CONTENTS

[**Task 1 - Database** 1](#_Toc37872059)

[a) ER Diagram for Memoir database 1](#_Toc37872060)

[b) SQL code for creating and populating tables 1](#_Toc37872061)

[1) MemoirSchema.sql 1](#_Toc37872062)

[2) MemoirInsert.sql 4](#_Toc37872063)

[**Task 2 - RESTful Web Service** 6](#_Toc37872064)

[**Task 3 - Dynamic and Static Queries** 7](#_Toc37872065)

[a) additional REST methods to query all the tables based on each attribute 7](#_Toc37872066)

[1) Cinema Table 7](#_Toc37872067)

[2) Credentials Table 8](#_Toc37872068)

[3) Person Table 9](#_Toc37872069)

[4) Memoir Table 11](#_Toc37872070)

[b) a REST method that enables querying the Person table using a combination of three attributes implemented as a **DYNAMIC** query 13](#_Toc37872071)

[c) a REST method that enables querying the memoir and the cinema tables using a combination of two attributes in the condition where each attribute is from a different table. The query should be a **DYNAMIC** query using an **IMPLICIT JOIN** 14](#_Toc37872072)

[d) a REST method that enables querying the memoir and the cinema tables using a combination of two attributes in the condition where each attribute is from a different table. The query should be a **STATIC** query using an **IMPLICIT JOIN** 15](#_Toc37872073)

[**Task 4 - Advanced REST methods** 16](#_Toc37872074)

[a) a REST method that will accept a person id, a starting date and an ending date and return a list that contains the cinema’s suburbs/postcodes and the total number of movies watched per suburb/postcode during that period 16](#_Toc37872075)

[b) a REST method that will accept a user person id and a year, and return a list that contains the month names and the total number of movies watched per month in that year 17](#_Toc37872076)

[c) a REST method that will accept a user person id and return the name(s), the rating score(s) and release date(s) of the movie(s) with the highest rating score given by that user 19](#_Toc37872077)

[d) a REST method that will accept a person id and return a list of movie names and their release years for those movies that their release year is the same as the year the user watched them. 20](#_Toc37872078)

[e) a REST method that will accept a person id and return a list of movie names and their release years for those movies that the user has watched their remakes as well 21](#_Toc37872079)

[f) a REST method that will accept a user person id and return a list of the movie names, their release dates and rating scores for FIVE movies that have been released in the recent year and have the highest rating score (five top ones) 23](#_Toc37872080)

[**References** 24](#_Toc37872081)

# Task 1 - Database

## ER Diagram for Memoir database:

Memoir database ER Diagram

## SQL code for creating and populating tables:

### MemoirSchema.sql :

CREATE TABLE cinema (

c\_id INTEGER NOT NULL,

c\_name VARCHAR(25) NOT NULL,

c\_location\_postcode CHAR(4) NOT NULL

);

ALTER TABLE cinema ADD CONSTRAINT cinema\_pk PRIMARY KEY ( c\_id );

CREATE TABLE credentials (

Username VARCHAR(30) NOT NULL,

password\_hash VARCHAR(32) NOT NULL,

sign\_up\_date DATE NOT NULL,

p\_id INTEGER NOT NULL

);

ALTER TABLE credentials

ADD CONSTRAINT credentials\_pk PRIMARY KEY ( username );

ALTER TABLE credentials ADD CONSTRAINT p\_id\_unique UNIQUE ( p\_id );

CREATE TABLE memoir (

m\_id INTEGER NOT NULL,

m\_movie\_name VARCHAR(20) NOT NULL,

m\_movie\_release\_date DATE NOT NULL,

m\_watching\_datetime TIMESTAMP NOT NULL,

m\_comment VARCHAR(100),

m\_rating NUMERIC(2, 1) NOT NULL,

p\_id INTEGER NOT NULL,

c\_id INTEGER NOT NULL

);

ALTER TABLE memoir

ADD CONSTRAINT chk\_date

CHECK ( DATE(m\_watching\_datetime) > m\_movie\_release\_date );

ALTER TABLE memoir

ADD CONSTRAINT chk\_rating CHECK ( m\_rating IN (

0,

0.5,

1,

1.5,

2,

2.5,

3,

3.5,

4,

4.5,

5

) );

ALTER TABLE memoir ADD CONSTRAINT memoir\_pk PRIMARY KEY ( m\_id );

ALTER TABLE memoir ADD CONSTRAINT memoir\_alternate\_pk

UNIQUE ( m\_watching\_datetime, p\_id );

CREATE TABLE person (

p\_id INTEGER NOT NULL,

p\_first\_name VARCHAR(10) NOT NULL,

p\_surname VARCHAR(10) NOT NULL,

p\_gender CHAR(1) NOT NULL,

p\_dob DATE NOT NULL,

p\_address VARCHAR(30) NOT NULL,

p\_state CHAR(3) NOT NULL,

p\_postcode CHAR(4) NOT NULL

);

ALTER TABLE person

ADD CONSTRAINT chk\_gender CHECK ( p\_gender IN (

'F',

'M'

) );

ALTER TABLE person

ADD CONSTRAINT chk\_state CHECK ( p\_state IN (

'ACT',

'NSW',

'NT',

'QLD',

'SA',

'TAS',

'VIC',

'WA'

) );

ALTER TABLE person ADD CONSTRAINT person\_pk PRIMARY KEY ( p\_id );

ALTER TABLE credentials

ADD CONSTRAINT credentials\_person\_fk FOREIGN KEY ( p\_id )

REFERENCES person ( p\_id )

ON DELETE CASCADE;

ALTER TABLE memoir

ADD CONSTRAINT memoir\_cinema\_fk FOREIGN KEY ( c\_id )

REFERENCES cinema ( c\_id );

ALTER TABLE memoir

ADD CONSTRAINT memoir\_person\_fk FOREIGN KEY ( p\_id )

REFERENCES person ( p\_id )

ON DELETE CASCADE;

### MemoirInsert.sql :

INSERT INTO person VALUES (

1,'Roger','Mason','M', DATE ('1982-02-25'),'17 Shepparson Avenue','VIC','3163');

INSERT INTO person VALUES (

2,'Serena','Petrova','F', DATE ('1975-12-06'),'160 Grote Street','SA','5000');

INSERT INTO person VALUES (

3,'Mark','Williams','M', DATE ('1993-07-30'),'2 Chippendale Way','NSW','2008');

INSERT INTO credentials VALUES (

'rogermason82','cbf4d9fb4123b06b28f583ff81567403', DATE ('2019-11-13'),1);

INSERT INTO credentials VALUES (

'serenapetrova75','89962fdbdec468b59a8c00842c105586', DATE ('2019-12-26'),2);

INSERT INTO credentials VALUES (

'markwilliams93','8d7f20caa2345fcd3e03e0bb6918ed3b', DATE ('2020-01-08'),3);

INSERT INTO cinema VALUES (1,'HOYTS Broadway','2007');

INSERT INTO cinema VALUES (2,'HOYTS Chadstone','3148');

INSERT INTO cinema VALUES (3,'Village cinemas Rivoli','3123');

INSERT INTO cinema VALUES (4,'GU Film House','5000');

INSERT INTO cinema VALUES (5,'Luna cinemas','6007');

INSERT INTO memoir VALUES (1,'Titanic', DATE ('1997-11-01'), TIMESTAMP ('20080912163000'),'Fantastic Film! Amazing!',4.5,1,2);

INSERT INTO memoir VALUES (2,'Titanic', DATE ('1997-11-01'), TIMESTAMP ('20131001123000'),'Reviewed the classic film after five years, still sensational!',5,1,4);

INSERT INTO memoir VALUES (3,'Oceans 8', DATE ('2018-06-08'), TIMESTAMP ('20180617160000'),'Pretty Good!',4.5,1,3);

INSERT INTO memoir VALUES (4,'The Intern', DATE ('2015-09-25'), TIMESTAMP ('20190725093000'),'Interesting Story!',4,1,4);

INSERT INTO memoir VALUES (5,'Love & Other Drugs', DATE ('2010-11-04'), TIMESTAMP ('20101201110000'),'Too sleepy to catch up. Hope to have another try!',3.5,1,5);

INSERT INTO memoir VALUES (6,'Love & Other Drugs', DATE ('2010-11-04'), TIMESTAMP ('20110109160000'),'What a moving story!!!',4.5,1,3);

INSERT INTO memoir VALUES (7,'Becoming Jane', DATE ('2007-03-09'), TIMESTAMP ('20190124153000'),'Perfect performance!',5,1,2);

INSERT INTO memoir VALUES (8,'One Day', DATE ('2011-08-19'), TIMESTAMP ('20180228190000'),'So boring!!',1.5,1,1);

INSERT INTO memoir VALUES (9,'Me Before You', DATE ('2016-05-23'), TIMESTAMP ('20190522200000'),'Just so so...',3,1,3);

INSERT INTO memoir VALUES (10,'Valentines Day', DATE ('2010-01-28'), TIMESTAMP ('20151119203000'),'Far from realistic.',2.5,1,3);

INSERT INTO memoir VALUES (11,'The Lion King', DATE ('2019-07-09'), TIMESTAMP ('20191017160000'),'Perfect! Even better than the 1994 version!',5,1,3);

INSERT INTO memoir VALUES (12,'The Lion King', DATE ('1994-06-24'), TIMESTAMP ('19950128190000'),'Lots of fun!',4.5,1,5);

INSERT INTO memoir VALUES (13,'Onward', DATE ('2020-03-06'), TIMESTAMP ('20200317180000'),'Ok but some settings are not consistent!',3.5,1,3);

INSERT INTO memoir VALUES (14,'Dolittle', DATE ('2020-01-17'), TIMESTAMP ('20200124133000'),'Great movie!',4.5,1,2);

INSERT INTO memoir VALUES (15,'Underwater', DATE ('2020-01-10'), TIMESTAMP ('20200201103000'),'Not worthy at all!',0.5,1,2);

INSERT INTO memoir VALUES (16,'Birds of Prey', DATE ('2020-02-07'), TIMESTAMP ('20200303173000'),'A good story!',4,1,3);

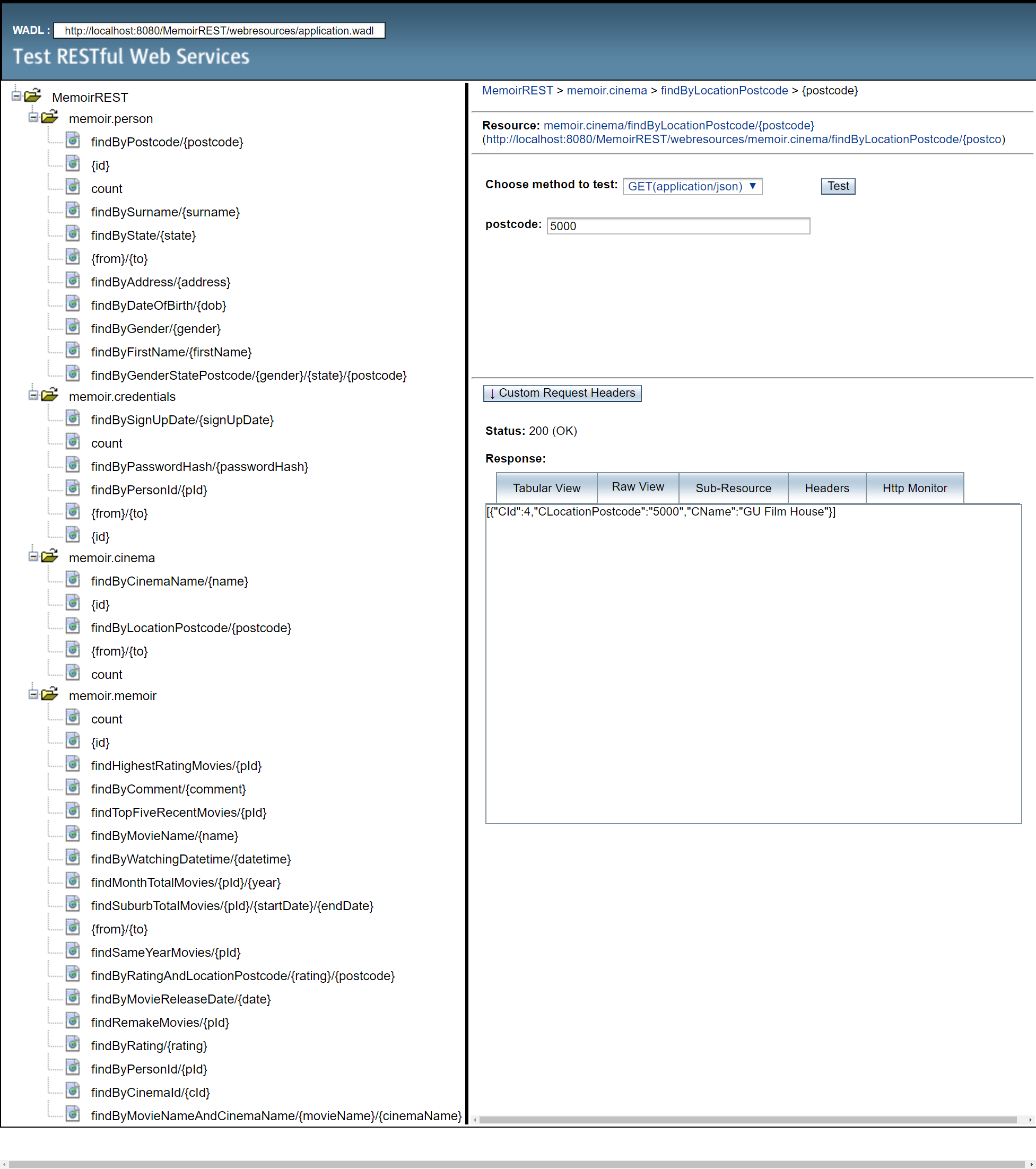
INSERT INTO memoir VALUES (17,'The Paragraph', DATE ('2020-02-14'), TIMESTAMP ('20200229120000'),'Not too bad...',3,1,2);

INSERT INTO memoir VALUES (18,'Bad Boys for Life', DATE ('2020-01-17'), TIMESTAMP ('20200118193000'),'Quite funny! Very Good!',5,1,2);

INSERT INTO memoir VALUES (19,'Sonic the Hedgehog', DATE ('2020-02-14'), TIMESTAMP ('20200308150000'),'Honestly not live up to expectation!',2,1,3);

INSERT INTO memoir VALUES (20,'Like a Boss', DATE ('2020-01-10'), TIMESTAMP ('20200131210000'),'So so...',2.5,1,2);

# Task 2 - RESTful Web Service



the browser screenshot showing all methods

# Task 3 - Dynamic and Static Queries

## additional REST methods to **query all the tables based on each attribute** that the table has.

### Cinema Table

#### Named query by cinema name

@GET

@Path("findByCinemaName/{name}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Cinema> findByCinemaName(@PathParam("name") String name) {

Query q = em.createNamedQuery("Cinema.findByCName");

q.setParameter("cName", name);

return q.getResultList();

}

#### Named query by location postcode

@GET

@Path("findByLocationPostcode/{postcode}")

@Produces(MediaType.APPLICATION\_JSON)

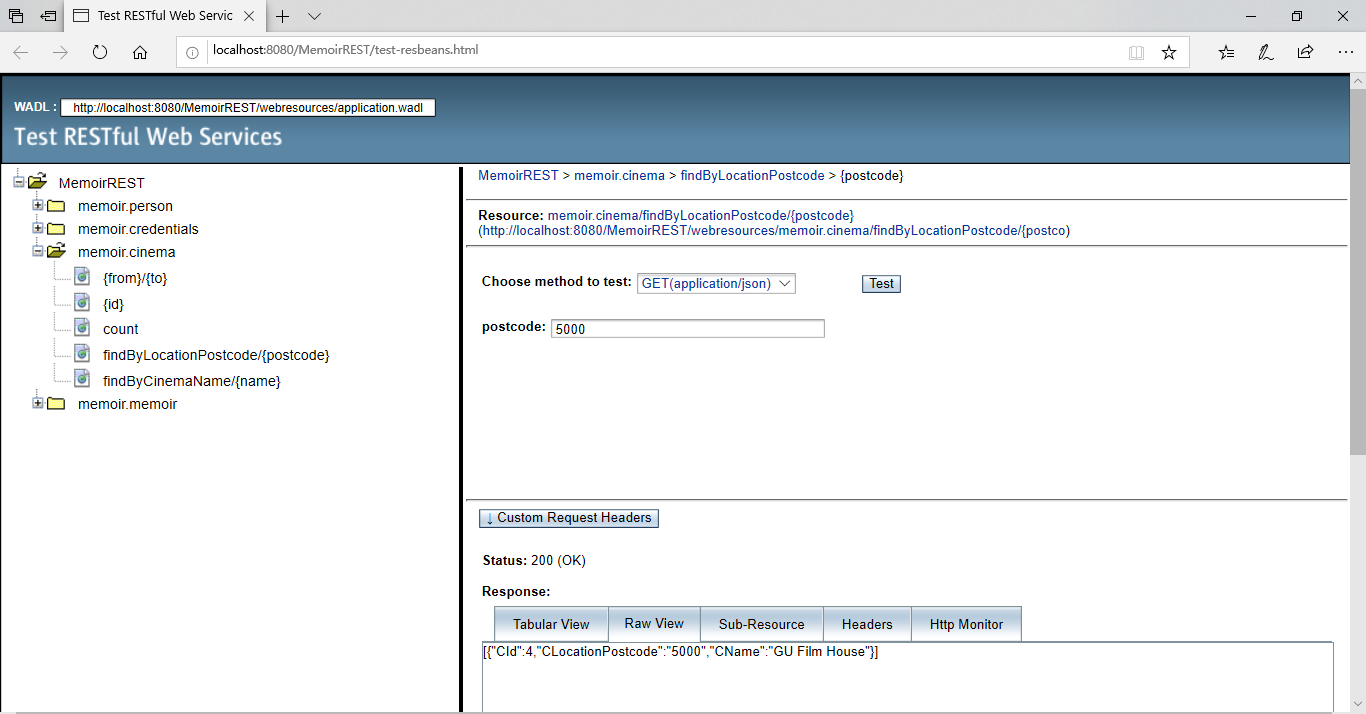
public List<Cinema> findByLocationPostcode(@PathParam("postcode") String postcode) {

Query q = em.createNamedQuery("Cinema.findByCLocationPostcode");

q.setParameter("cLocationPostcode", postcode);

return q.getResultList();

}



screenshot of named query by cinema location postcode

### Credentials Table

#### Named query by password hash

@GET

@Path("findByPasswordHash/{passwordHash}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Credentials> findByPasswordHash(@PathParam("passwordHash") String pw) {

Query q = em.createNamedQuery("Credentials.findByPasswordHash");

q.setParameter("passwordHash", pw);

return q.getResultList();

}

#### Named query by sign-up date

@GET

@Path("findBySignUpDate/{signUpDate}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Credentials> findBySignUpDate(@PathParam("signUpDate") Date signDate) {

Query q = em.createNamedQuery("Credentials.findBySignUpDate");

q.setParameter("signUpDate", signDate);

return q.getResultList();

}

#### Named query by person id

@GET

@Path("findByPersonId/{pId}")

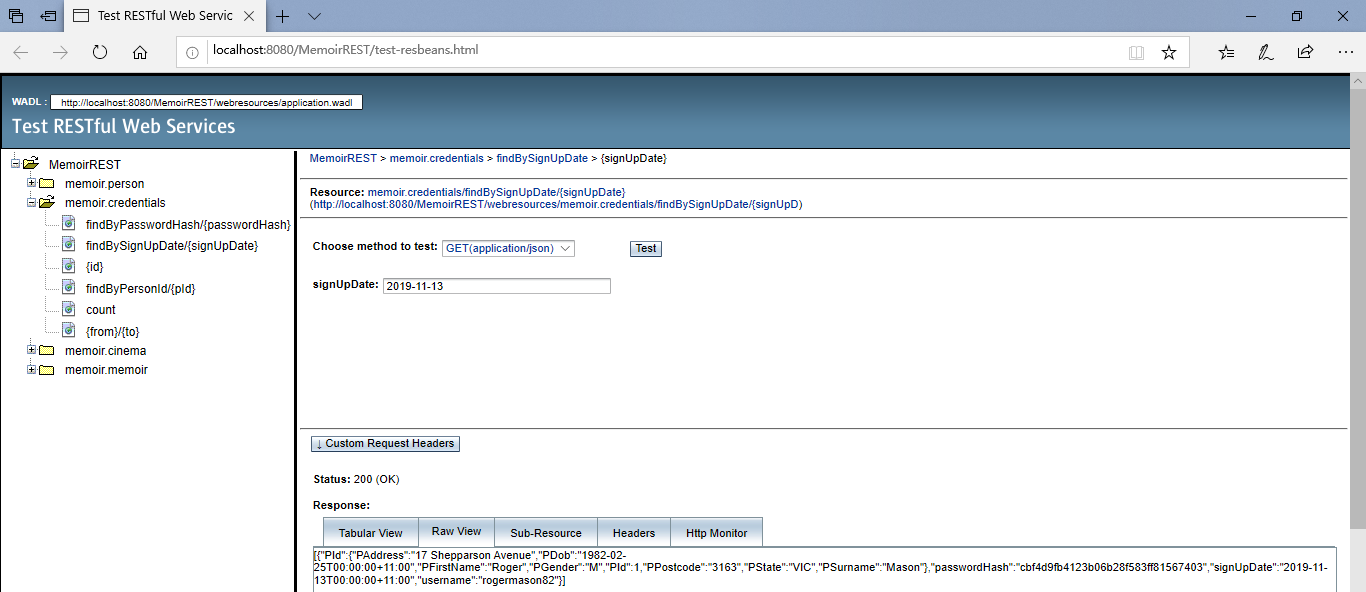
@Produces(MediaType.APPLICATION\_JSON)

public List<Credentials> findByPId(@PathParam("pId") Integer pId) {

Query q = em.createNamedQuery("Credentials.findByPId");

q.setParameter("pId", pId);

return q.getResultList();

 }

screenshot of named query by sign-up date

### Person Table

#### Named query by first name

@GET

@Path("findByFirstName/{firstName}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findByFirstName(@PathParam("firstName") String fName) {

Query q = em.createNamedQuery("Person.findByPFirstName");

q.setParameter("pFirstName", fName);

return q.getResultList();

}

#### Named query by surname

@GET

@Path("findBySurname/{surname}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findBySurname(@PathParam("surname") String sName) {

Query q = em.createNamedQuery("Person.findByPSurname");

q.setParameter("pSurname", sName);

return q.getResultList();

}

#### Named query by gender

@GET

@Path("findByGender/{gender}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findByGender(@PathParam("gender") String gender) {

Query q = em.createNamedQuery("Person.findByPGender");

q.setParameter("pGender", gender);

return q.getResultList();

}

#### Named query by date of birth

@GET

@Path("findByDateOfBirth/{dob}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findByDateOfBirth(@PathParam("dob") Date dob) {

Query q = em.createNamedQuery("Person.findByPDob");

q.setParameter("pDob", dob);

return q.getResultList();

}

#### Named query by address

@GET

@Path("findByAddress/{address}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findAddress(@PathParam("address") String address) {

Query q = em.createNamedQuery("Person.findByPAddress");

q.setParameter("pAddress", address);

return q.getResultList();

}

#### Named query by date of state

@GET

@Path("findByState/{state}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findByState(@PathParam("state") String state) {

Query q = em.createNamedQuery("Person.findByPState");

q.setParameter("pState", state);

return q.getResultList();

}

#### Named query by date of postcode

@GET

@Path("findByPostcode/{postcode}")

@Produces(MediaType.APPLICATION\_JSON)

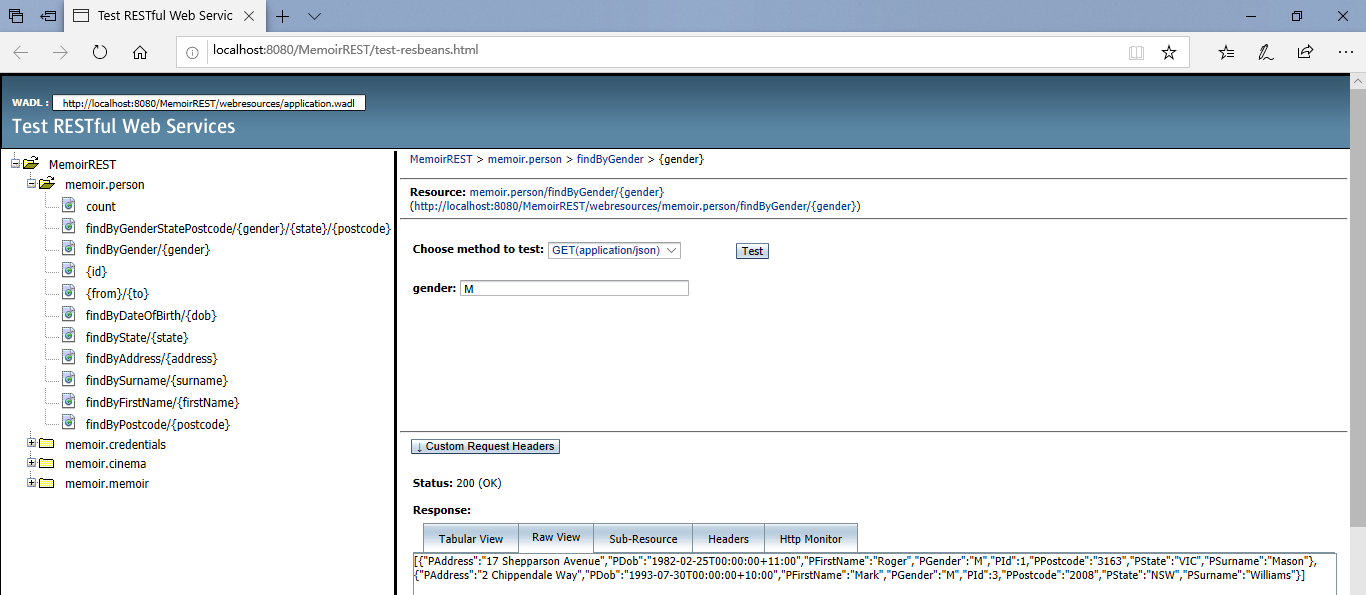
public List<Person> findByPostcode(@PathParam("postcode") String postcode) {

Query q = em.createNamedQuery("Person.findByPPostcode");

q.setParameter("pPostcode", postcode);

return q.getResultList();

}



screenshot of named query by gender

### Memoir Table

#### Named query by movie name

@GET

@Path("findByMovieName/{name}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByMovieName(@PathParam("name") String name) {

Query q = em.createNamedQuery("Memoir.findByMMovieName");

q.setParameter("mMovieName", name);

return q.getResultList();

}

#### Named query by movie release date

@GET

@Path("findByMovieReleaseDate/{date}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByMovieReleaseDate(@PathParam("date") Date date) {

Query q = em.createNamedQuery("Memoir.findByMMovieReleaseDate");

q.setParameter("mMovieReleaseDate", date);

return q.getResultList();

}

#### Named query by watching datetime

@GET

@Path("findByWatchingDatetime/{datetime}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByWatchingDatetime(@PathParam("datetime") Timestamp datetime) {

Query q = em.createNamedQuery("Memoir.findByMWatchingDatetime");

q.setParameter("mWatchingDatetime", datetime);

return q.getResultList();

}

#### Named query by comment

@GET

@Path("findByComment/{comment}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByComment(@PathParam("comment") String comment) {

Query q = em.createNamedQuery("Memoir.findByMComment");

q.setParameter("mComment", comment);

return q.getResultList();

}

#### Named query by rating

@GET

@Path("findByRating/{rating}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByRating(@PathParam("rating") BigDecimal rating) {

Query q = em.createNamedQuery("Memoir.findByMRating");

q.setParameter("mRating", rating);

return q.getResultList();

}

#### Named query by cinema id

@GET

@Path("findByCinemaId/{cId}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByCinemaId(@PathParam("cId") Integer cId) {

Query q = em.createNamedQuery("Memoir.findByCId");

q.setParameter("cId", cId);

return q.getResultList();

}

#### Named query by person id

@GET

@Path("findByPersonId/{pId}")

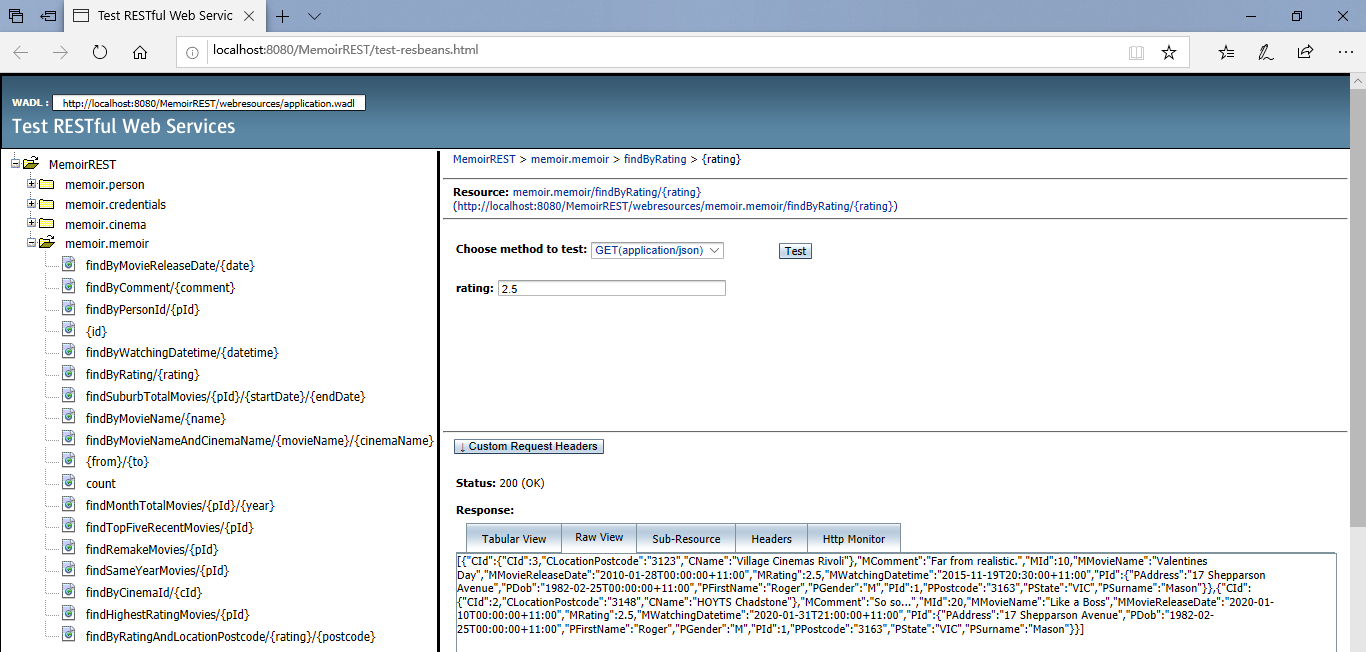
@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByPersonId(@PathParam("pId") Integer pId) {

Query q = em.createNamedQuery("Memoir.findByPId");

q.setParameter("pId", pId);

return q.getResultList();

 }

screenshot of named query by rating

## a REST method that enables **querying the Person table using a combination of three attributes** implemented as a **DYNAMIC** query.

The following REST method enables querying the Person table using a combination of gender, state and postcode, implemented as a DYNAMIC query.

@GET

@Path("findByGenderStatePostcode/{gender}/{state}/{postcode}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Person> findByGenderStatePostcode(

@PathParam("gender") String gender,

@PathParam("state") String state,

@PathParam("postcode") String postcode) {

TypedQuery<Person> q = em.createQuery(

"SELECT p FROM Person p "

+ "WHERE p.pGender = :gender "

+ "AND p.pState = :state "

+ "AND p.pPostcode = :postcode",Person.class);

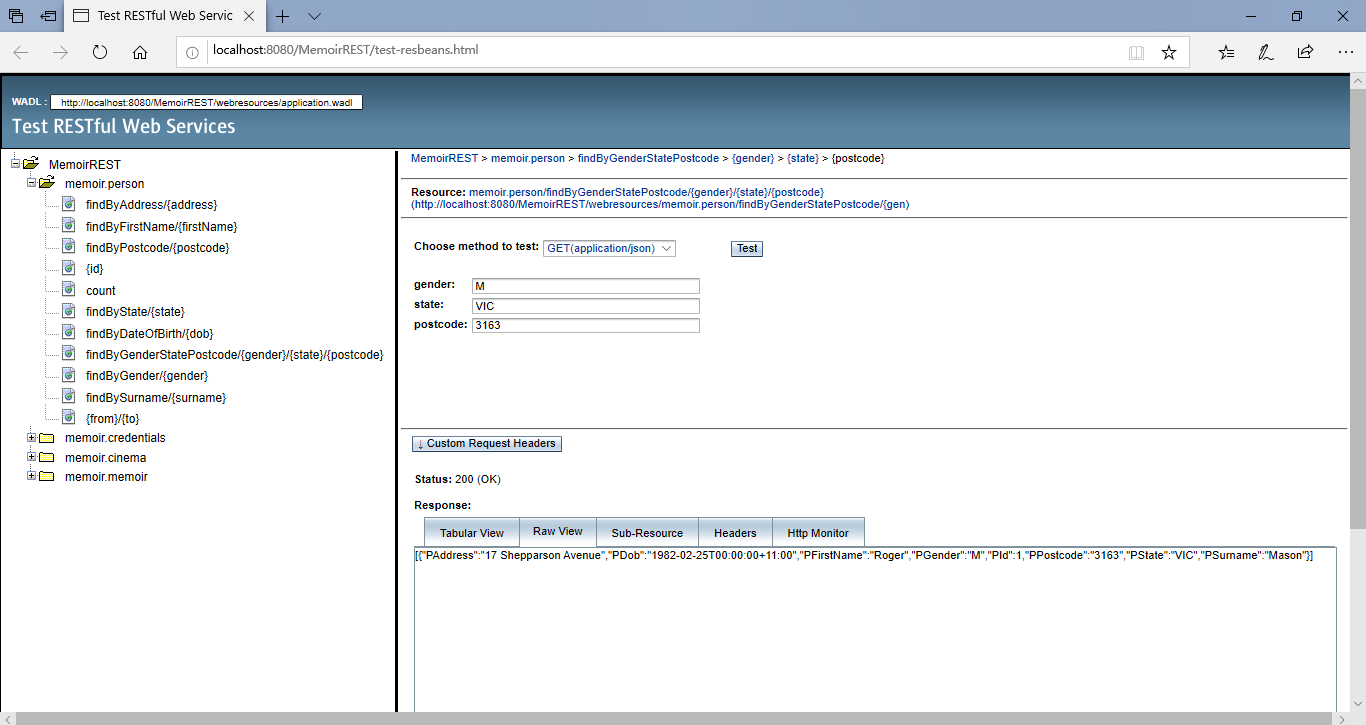
q.setParameter("gender", gender);

q.setParameter("state", state);

q.setParameter("postcode", postcode);

return q.getResultList();

}



screenshot of dynamic query by gender, state and postcode

## a REST method that enables **querying the memoir and the cinema tables using a combination of two attributes in the condition where each attribute is from a different table**. The query should be a **DYNAMIC query using an IMPLICIT JOIN**.

The following REST method enables querying the memoir and the cinema tables using a combination of movie name and cinema, implemented as a DYNAMIC query with an implicit join.

@GET

@Path("findByMovieNameAndCinemaName/{movieName}/{cinemaName}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByMovieNameAndCinemaName(

@PathParam("movieName") String mName,

@PathParam("cinemaName") String cName) {

TypedQuery<Memoir> q = em.createQuery(

"SELECT m FROM Memoir m "

+ "WHERE m.mMovieName = :mName "

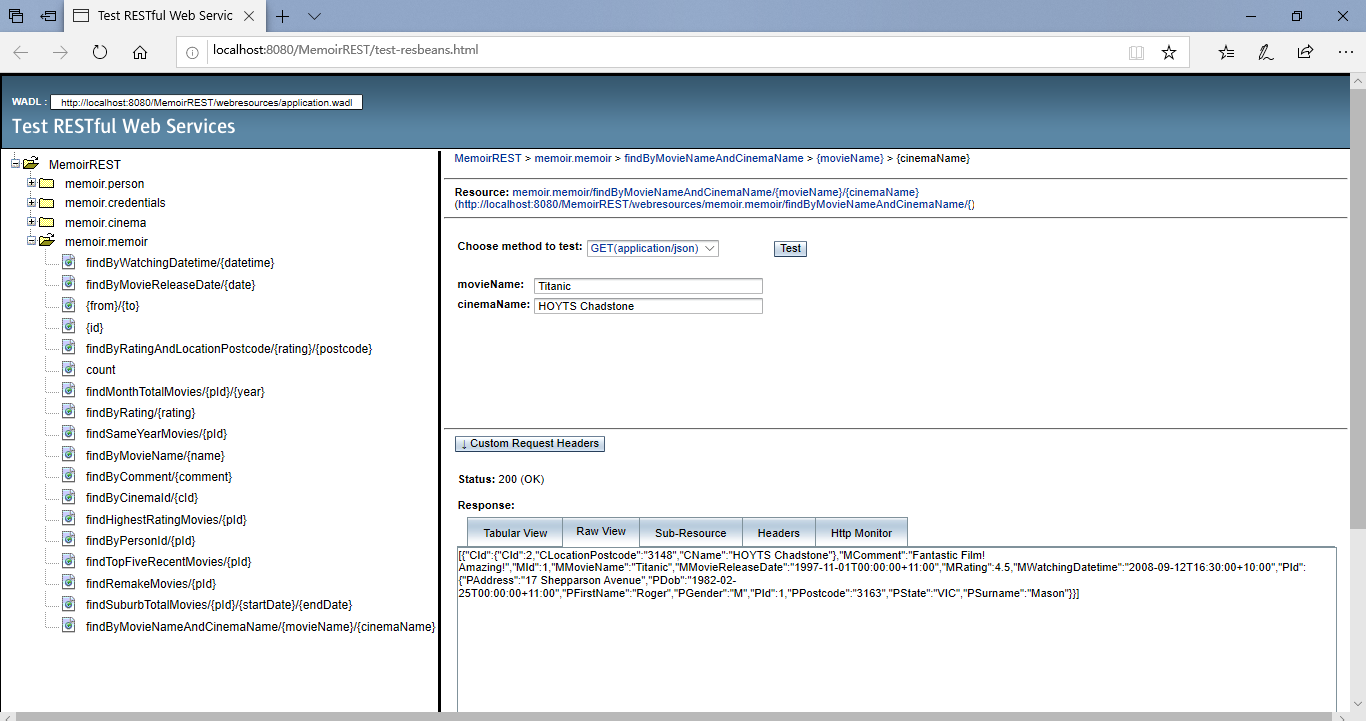
+ "AND m.cId.cName = :cName", Memoir.class);

q.setParameter("mName", mName);

q.setParameter("cName", cName);

return q.getResultList();

}



screenshot of dynamic query by movie name and cinema name

## a REST method that enables **querying the memoir and the cinema tables using a combination of two attributes in the condition where each attribute is from a different table**. The query should be a **STATIC query using an IMPLICIT JOIN**.

The following REST method enables querying the memoir and the cinema tables using a combination of rating and cinema location postcode, implemented as a STATIC query with an implicit join.

@GET

@Path("findByRatingAndLocationPostcode/{rating}/{postcode}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Memoir> findByRatingAndLocationPostcode(

@PathParam("rating") BigDecimal rating,

@PathParam("postcode") String postcode) {

Query q = em.createNamedQuery("Memoir.findByRatingAndLocationPostcode");

q.setParameter("mRating", rating);

q.setParameter("cPostcode", postcode);

return q.getResultList();

}

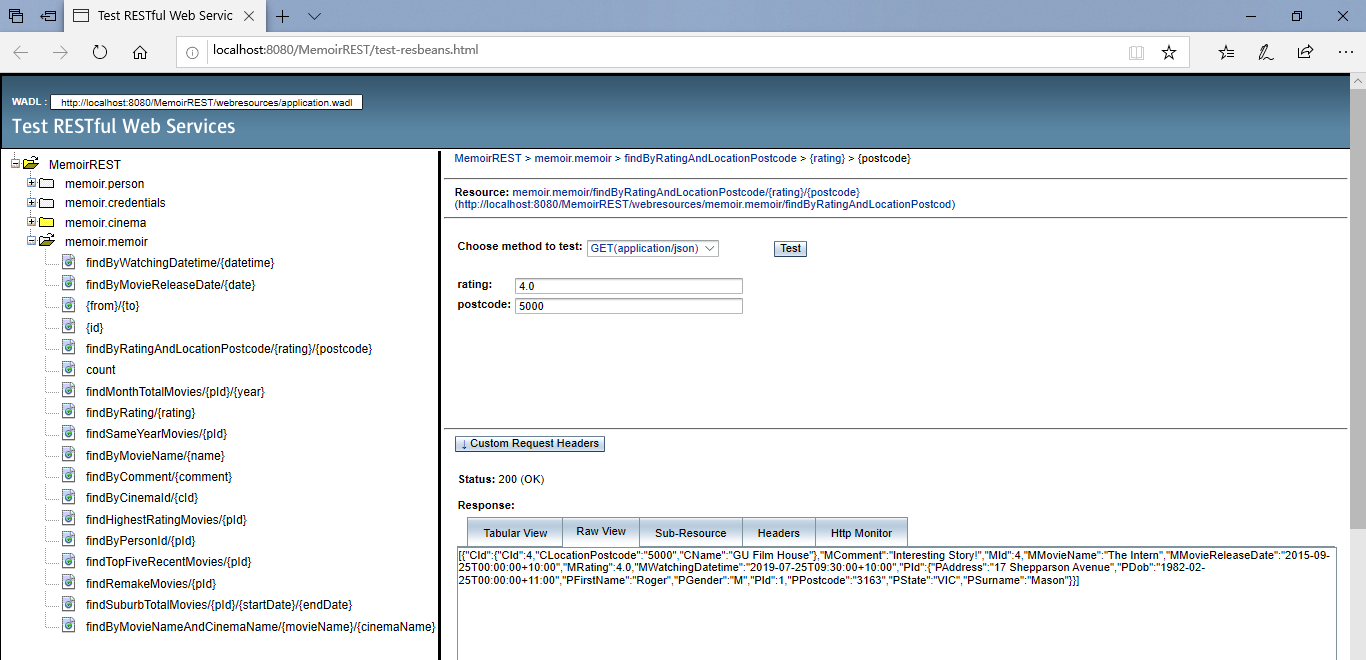
The corresponding code for the NamedQuery in the entity class:

@NamedQueries({

…

, @NamedQuery(name = "Memoir.findByRatingAndLocationPostcode",

query = "SELECT m FROM Memoir m WHERE m.mRating = :mRating AND m.cId.cLocationPostcode = :cPostcode")})



screenshot of named query by movie name and cinema name

# Task 4 - Advanced REST methods

## a REST method that will accept a person id, a starting date and an ending date and return a list that contains the cinema’s suburbs/postcodes and the total number of movies watched per suburb/postcode during that period.

The following REST method accepts a person id, a starting date, an ending date and enables returning a list that contains the cinema’s postcodes and the total number of movies watched per postcode implemented by using JsonObject to create JSON objects and JsonArrayBuilder to create the JsonArray object.

@GET

@Path("findSuburbTotalMovies/{pId}/{startDate}/{endDate}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findSuburbTotalMovies(

@PathParam("pId") Integer pId,

@PathParam("startDate") Date startDate,

@PathParam("endDate") Date endDate) {

Query q = em.createQuery(

"SELECT m.cId.cLocationPostcode, COUNT(m.mId) "

+ "FROM Memoir m "

+ "WHERE m.mWatchingDatetime <= :endDate "

+ "AND m.mWatchingDatetime >= :startDate "

+ "AND m.pId.pId = :pId "

+ "GROUP BY m.cId.cLocationPostcode", Object[].class);

q.setParameter("pId", pId);

q.setParameter("startDate", startDate);

q.setParameter("endDate", endDate);

List<Object[]> queryList = q.getResultList();

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

for (Object[] row : queryList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("SuburbPostcode", (String) row[0])

.add("TotalMovieWatched", (long) row[1]).build();

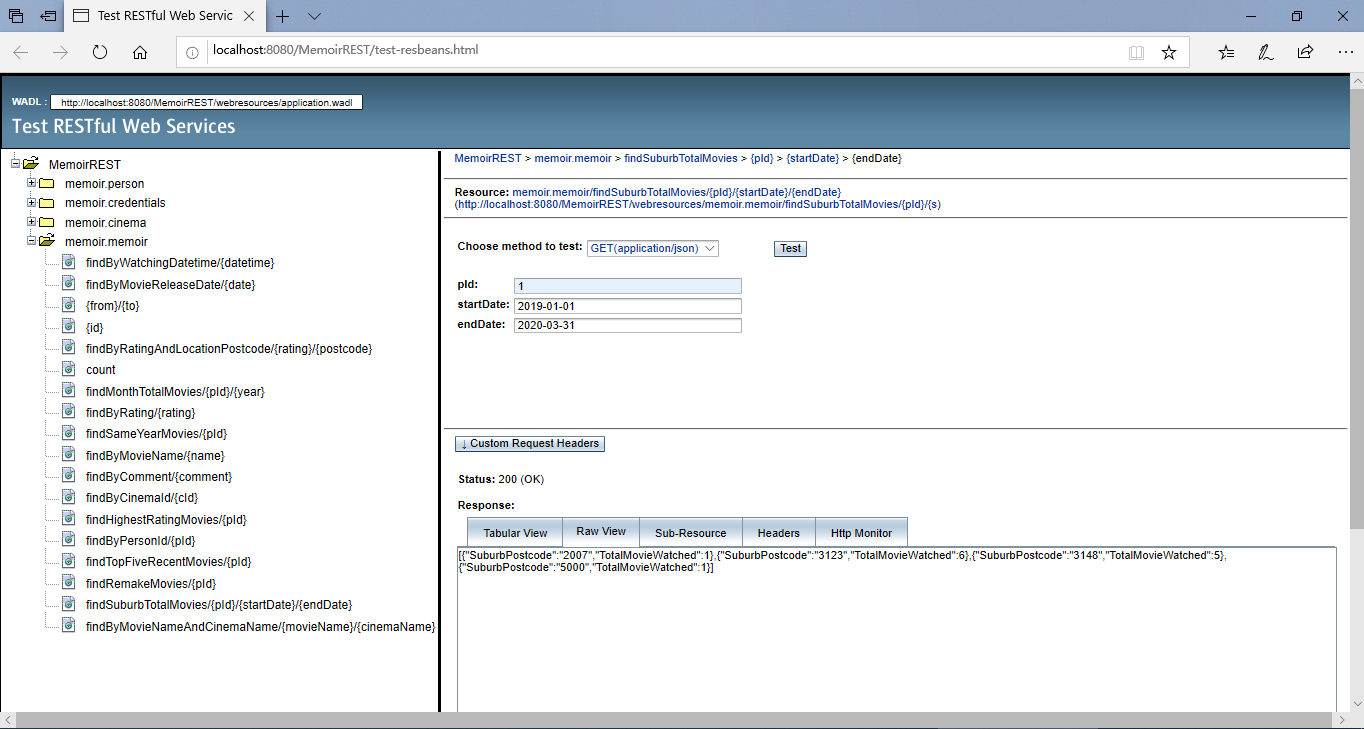
arrayBuilder.add(jsonObject);

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}



screenshot of task 4 a) query test result

## a REST method that will accept a user person id and a year, and return a list that contains the month names and the total number of movies watched per month in that year.

In this method, the first step is to get the original “queryList” which only contains the months with movie watching records. The second step is to build an “outcomeList” which contains all the months from 1 to 12 where those months without any watching record will return 0. The final step is to use JsonObject to create JSON objects and JsonArrayBuilder to create the JsonArray object with the help of intToMonth function to convert month numbers to month names.

@GET

@Path("findMonthTotalMovies/{pId}/{year}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findMonthTotalMovies(

@PathParam("pId") Integer pId,

@PathParam("year") Integer year) {

Query q = em.createQuery(

"SELECT EXTRACT(MONTH FROM m.mWatchingDatetime), COUNT(m.mId) "

+ "FROM Memoir m "

+ "WHERE EXTRACT(YEAR FROM m.mWatchingDatetime) = :year "

+ "AND m.pId.pId = :pId "

+ "GROUP BY EXTRACT(MONTH FROM m.mWatchingDatetime)", Object[].class);

q.setParameter("pId", pId);

q.setParameter("year", year);

List<Object[]> queryList = q.getResultList();

List<Integer[]> outcomeList = new ArrayList<Integer[]>();

for (int mm = 1; mm <= 12; mm++) {

Integer[] singleMonth = new Integer[2];

singleMonth[0] = mm;

singleMonth[1] = 0;

outcomeList.add(singleMonth);

for (Object[] row : queryList) {

if ((int) row[0] == mm) {

singleMonth[1] = ((Long) row[1]).intValue();

break;

}

}

}

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

for (Integer[] row : outcomeList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("Month", intToMonth((int) row[0]))

.add("TotalMoviesWatched", (int) row[1]).build();

arrayBuilder.add(jsonObject);

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}

public String intToMonth(int i) {

String month = "";

switch(i) {

case 1: month = "January"; break;

case 2: month = "February"; break;

case 3: month = "March"; break;

case 4: month = "April"; break;

case 5: month = "May"; break;

case 6: month = "June"; break;

case 7: month = "July"; break;

case 8: month = "August"; break;

case 9: month = "September"; break;

case 10: month = "October"; break;

case 11: month = "November"; break;

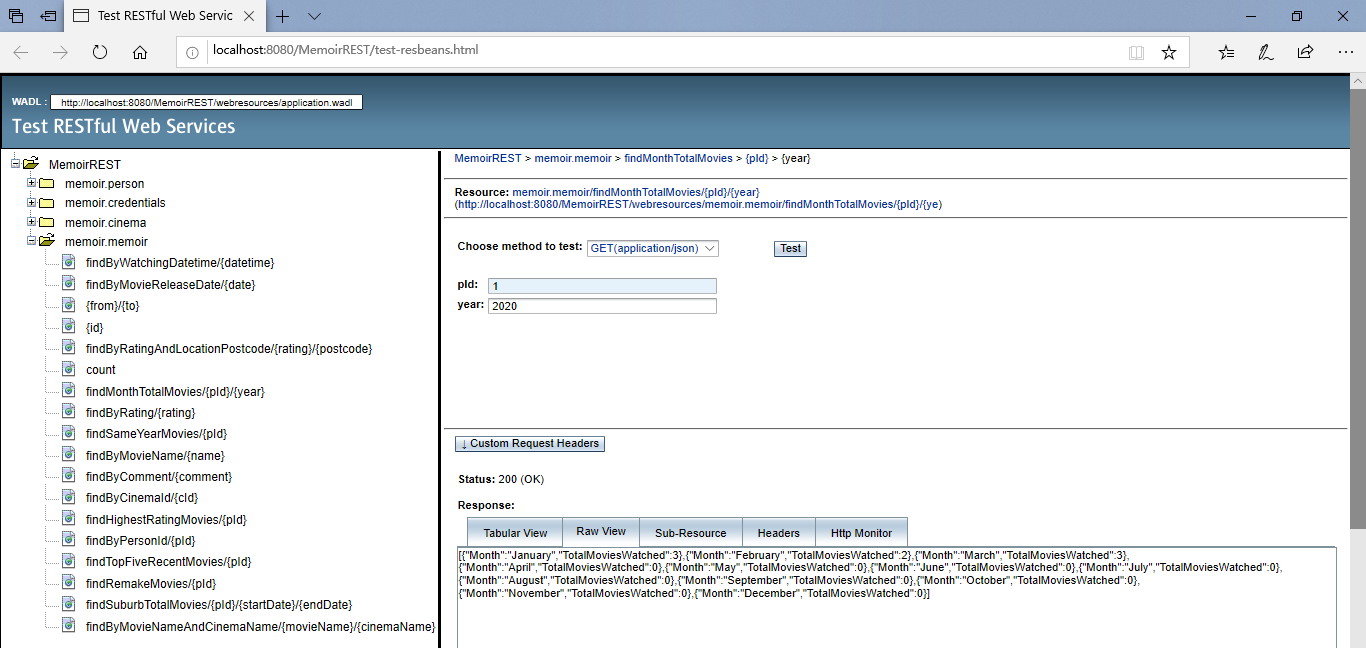
case 12: month = "December"; break;

default: break;

}

return month;

}



screenshot of task 4 b) query test result

## a REST method that will accept a user person id and return the name(s), the rating score(s) and release date(s) of the movie(s) with the highest rating score given by that user.

The following REST method accepts a person id and enables returning a list that contains the names, the rating scores and release dates implemented by using JsonObject to create JSON objects and JsonArrayBuilder to create the JsonArray object.

@GET

@Path("findHighestRatingMovies/{pId}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findHighestRatingMovies(@PathParam("pId") Integer pId) {

Query q = em.createQuery(

"SELECT m.mMovieName, OPERATOR('DateToString', m.mMovieReleaseDate), m.mRating "

+ "FROM Memoir m "

+ "WHERE m.mRating = (SELECT MAX(m.mRating) FROM Memoir m) "

+ "AND m.pId.pId = :pId ", Object[].class);

q.setParameter("pId", pId);

List<Object[]> queryList = q.getResultList();

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

for (Object[] row : queryList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("MovieName", (String) row[0])

.add("Rating", (BigDecimal) row[2])

.add("ReleaseDate", (String) row[1]).build();

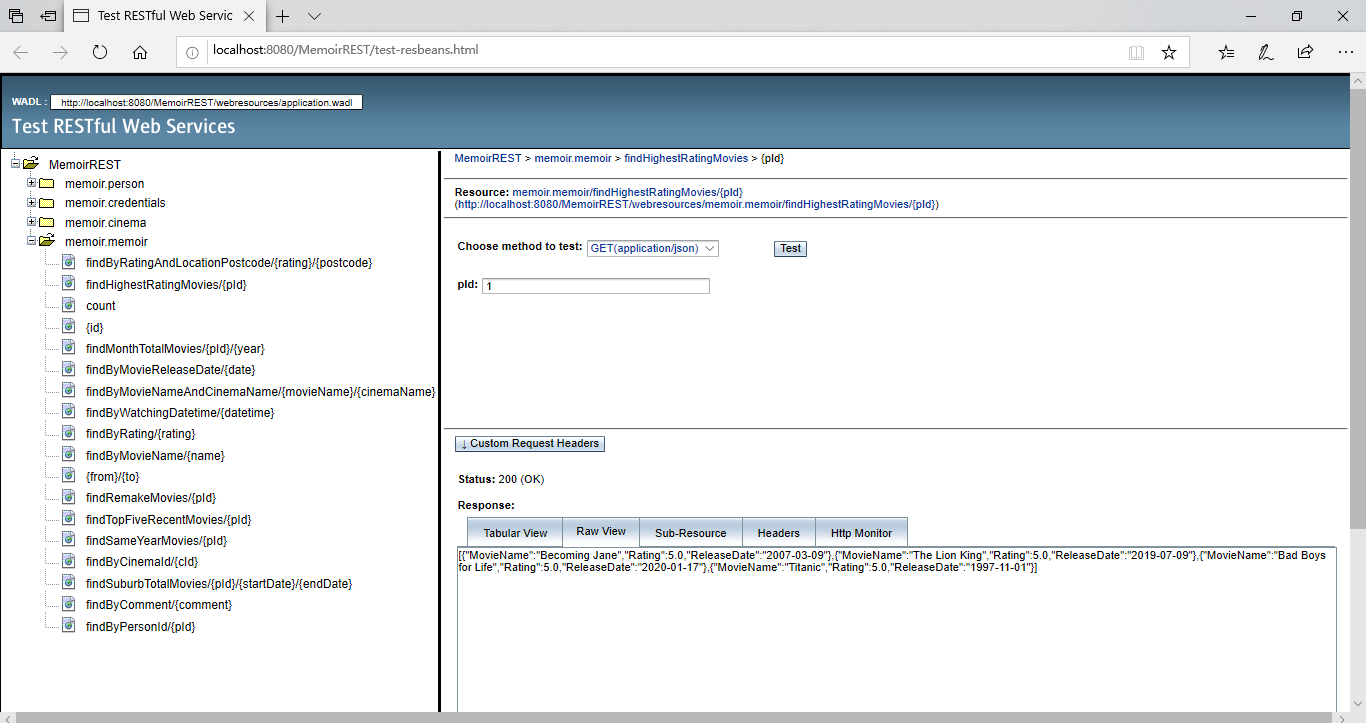
arrayBuilder.add(jsonObject);

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}



screenshot of task 4 c) query test result

## a REST method that will accept a person id and return a list of movie names and their release years for those movies that their release year is the same as the year the user watched them.

The following REST method accepts a person id and enables returning a list that contains the names and the release years for those movies that their release year is the same as the year the user watched them, implemented by using JsonObject to create JSON objects and JsonArrayBuilder to create the JsonArray object.

@GET

@Path("findSameYearMovies/{pId}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findSameYearMovies(@PathParam("pId") Integer pId) {

Query q = em.createQuery(

"SELECT m.mMovieName, EXTRACT(YEAR FROM m.mMovieReleaseDate) "

+ "FROM Memoir m "

+ "WHERE EXTRACT(YEAR FROM m.mMovieReleaseDate) = EXTRACT(YEAR FROM m.mWatchingDatetime) "

+ "AND m.pId.pId = :pId ", Object[].class);

q.setParameter("pId", pId);

List<Object[]> queryList = q.getResultList();

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

for (Object[] row : queryList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("MovieName", (String) row[0])

.add("ReleaseYear", (int) row[1]).build();

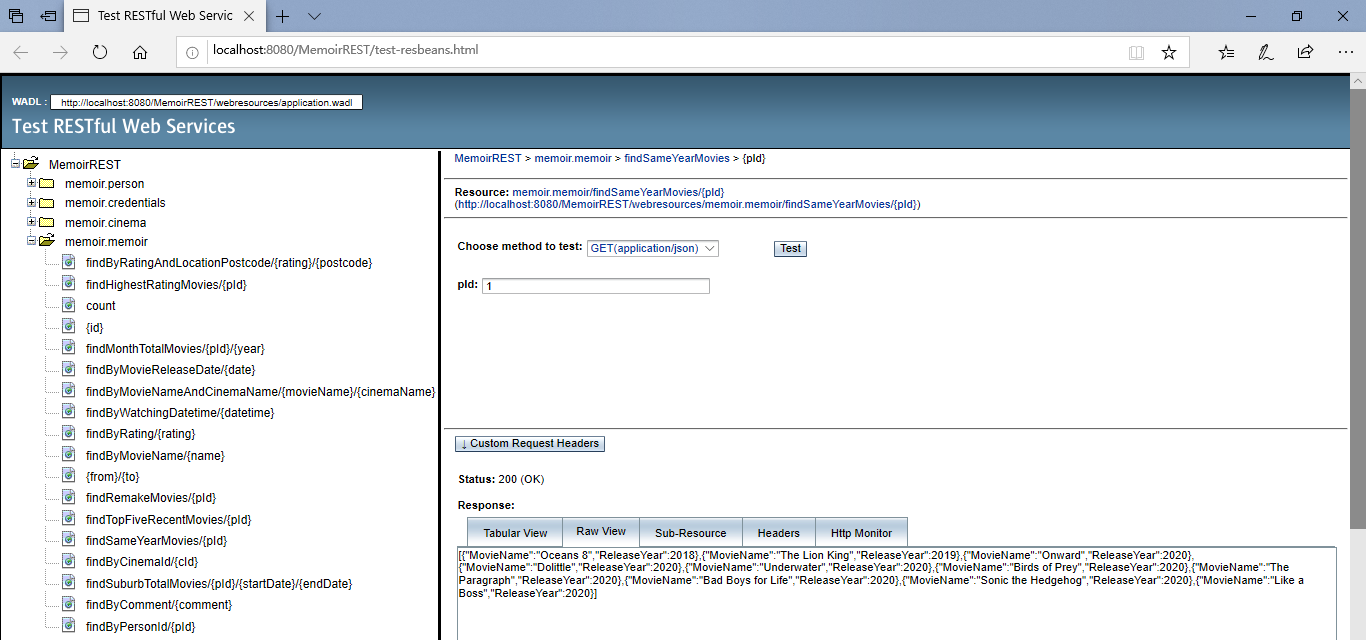
arrayBuilder.add(jsonObject);

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}



screenshot of task 4 d) query test result

## a REST method that will accept a person id and return a list of movie names and their release years for those movies that the user has watched their remakes as well.

The following REST method accepts a person id and enables returning a list that contains the names and their release years for those movies that the user has watched their remakes as well, implemented by using JsonObject to create JSON objects and JsonArrayBuilder to create the JsonArray object.

@GET

@Path("findRemakeMovies/{pId}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findRemakeMovies(@PathParam("pId") Integer pId) {

Query q = em.createQuery(

"SELECT m1.mMovieName, EXTRACT(YEAR FROM m1.mMovieReleaseDate), EXTRACT(YEAR FROM m2.mMovieReleaseDate) "

+ "FROM Memoir m1 join Memoir m2 "

+ "ON m1.mMovieName = m2.mMovieName AND m1.mMovieReleaseDate < m2.mMovieReleaseDate "

+ "WHERE m1.pId.pId = :pId", Object[].class);

q.setParameter("pId", pId);

List<Object[]> queryList = q.getResultList();

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

for (Object[] row : queryList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("movieName", (String) row[0])

.add("FirstRelease", (int) row[1])

.add("Remake", (int) row[2]).build();

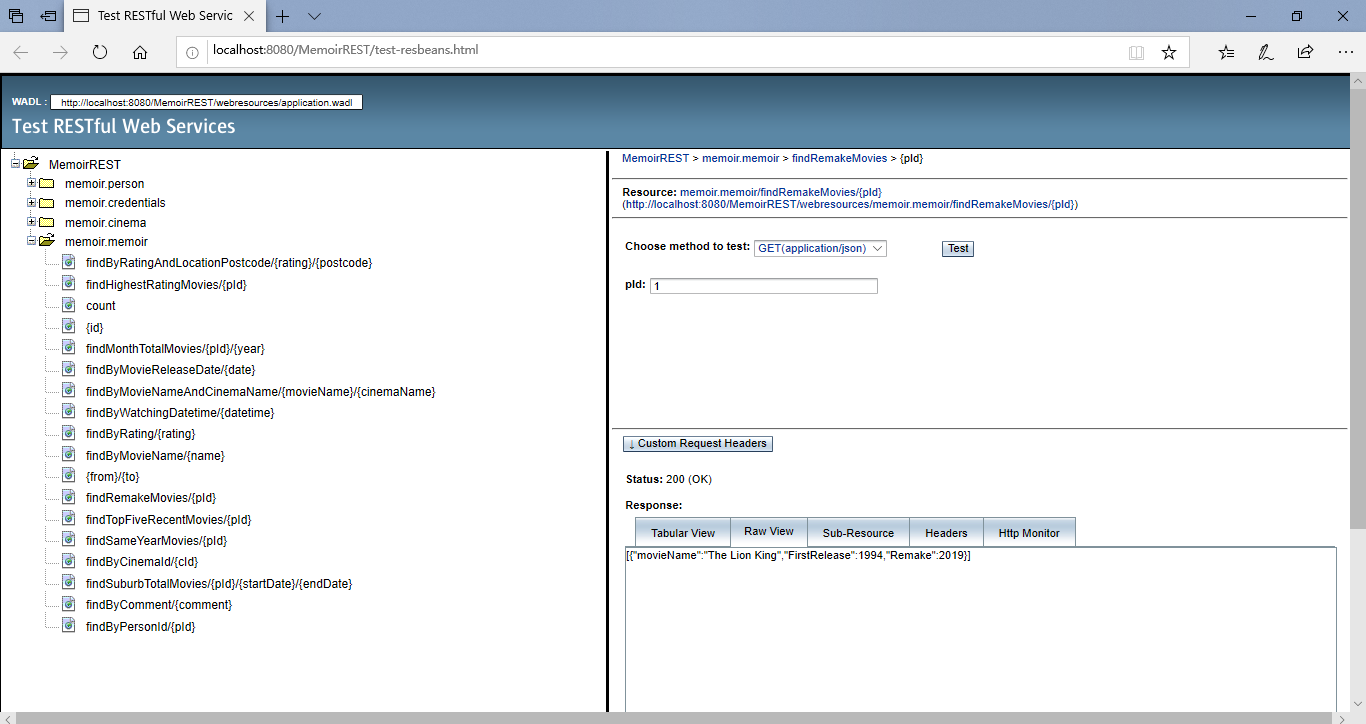
arrayBuilder.add(jsonObject);

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}



screenshot of task 4 e) query test result

## a REST method that will accept a user person id and return a list of the movie names, their release dates and rating scores for FIVE movies that have been released in the recent year and have the highest rating score (five top ones).

In this method, the first step is to get the original “queryList” which contains all the movies watched in the recent year in descending order. The second step is to use JsonObject to create JSON objects and JsonArrayBuilder to store the first five JSON objects to get the final JsonArray object.

@GET

@Path("findTopFiveRecentMovies/{pId}")

@Produces(MediaType.APPLICATION\_JSON)

public Object findTopFiveRecentMovies(@PathParam("pId") Integer pId) {

Query q = em.createQuery(

"SELECT m.mMovieName, OPERATOR('DateToString', m.mMovieReleaseDate), m.mRating "

+ "FROM Memoir m "

+ "WHERE EXTRACT(YEAR FROM m.mMovieReleaseDate) = EXTRACT(YEAR FROM CURRENT\_DATE) "

+ "AND m.pId.pId = :pId "

+ "ORDER BY m.mRating DESC ", Object[].class);

q.setParameter("pId", pId);

List<Object[]> queryList = q.getResultList();

JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();

int recordNumber = 0;

for (Object[] row : queryList) {

JsonObject jsonObject = Json.createObjectBuilder().

add("MovieName", (String) row[0])

.add("Rating",(BigDecimal) row[2])

.add("ReleaseDate", (String) row[1]).build();

arrayBuilder.add(jsonObject);

recordNumber ++;

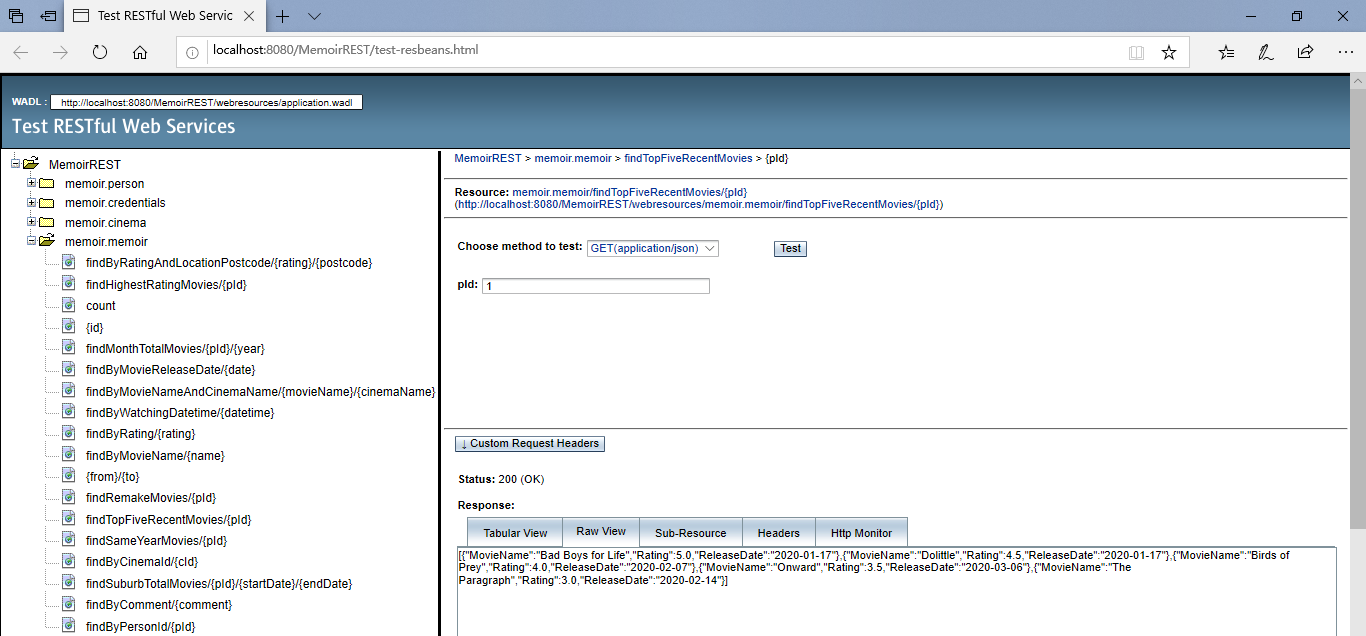
if (recordNumber >= 5) break;

}

JsonArray jsonArray = arrayBuilder.build();

return jsonArray;

}



screenshot of task 4 f) query test result

# References

1. EclipseLink JPA documentation: <https://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/JPQL#Functions>
2. Derby Reference Manual:

<https://builds.apache.org/job/Derby-docs/lastSuccessfulBuild/artifact/trunk/out/ref/index.html>

1. Java EE 7 APIs: <https://docs.oracle.com/javaee/7/api/>