MONASH University
Information Technology

ASSESSMENT COVER SHEET

Student ID number	30860490
Given Name	Yuze
Family name	Ling

Unit Name and Code:	**FIT 5046 Mobile and distributed computing systems**		
Campus:	**Caulfield**		
Assignment Title:	**Assignment 1**		
Name of Lecturer:	**Pari Delir Haghighi**		
Name of Tutor:	**Josh Olsen**		
Tutorial Day and Time:	**Thursday 10:00am~12:00pm**		
Phone Number:	**0466060438**		
Email Address:	**ylin0081@student.monash.edu**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes ☒ No			
Due Date:	**Apr 24, 2020**	Date Submitted:	**Apr 19, 2020**

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date)_____ Signature of lecturer/tutor _____

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature .....Yuze Ling..... Date…… Apr 19, 2020……

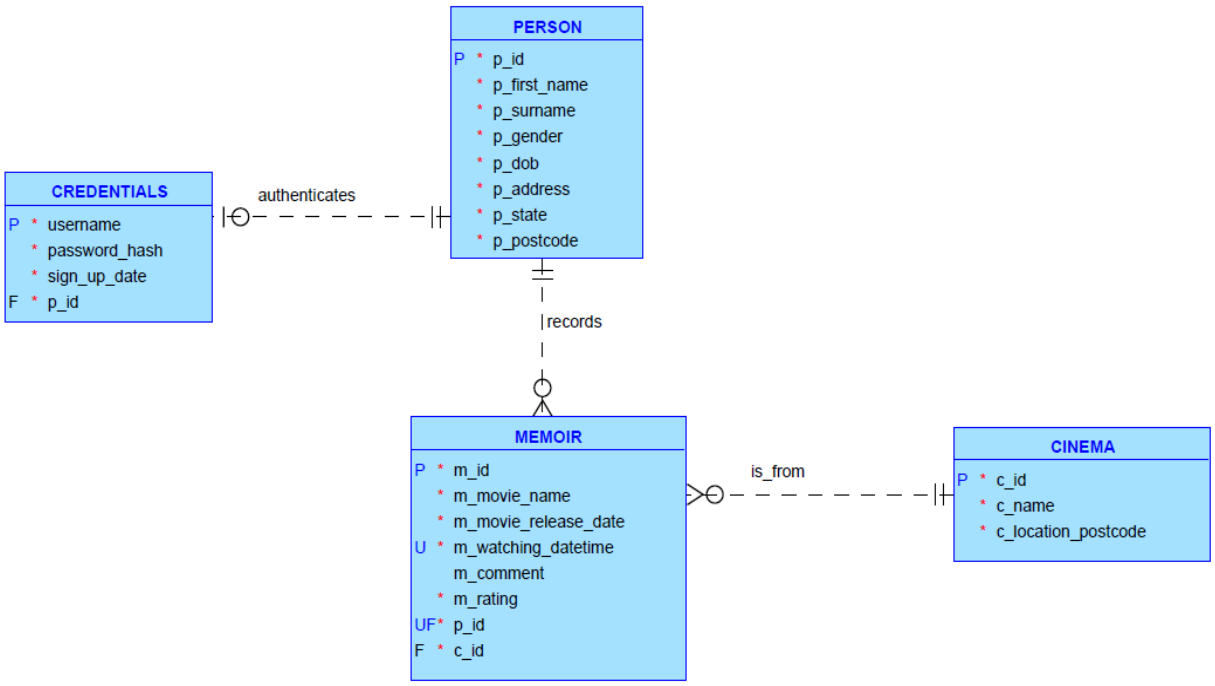
* delete (iii) if not applicable

The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. *Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au*

CONTENTS

Task 1 - Database

a) ER Diagram for Memoir database:



Memoir database ER Diagram

b) SQL code for creating and populating tables:

1) 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;
```

2) 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);

c) Screenshots showing populated data for each table

C_ID	C_NAME	C_LOCATION_POSTCODE
1	HOYTS Broadway	2007
2	HOYTS Chadstone	3148
3	Village Cinemas Rivoli	3123
4	GU Film House	5000
5	Luna Cinemas	6007

the screenshot showing populated data for cinema table

P_ID	P_FIRST_NAME	P_SURNAME	P_GENDER	P_DOB	P_ADDRESS	P_STATE	P_POSTCODE
1	Roger	Mason	M	1982-02-25	17 Shepparson Avenue	VIC	3163
2	Serena	Petrova	F	1975-12-06	160 Grote Street	SA	5000
3	Mark	Williams	M	1993-07-30	2 Chippendale Way	NSW	2008

the screenshot showing populated data for person table

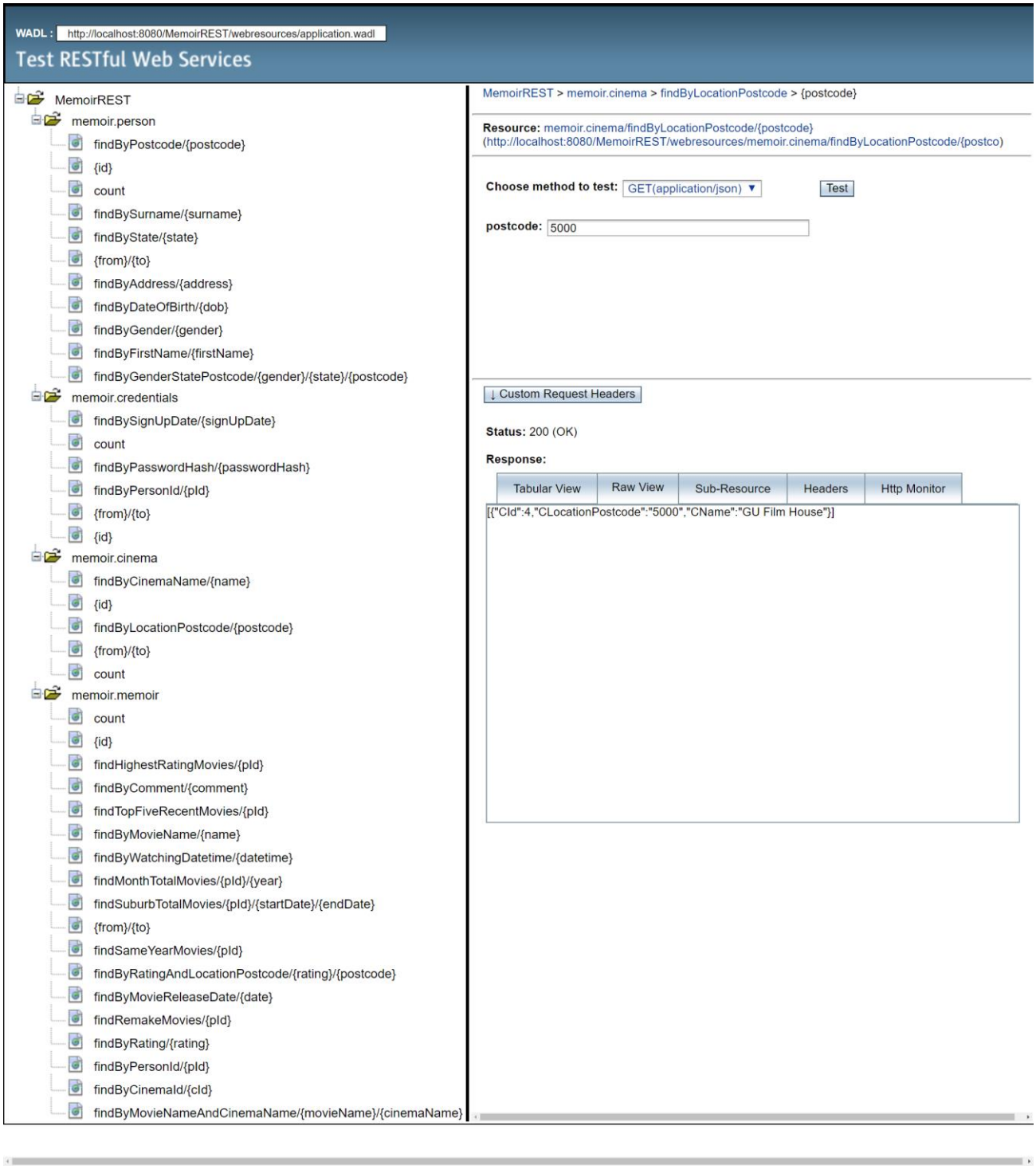
USERNAME	PASSWORD_HASH	SIGN_UP_DATE	P_ID
rogermason82	cbf4d9fb4123b06b28f583ff81567403	2019-11-13	1
serenapetrova75	89962fdbdec468b59a8c00842c105586	2019-12-26	2
markwilliams93	8d7f20caa2345fcd3e03e0bb6918ed3b	2020-01-08	3

the screenshot showing populated data for credentials table

M_ID	M_MOVIE_NAME	M_MOVIE_RELEASE_DATE	M_WATCHING_DATETIME	M_COMMENT	M_RATING	P_ID	C_ID
1	Titanic	1997-11-01	2008-09-12 16:30:00.0	Fantastic Film! Amazing!	4.5	1	2
2	Titanic	1997-11-01	2013-10-01 12:30:00.0	Reviewed the classic film after five years, still sensational!	5.0	1	4
3	Oceans 8	2018-06-08	2018-06-17 16:00:00.0	Pretty Good!	4.5	1	3
4	The Intern	2015-09-25	2019-07-25 09:30:00.0	Interesting Story!	4.0	1	4
5	Love & Other Drugs	2010-11-04	2018-08-01 11:00:00.0	Too sleepy to catch up. Hope to have another try!	3.5	1	5
6	Love & Other Drugs	2010-11-04	2019-03-09 16:00:00.0	What a moving story!!!	4.5	1	3
7	Becoming Jane	2007-03-09	2019-01-24 15:30:00.0	Perfect performance!	5.0	1	2
8	One Day	2011-08-19	2018-02-28 19:00:00.0	So boring!!	1.5	1	1
9	Me Before You	2016-05-23	2019-05-22 20:00:00.0	Just so so...	3.0	1	3
10	Valentines Day	2010-01-28	2015-11-19 20:30:00.0	Far from realistic.	2.5	1	3
11	The Lion King	2019-07-09	2019-10-17 16:00:00.0	Perfect! Even better than the 1994 version!	5.0	1	3
12	The Lion King	1994-06-24	1995-01-28 19:00:00.0	Lots of fun!	4.5	1	5
13	Onward	2020-03-06	2020-03-17 18:00:00.0	Ok but some settings are not consistent!	3.5	1	3
14	Dolittle	2020-01-17	2020-01-24 13:30:00.0	Great movie!	4.5	1	2
15	Underwater	2020-01-10	2020-02-01 10:30:00.0	Not worthy at all!	0.5	1	2
16	Birds of Prey	2020-02-07	2020-03-03 17:30:00.0	A good story!	4.0	1	3
17	The Paragraph	2020-02-14	2020-02-29 12:00:00.0	Not too bad...	3.0	1	2
18	Bad Boys for Life	2020-01-17	2020-01-18 19:30:00.0	Quite funny! Very Good!	5.0	1	1
19	Sonic the Hedgehog	2020-02-14	2020-03-08 15:00:00.0	Honestly not live up to expectation!	2.0	1	3
20	Like a Boss	2020-01-10	2020-01-31 21:00:00.0	So so...	2.5	1	2

the screenshot showing populated data for memoir table

Task 2 - RESTful Web Service



the browser screenshot showing all methods

Task 3 - Dynamic and Static Queries

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

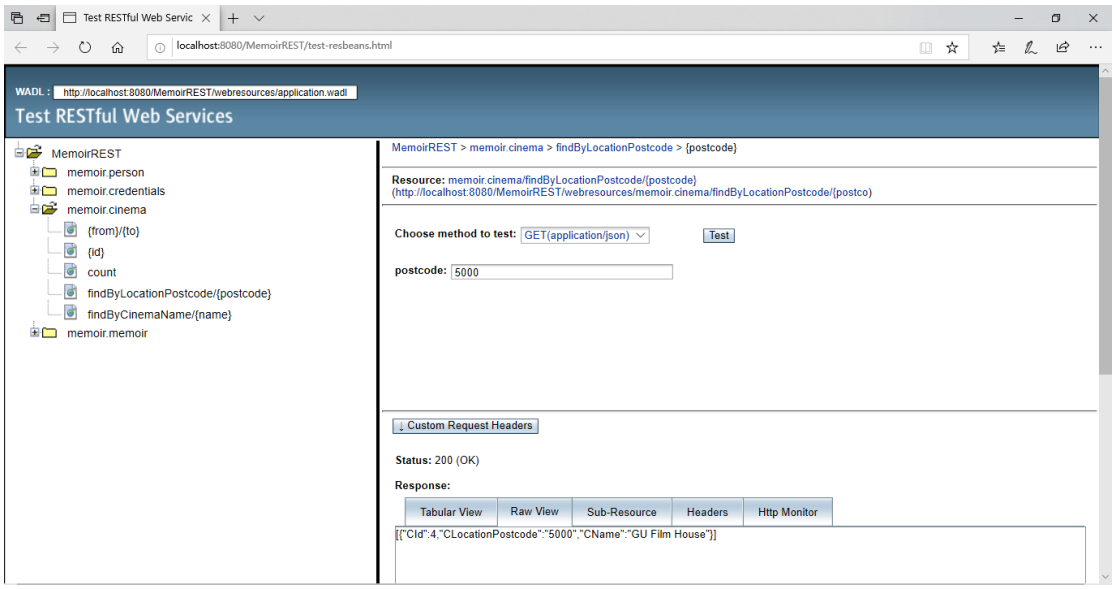
1) Cinema Table

(i) 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();
}
```

(ii) 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

2) Credentials Table

(i) Named query by password hash

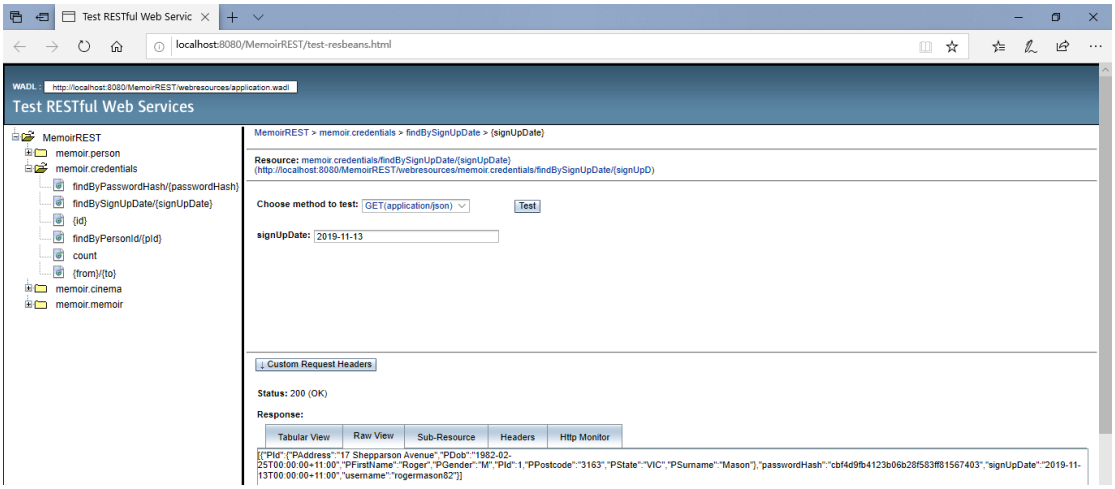
```
@GET
@Path("findByPasswordHash/{passwordHash}")
@Produces(MediaType.APPLICATION_JSON)
public List<Credentials> findByPasswordHash(@PathParam("passwordHash") String pw) {
    Query q = em.createNamedQuery("Credentials.findByPasswordHash");
    q.setParameter("passwordHash", pw);
    return q.getResultList();
}
```

(ii) 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();
}
```

(iii) 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

3) Person Table

(i) 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();
}
```

(ii) 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();
}
```

(iii) 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();
}
```

(iv) 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();
}
```

(v) Named query by address

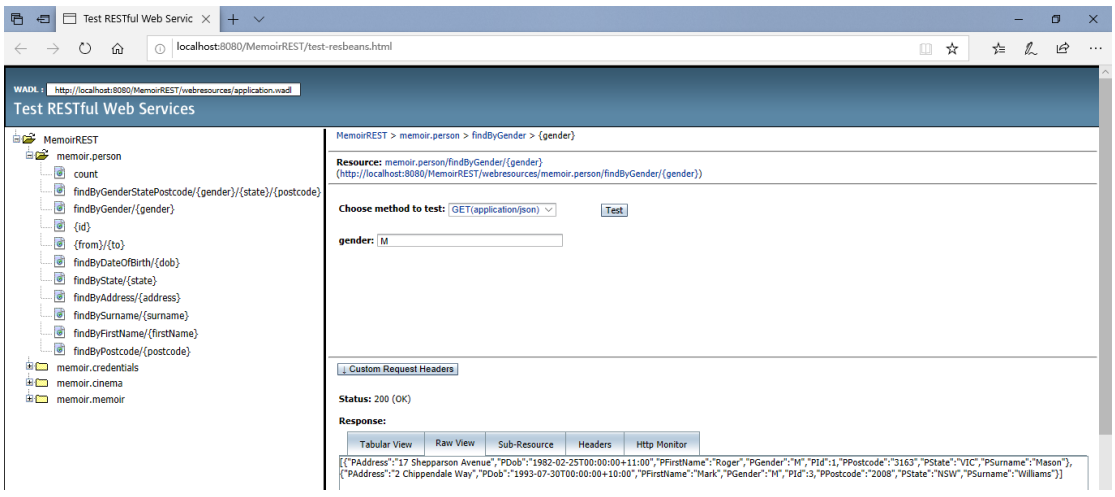
```
@GET
@Path("findByAddress/{address}")
@Produces(MediaType.APPLICATION_JSON)
public List<Person> findAddress(@PathParam("address") String address) {
    Query q = em.createNamedQuery("Person.findByPAddress");
    q.setParameter("pAddress", address);
    return q.getResultList();
}
```

(vi) 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();
}
```

(vii) 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

4) Memoir Table

(i) 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();
}
```

(ii) 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();
}
```

(iii) 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();
}
```

(iv) 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();
}
```

(v) Named query by rating

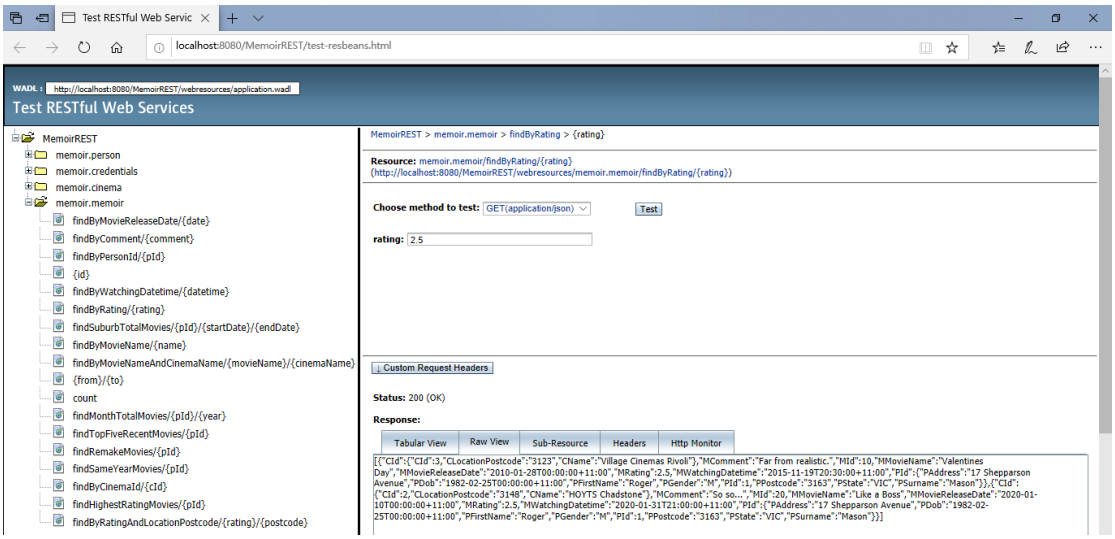
```
@GET
@Path("findByRating/{rating}")
@Produces(MediaType.APPLICATION_JSON)
public List<Memoir> findByRating(@PathParam("rating") BigDecimal rating) {
    Query q = em.createNamedQuery("Memoir.findByMRating");
    q.setParameter("mRating", rating);
    return q.getResultList();
}
```

(vi) 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();
}
```

(vii) 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();
}
```

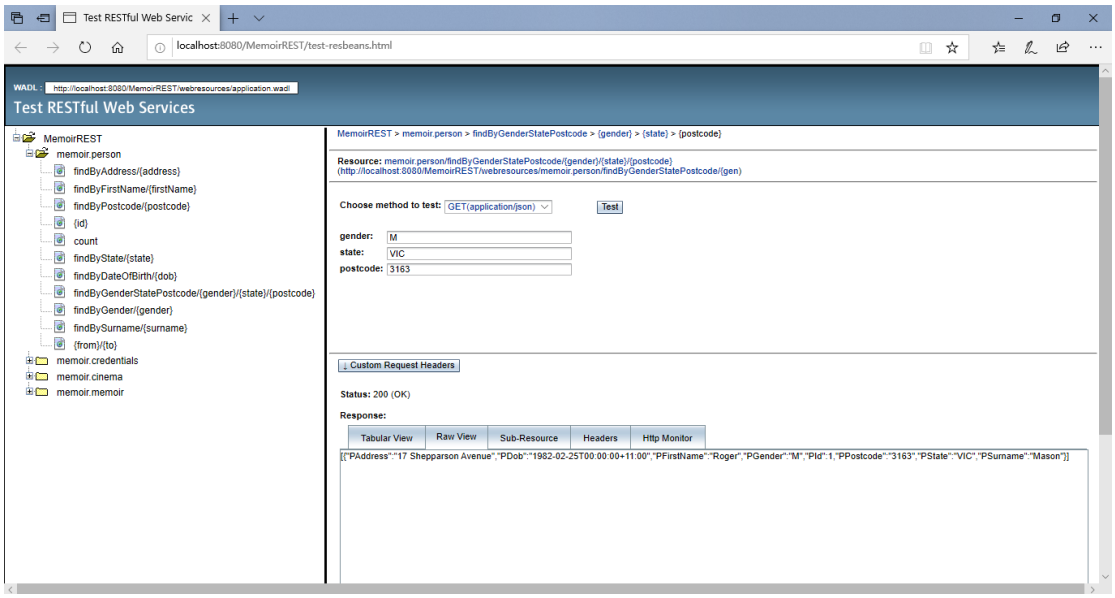


screenshot of named query by rating

b) 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();
}
```

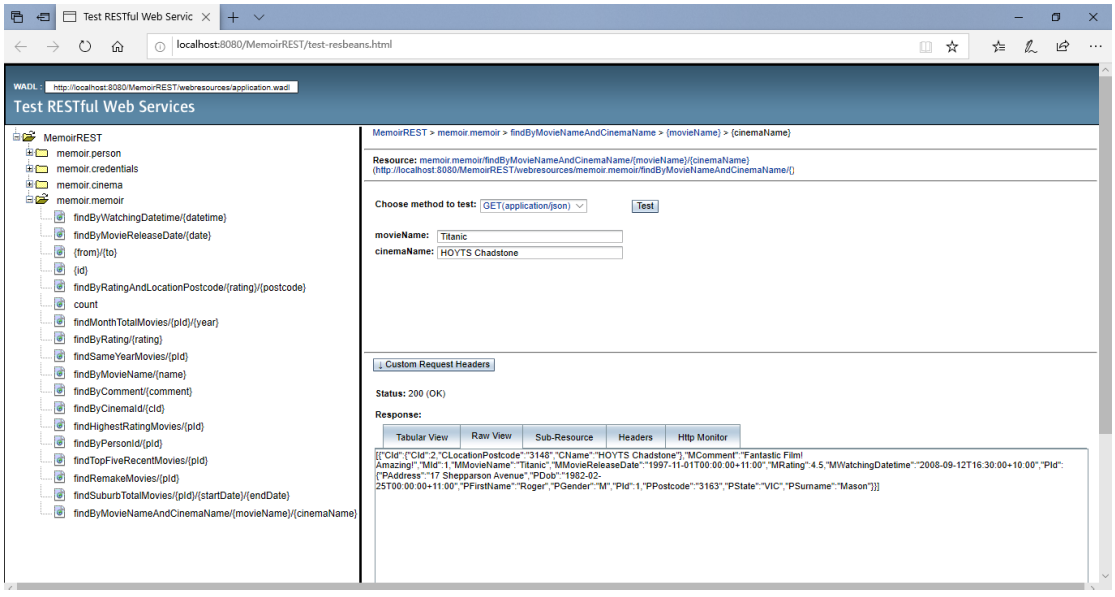


screenshot of dynamic query by gender, state and postcode

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**.

The following REST method enables querying the memoir and the cinema tables using a combination of movie name and cinema name, 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

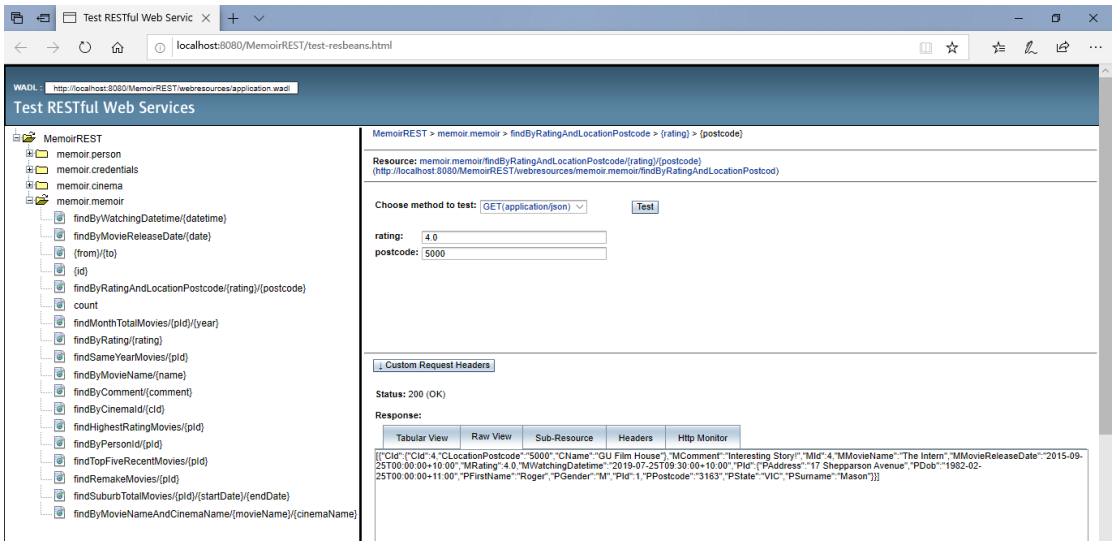
d) a REST method that enables **querying the memoir and the cinema tables using a combination of two attributes in the condition where each attribute is from a different table**. The query should be a **STATIC query using an IMPLICIT JOIN**.

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

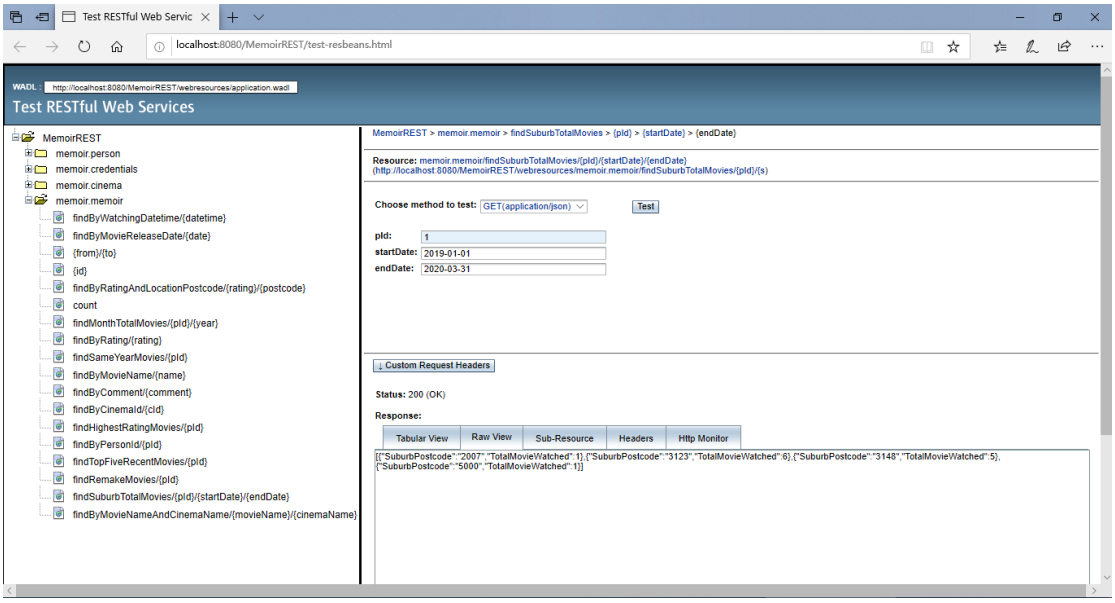
Task 4 - Advanced REST methods

a) 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

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.

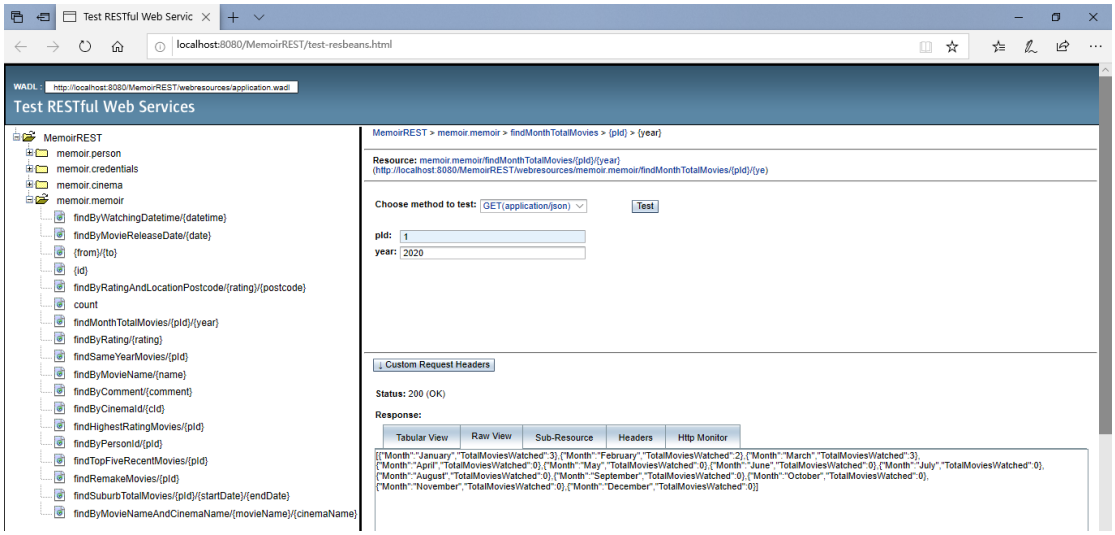
In this method, the first step is to get the original “queryList” which only contains the months with movie watching records. The second step is to build an “outcomeList” which contains all the months from 1 to 12 where those months without any watching record will return 0. The final step is to use 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

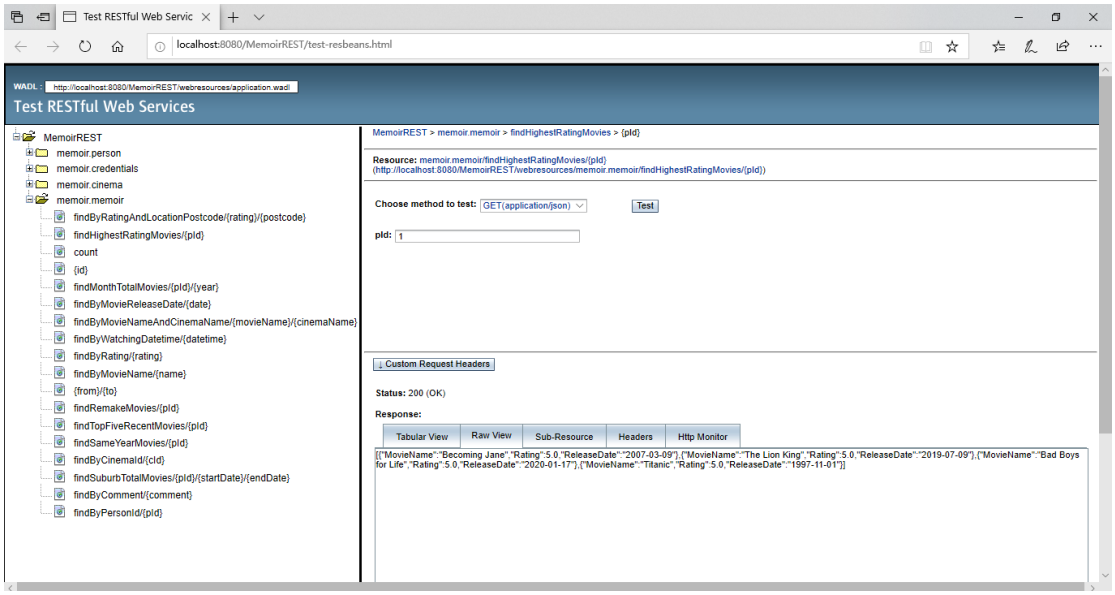
c) a REST method that will accept a user person id and return the name(s), the rating score(s) and release date(s) of the movie(s) with the highest rating score given by that user.

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;
    }
```



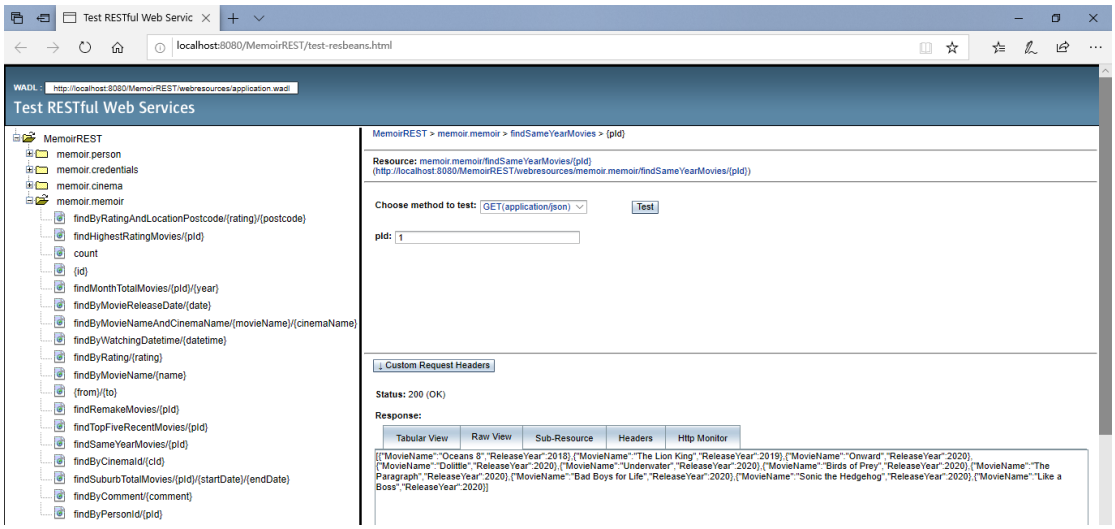
screenshot of task 4 c) query test result

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.

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

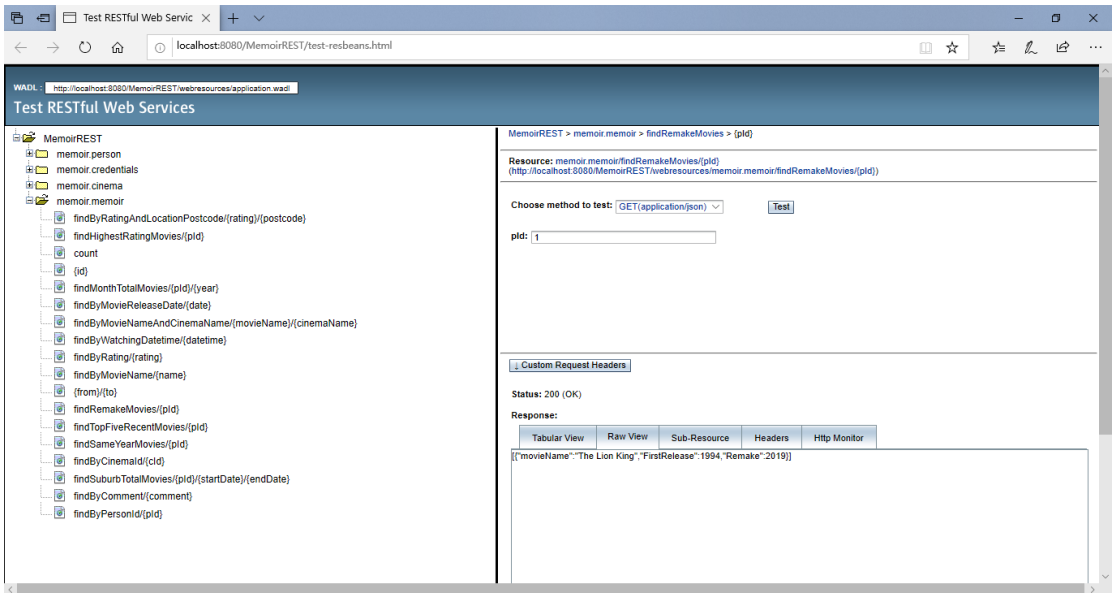
e) a REST method that will accept a person id and return a list of movie names and their release years for those movies that the user has watched their remakes as well.

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;
    }
```



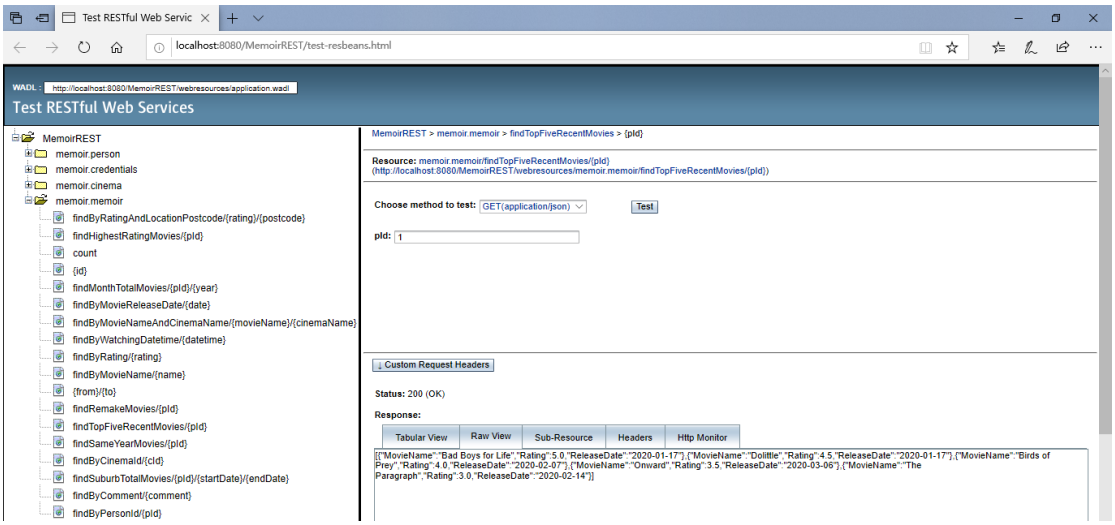
screenshot of task 4 e) query test result

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).

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
3. Java EE 7 APIs: https://docs.oracle.com/javaee/7/api/