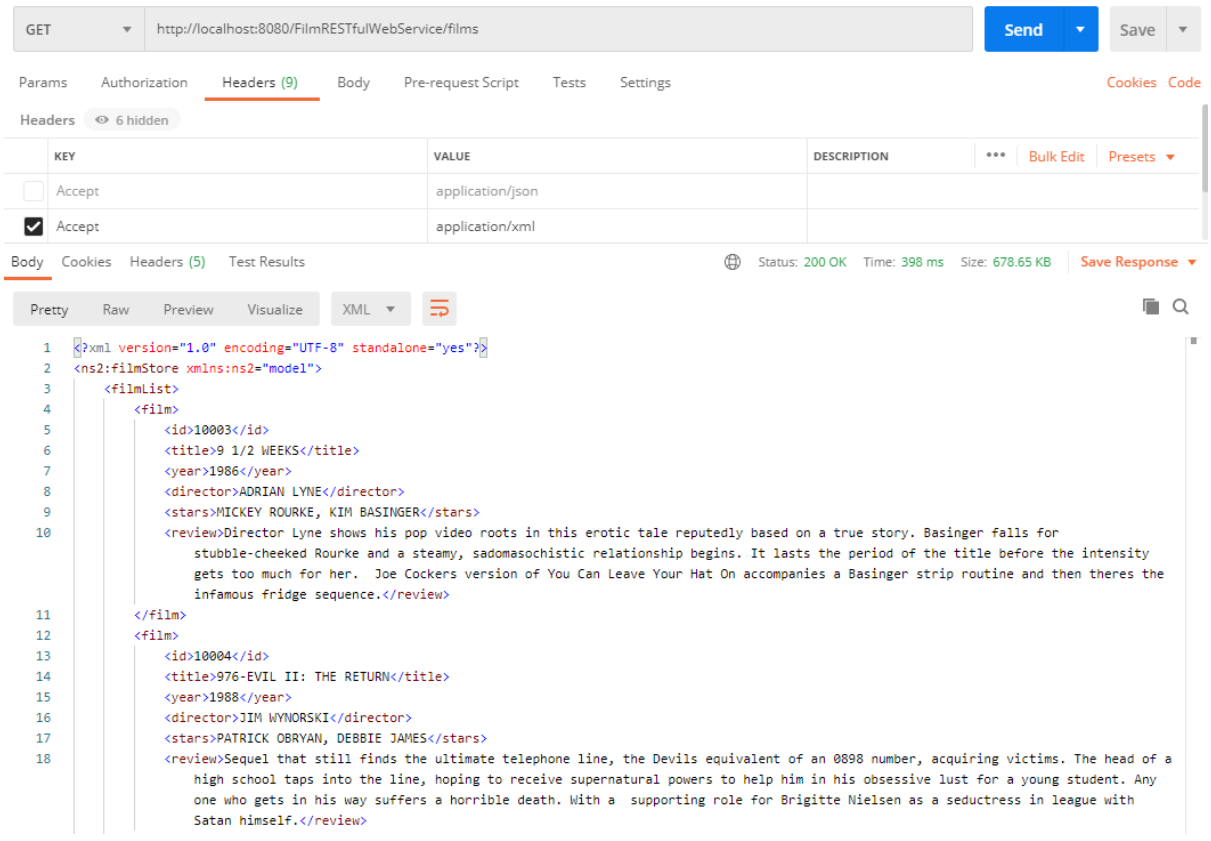
I will be using Postman in order to demonstrate the RESTful type of interface I have implemented.

Resource Path	**Request Type**
http://localhost:8080/FilmRESTfulWebService/films	**GET** – Retrieves all films in requested format
http://localhost:8080/FilmRESTfulWebService/films/{searchName}	**GET** – Retrieves films using the search name as a filter
http://localhost:8080/FilmRESTfulWebService/films/film/{filmID}	**GET** – Retrieves a film by its ID
http://localhost:8080/FilmRESTfulWebService/films/film	**POST** – Create a Film with either JSON or XML format
http://localhost:8080/FilmRESTfulWebService/films/film/{filmID}	**PUT** – Update a Film with either JSON or XML format
http://localhost:8080/FilmRESTfulWebService/films/film/{filmID}	**DELETE** – Delete a film using the ID sent in the path variable.

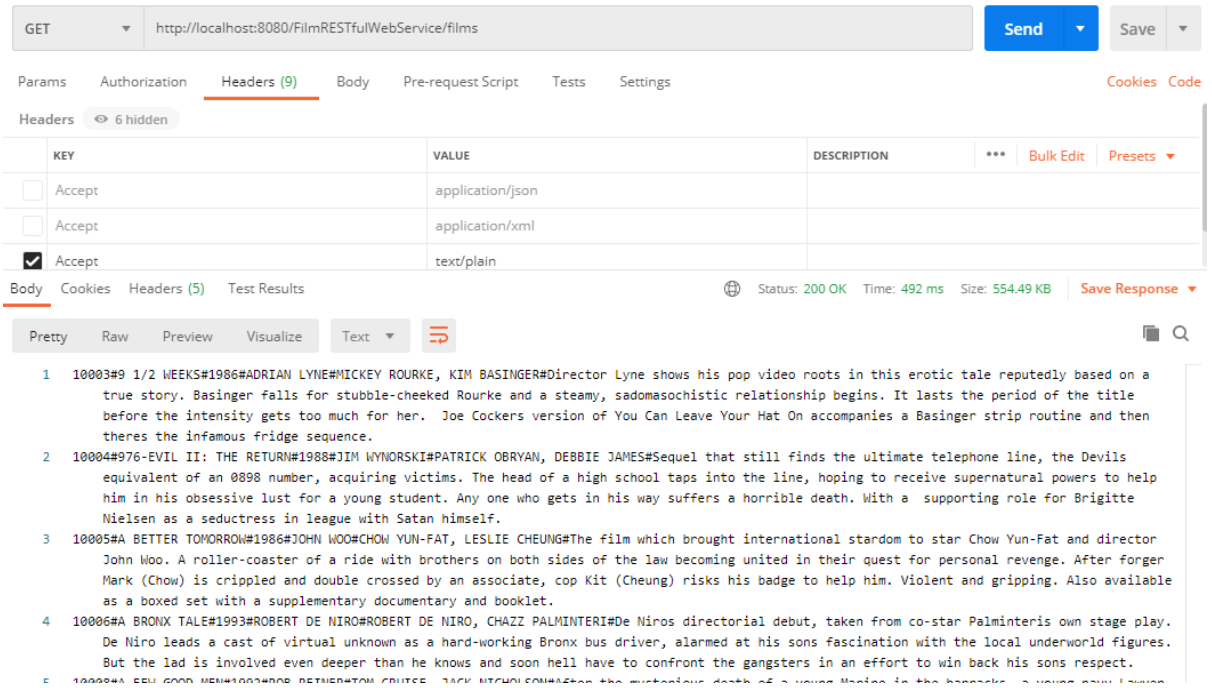
5.1 – Get All Films (JSON)



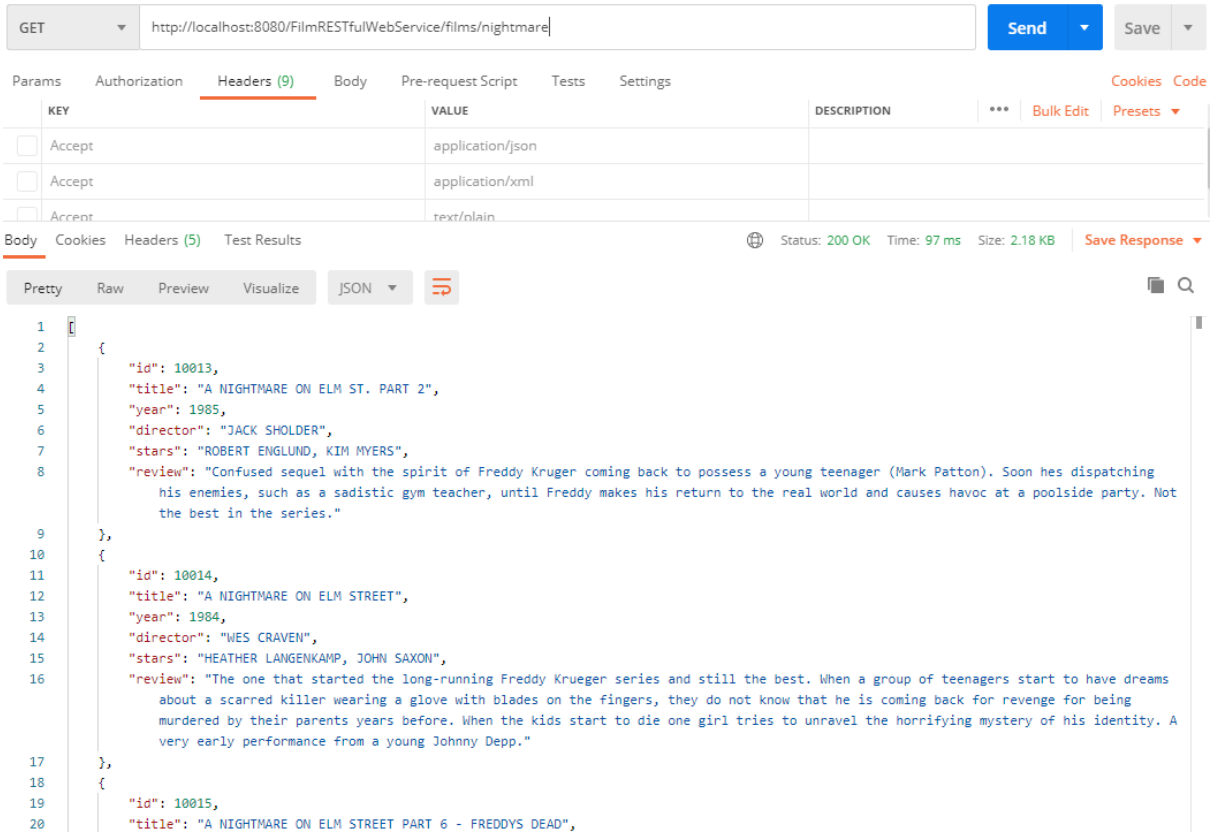
5.2 – Get All Films (XML)



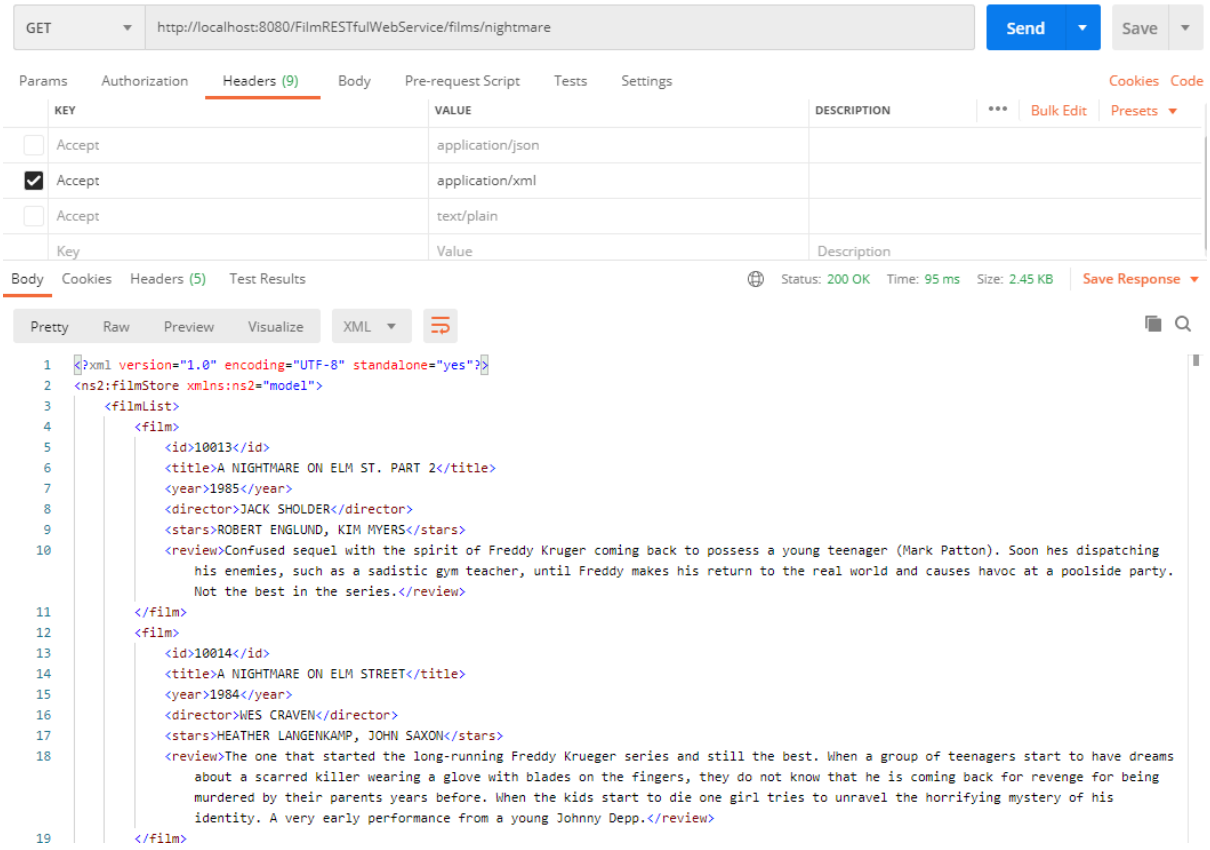
5.3 – Get All Films (TEXT)



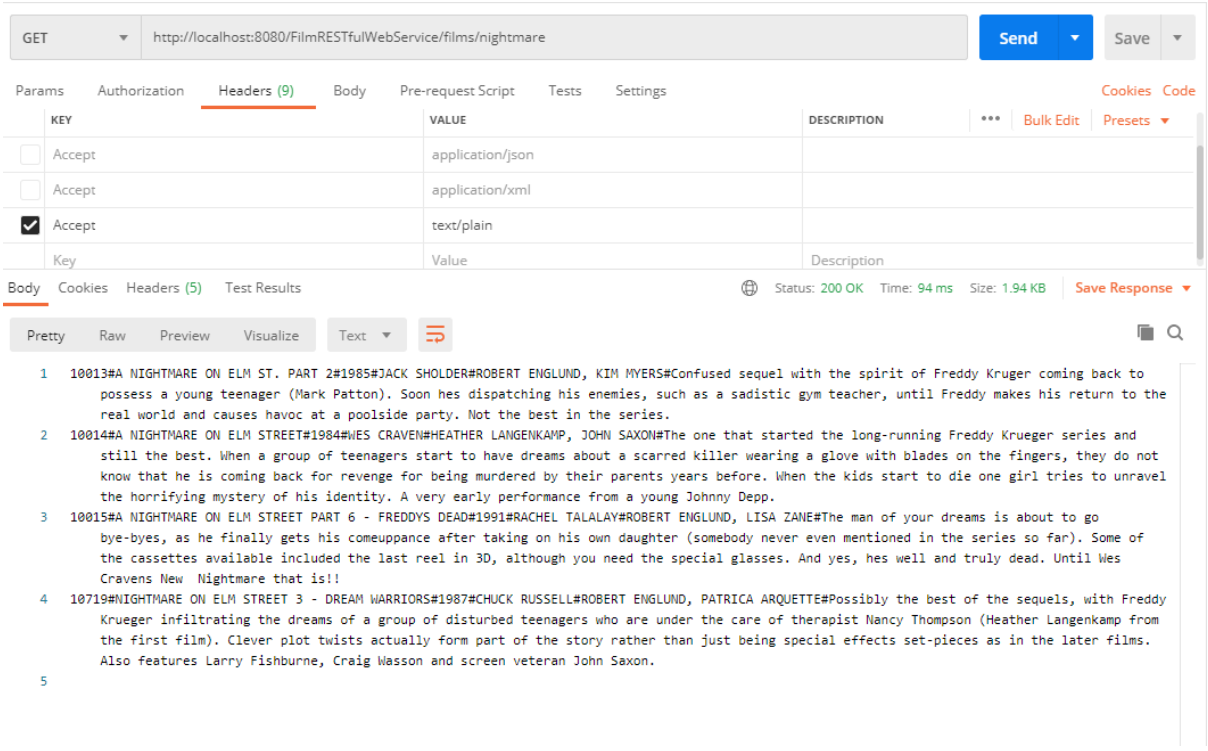
5.4 – Get Films By Title (JSON)



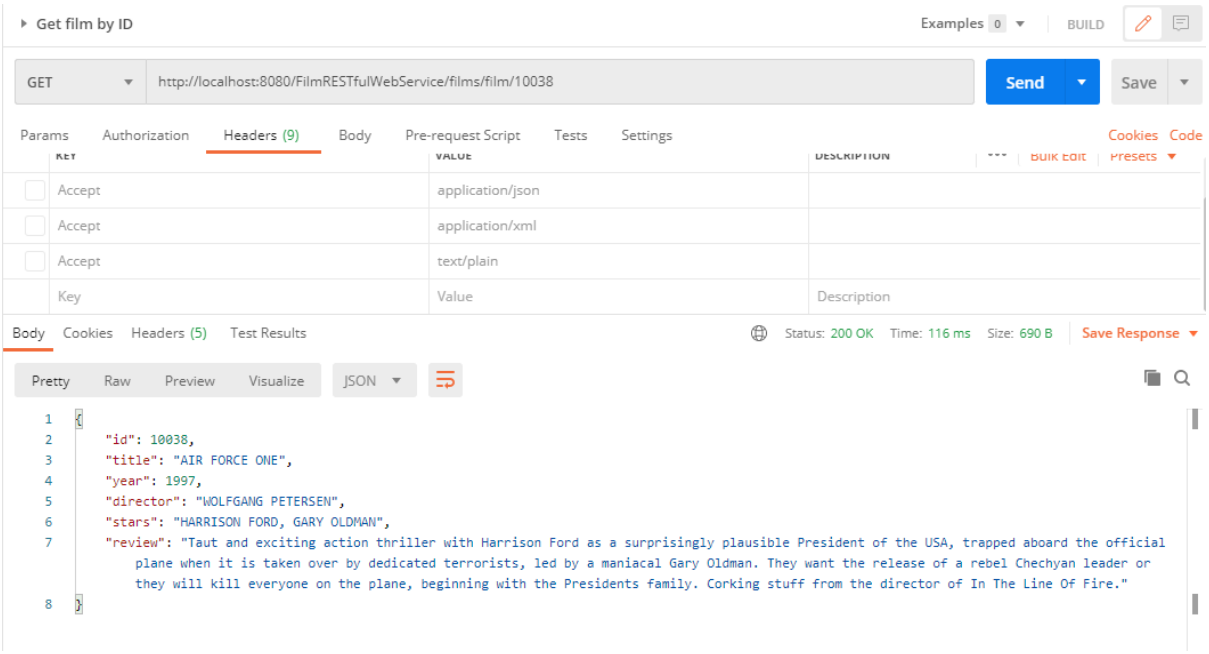
5.5 – Get Films By Title (XML)



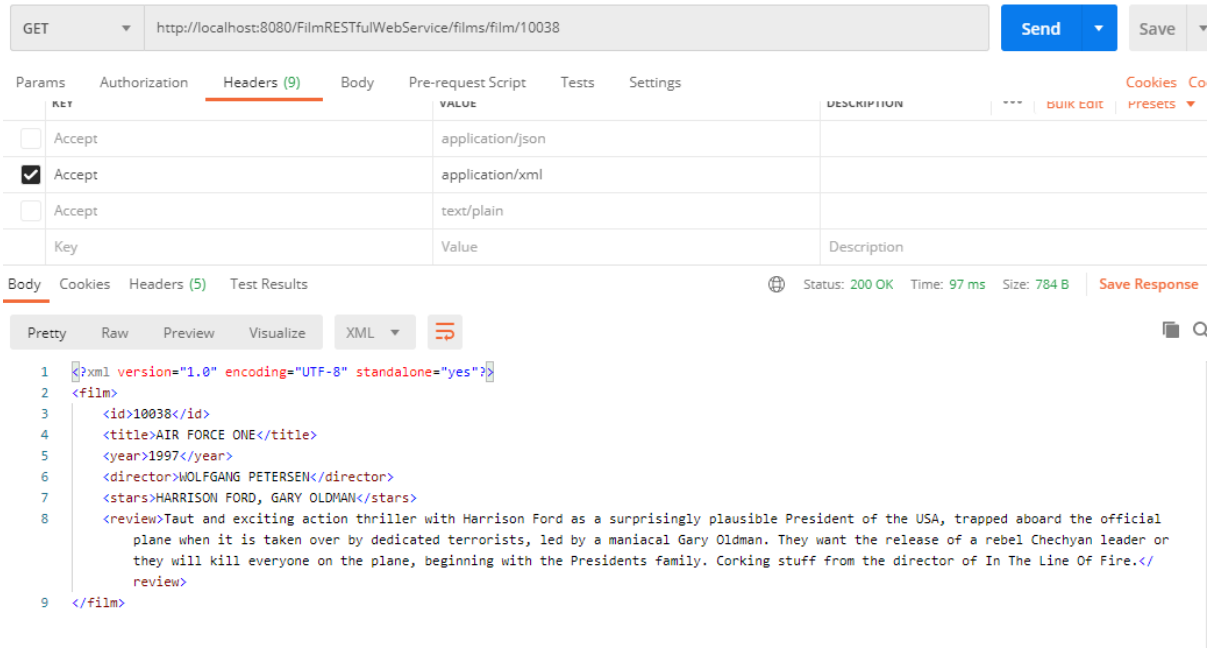
5.6 - Get Films By Title (TEXT)



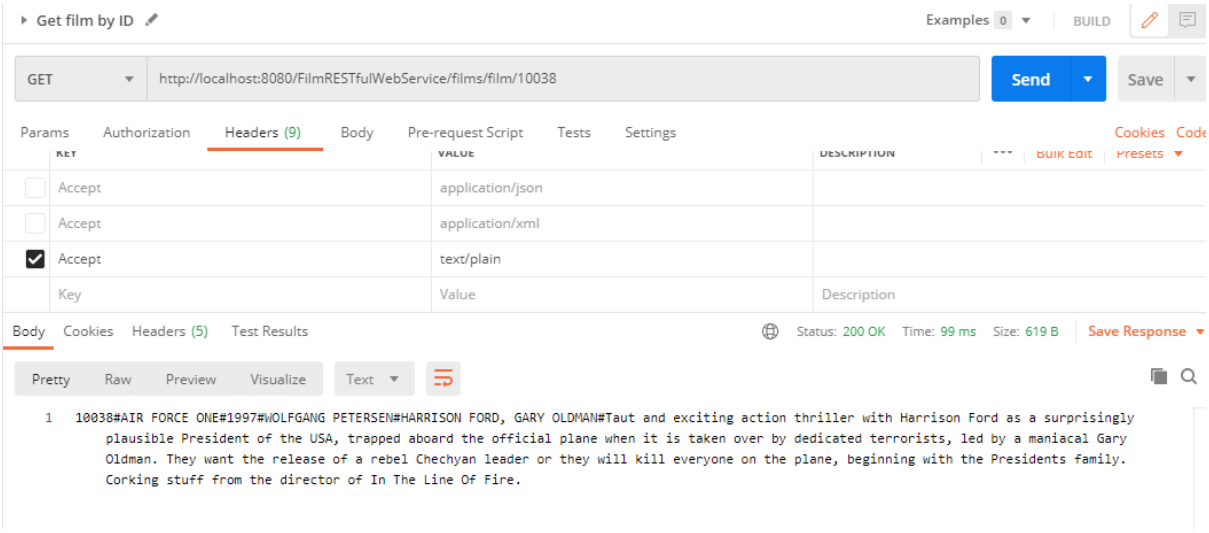
5.7 – Get Film By ID (JSON)



5.8 – Get Film By ID (XML)

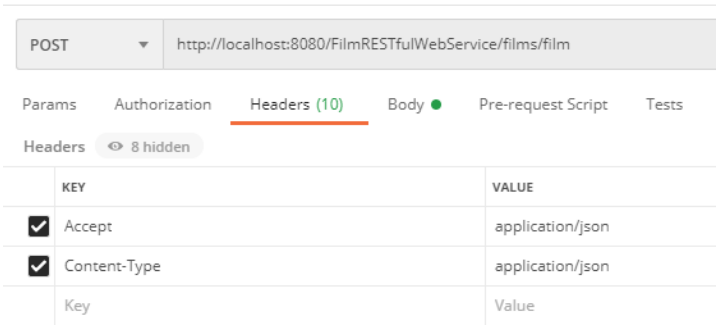


5.9 – Get Film By ID (TEXT)

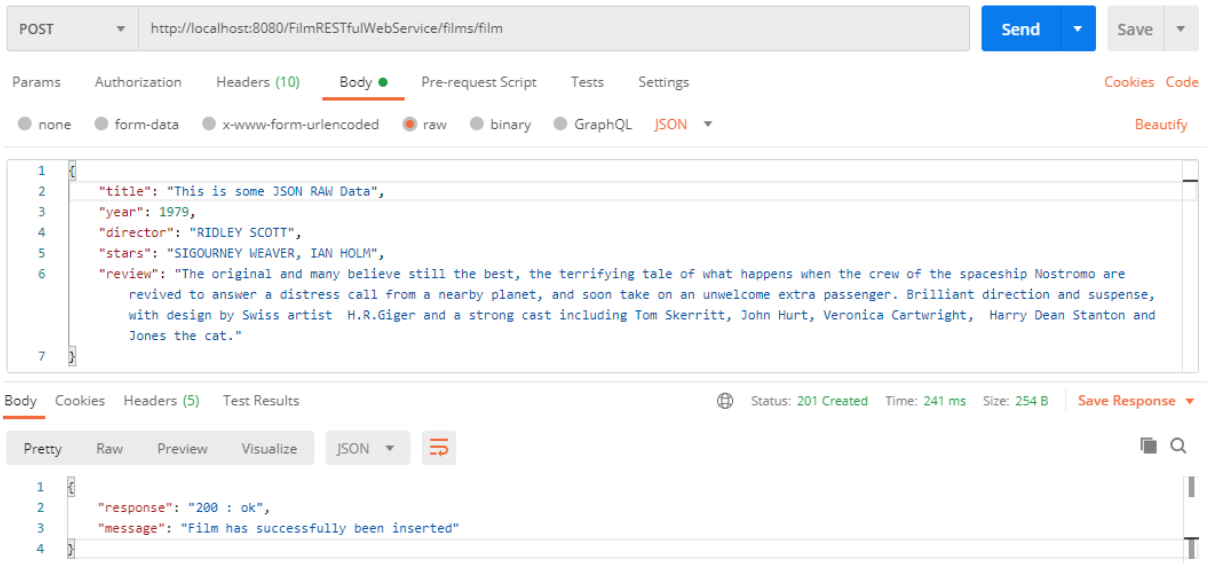


5.10 – Insert Film (JSON)

Headers used:



Body and response:



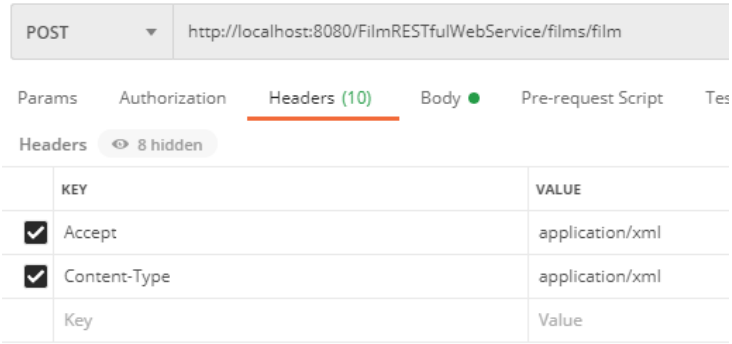
MySQL workbench to confirm it was inserted:

```
SELECT * FROM films WHERE films.title LIKE '%JSON RAW%'
```

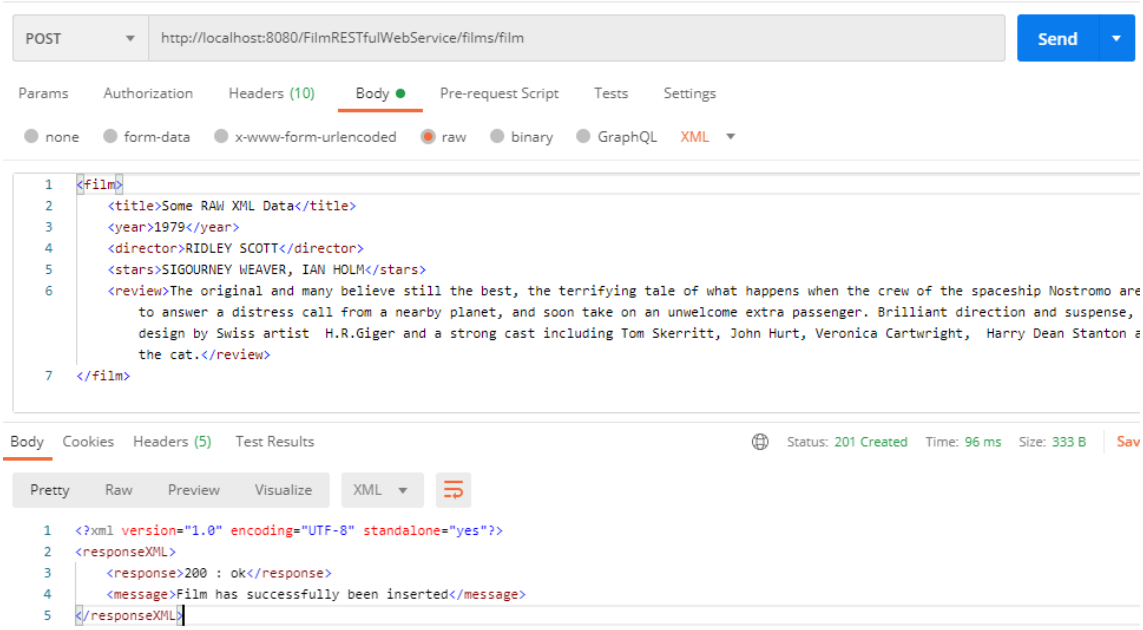
id	title	year	director	stars	review
11347	This is some JSON RAW Data	1979	RIDLEY SCOTT	SIGOURNEY WEAVER, IAN HOLM	The original and many believe still the best, the terrifyin...

5.11 – Insert Film (XML)

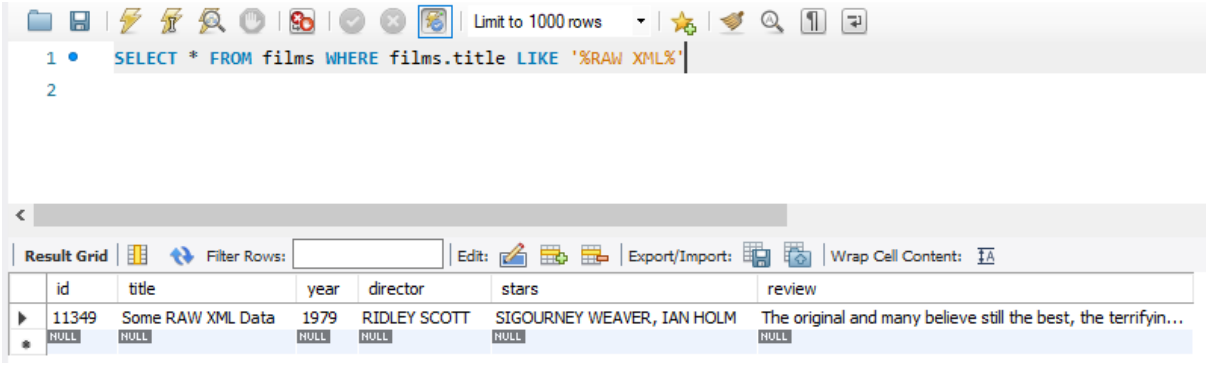
Headers used:



Body and response:

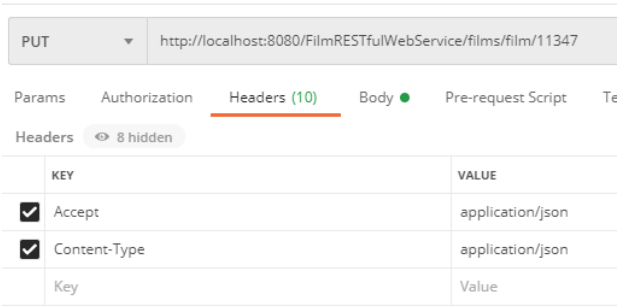


MySQL workbench to confirm it was inserted:

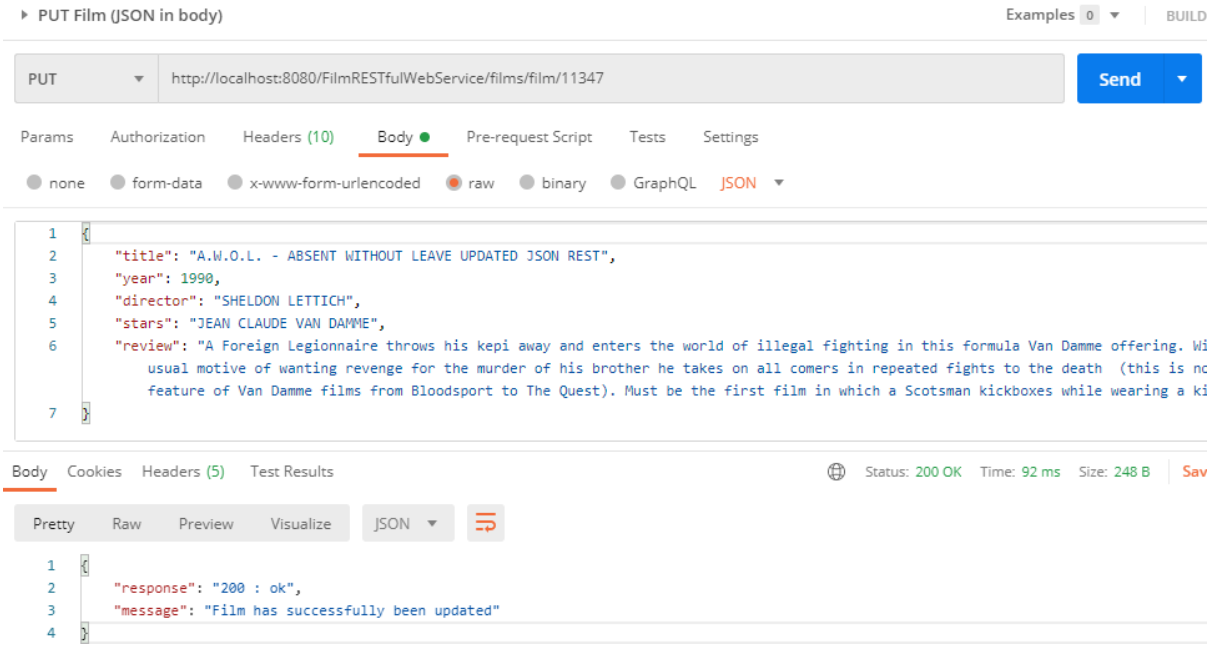


5.12 – Update Film (JSON)

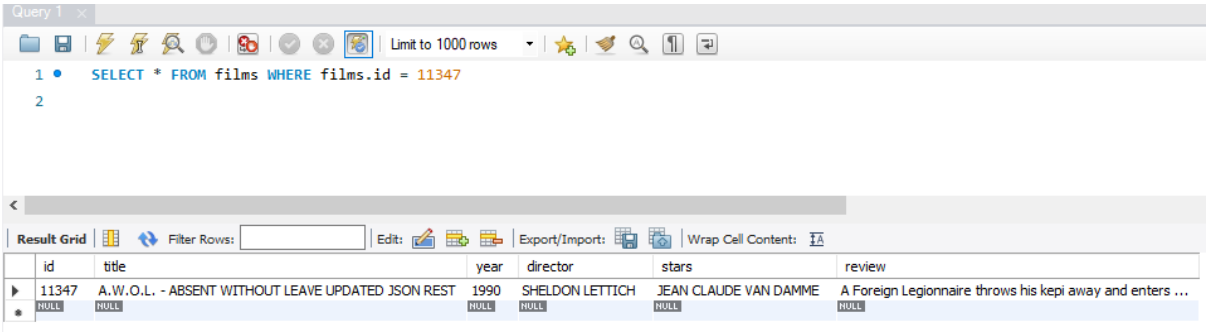
Headers used:



Body and response:

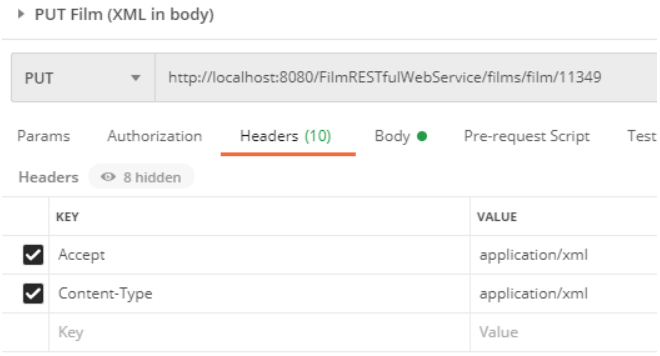


MySQL workbench to confirm it was updated:

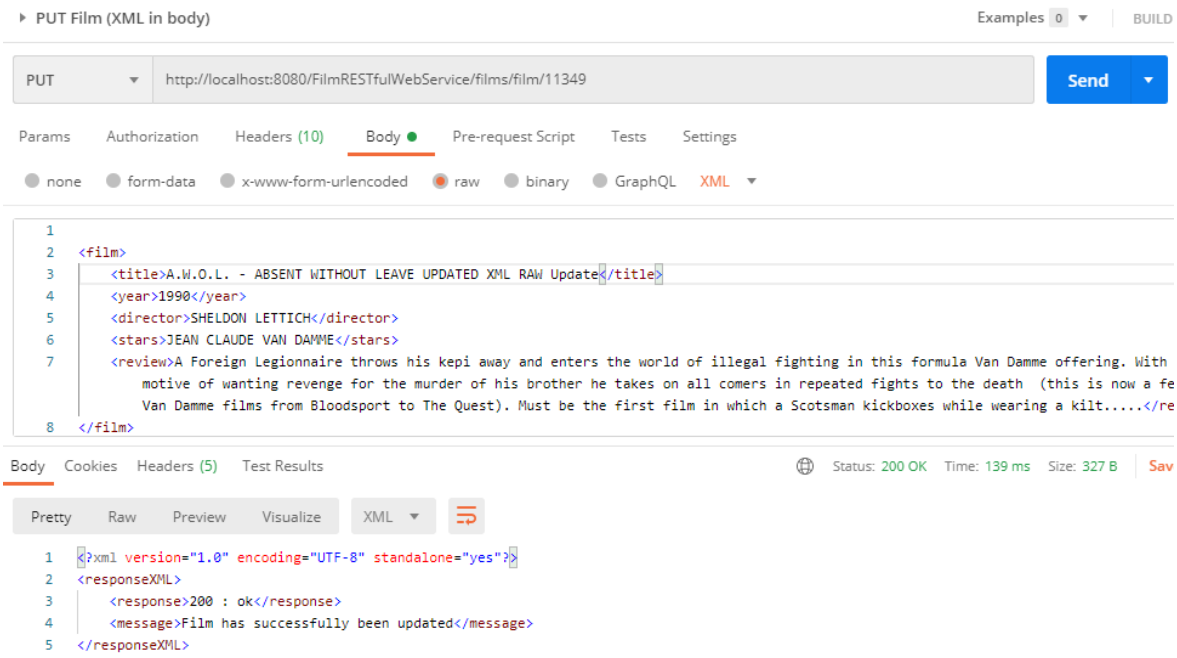


5.13 – Update Film (XML)

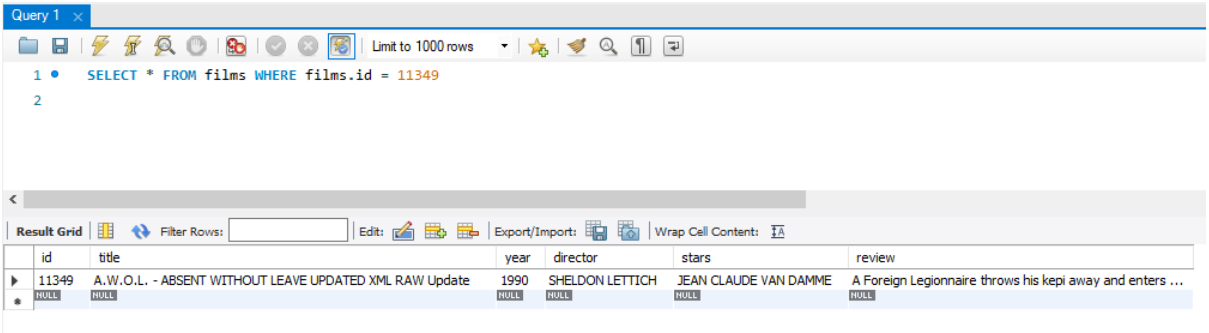
Headers used:



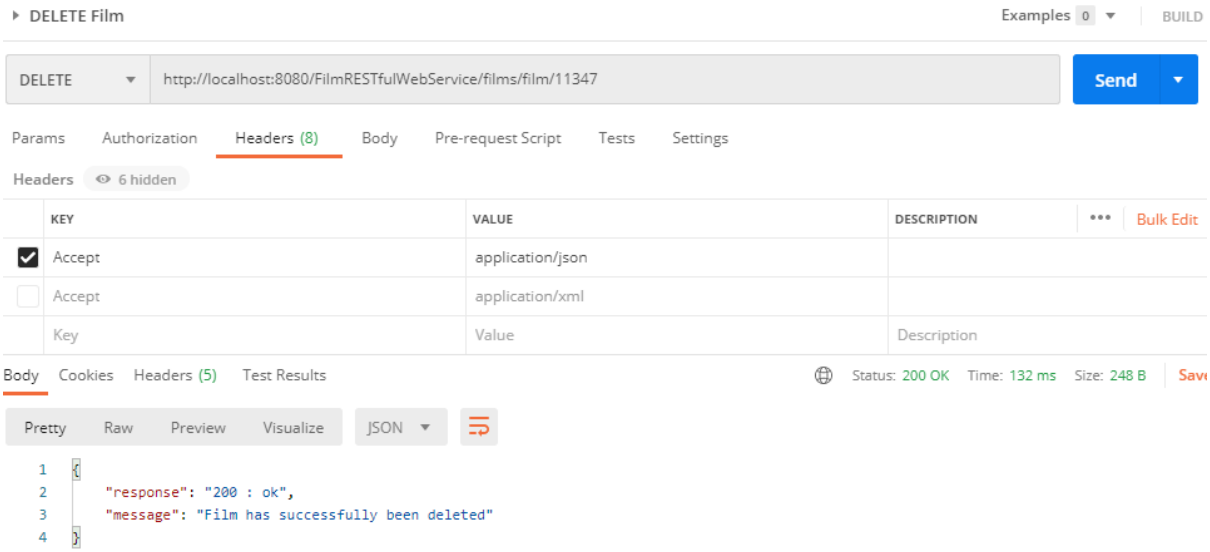
Body and response:



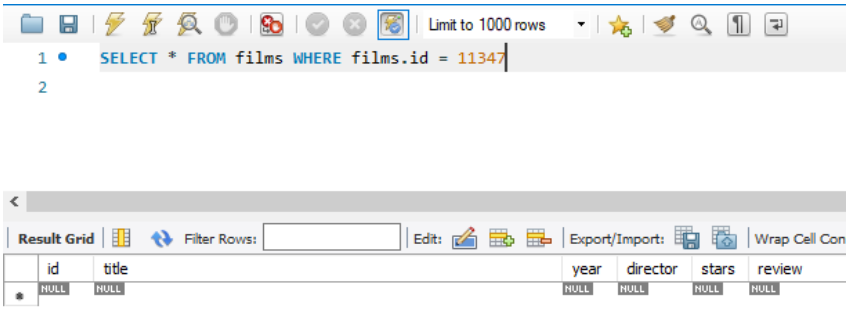
MySQL workbench to confirm it was updated:



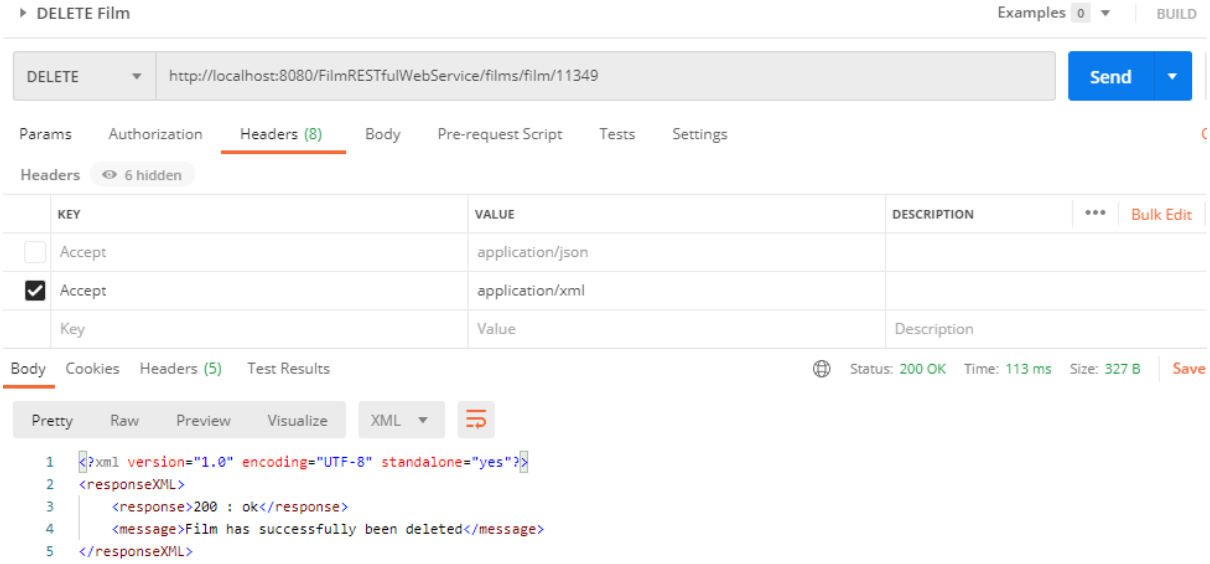
5.14 – Delete Film (JSON Response)



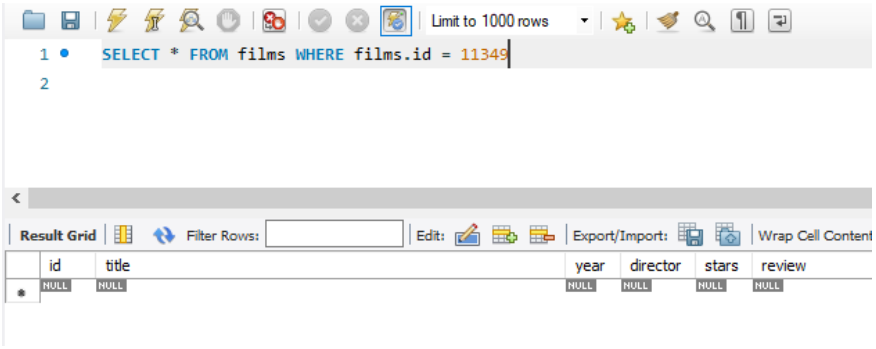
MySQL Workbench to confirm it was deleted:



5.15 – Delete Film (XML Response)



MySQL Workbench to confirm it was deleted:



6.0 – Conclusion

This document has hopefully demonstrated the full functionality that has been implemented in order to satisfy the criteria to the fullest.

- A HTTP API utilizing all CRUD features was implemented and tested
- The data can be returned in all three different formats - XML, JSON (default), TEXT – using GSON and JAXB as one solution (in FilmHTTPWebService), and with Spring for another (FilmRESTfulWebService) – enabling the web service to cope with a range of data objects
- Google App Engine was used with the original code from the FilmHTTPWebService project, and Cloud SQL (a google database) was then added to make it a complete cloud solution. This was tested for all CRUD operations
- A WSDL file and a SOAP client was generated, and has been further discussed in the critical analysis
- A RESTful service was implemented and tested using Spring, with the correct interface for all CRUD operations
- An AJAX based web front-end was created and used for three of the solutions (HTTP, REST, Cloud). This utilized jQuery for AJAX and the majority of the JavaScript code, and used DataTables to dynamically generate the table; this was further styled with bootstrap, providing an “enhanced user interface”

The critical analysis is located in another document and should be grouped together with this and the project solutions. This critical analysis contains a detailed discussion on each of the API's (HTTP, REST, SOAP), WSDL's, the sound Software Engineering techniques, approaches taken throughout this assignment, and more.