Software Design - Course Project Phase 1

EECS 3311A
Summer 2024

Professor Ilir Dema
July 22, 2024

Team AE: Ankit Modhera, Yienisha Abeyratne,

Sajeevan Panchan, Vachaspathy Sreedhar

Introduction

Introduce yourself (and your partner if you have one), and describe shortly what you intend to learn in this project, by identifying a personal learning goal (i.e. you may decide to use a version tracking software to track your work, such as git, etc.)

Yienisha Abeyratne

- I'm Yienisha, and I am part of the AE group. Fun fact, I haven't had poutine my entire life
- My personal learning goal is to learn how to create APIs and use the design patterns learnt in class on this project. Hopefully, I could add the project to my resume.

Ankit Modhera

- I'm Ankit, I am part of the group AE. Fun fact, I like turtles
- My personal goal is to learn how to create and use APIs, GUIs and other practical skills covered in EECS 3311.

Sajeevan Panchan

- My name is Sajeevan and I am in group AE
- My personal learning goal is to get a better understanding of database management by exploring how to design, implement and integrate API's.

Vachaspathy Sreedhar

- Hey there, my name's Vachaspathy. But I simply prefer to be called Vachas (trust me, it's way easier to pronounce and remember).
- As a double major in stats and computer science, my personal learning goal through this project is to use statistical methods to understand better the performance of the API, such as response time analysis, and explore ways to optimize it based on statistical insights

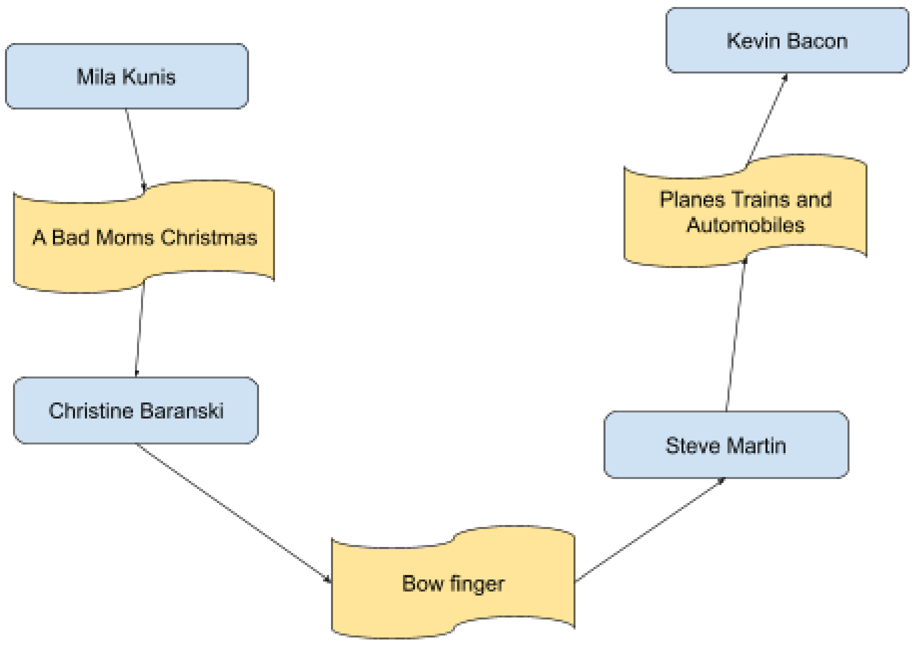
Short Description

Give a short description (at most two paragraphs) in your own words, of the application that you will develop. You may rephrase material from the handout, but you need to explain it in your own words

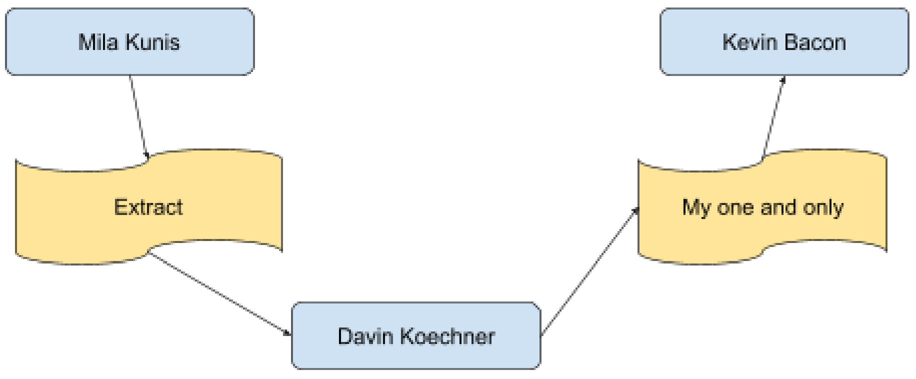
The Six Degrees of Kevin Bacon is a game where a player selects an actor from a list of actors and the degree of separation between them is displayed.

A sample that we looked into to get an idea is: https://oracleofbacon.org/

The link between actors can be based on several genres or a single genre. For example, with only "comedy" movies Mila Kunis has a Bacon number of 3



With Comedy, drama, horror, mystery and romance, the Bacon number is 2



The application we are developing is a backend service that computes the "six degrees of Kevin Bacon," identifying the shortest path between Kevin Bacon and any given actor through their shared movies using the Neo4j graph database. This project involves creating REST API endpoints to add and query actors, movies, and their relationships practising NoSQL database management and server/client software architecture. Key features include sorting movies by popularity, retrieving movies released in a specific year, and providing box office performance data. For example, the endpoint POST /api/v1/moviesByPopularity returns a list of movies sorted by their IMDB rating, while POST /api/v1/moviesByYear lists movies released in a specified year, and GET /api/v1/boxOfficePerformance provides the box office earnings of a given movie. Testing involves verifying successful responses and handling errors, ensuring robust and reliable endpoints for all functionalities.

New features

Identify two to three new features (if you work alone, two features suffice, if you work with a partner do three features). A sample feature might be i. Add a new property to movies (i.e. genre) and add an endpoint that generates a list of movies for a given genre.

1. Date (Year) - list of movies released in a certain year.
2. Popularity - list of movies sorted by popularity. (eg: IMDB rating)
3. Box office performance - how much the movie made in theatres and the categorical terms that are used to describe the performance of a film at the box office (blockbuster, hit, flop, etc)

Endpoints

Specify the type of the endpoint (POST/GET/etc) and create a description of it following the examples in this handout.

Popularity

- GET /api/v1/moviesByPopularity
    - Description: This endpoint returns a list of movies sorted by popularity(IMDB)
    - Response:
        - ‘Movies’: List of Movie objects (movieId, movieName, rating)
    - Response Example:

        -
          ```
          {
            "Movies":[
              {
                "movieId":“nm1119903”,
                "name":"The GodFather",
                "rating":8.3
              },
              {
                "movieId":“nm1119903”,
                "name":"Avatar",
                "rating":9.5
              }
            ]
          }
          ```
    - Expected Response:
        - **200 OK** - For a successful query
        - **500 INTERNAL SERVER ERROR** - If the query was unsuccessful

- PUT/api/v1/addRating
  - Description: This endpoint is to add IMDB rating of a movie to a movie already in the database
  - Body Parameters:
    - movieId: String
    - rating: double
  - Body Example

    -
      ```
      {
              "moveId":"nm1119903",
              "rating":9.5
      }
      ```
  - Expected Response:
    - **200 OK** - For a successful add
    - **400 BAD REQUEST** - If the request body is improperly formatted or missing required information
    - **404 NOT FOUND** - If the movie does not exist in the database
    - **500 INTERNAL SERVER ERROR** - If save or add was unsuccessful

Date

- GET /api/v1/moviesByYear
  - Description: This endpoint returns a list of movies released in a specified year
  - Body Parameters:
    - ‘year’: integer
  - Body Example

    -
      ```
      {
              "moveId":"nm1119903",
              "year":2024
      }
      ```
  - Response:
    - ‘Movies’: List of Movie Objects(movieId, name, rating)
  - Response Example

    -
      ```
      [
          {
             "movieId":“nm1119903”,
             "name":"The GodFather",
             "rating":8.3
            “year: 1972
          },
          {
             "movieId":“tt049954”,
             "name":"Avatar",
             "Rating":9.5
            “year”  : 2009
      ```

```
        }
    ]
```

- Expected Response
    - 200 OK - For a successful Query
    - 400 BAD REQUEST - If the Request parameter is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If the query was unsuccessful

- GET /api/v1/addRating
    - Description: This endpoint is to add a rating node into the dataBase
    - Body Parameters:
        - movieID: String
        - movieName: String
        - rating: int
    - Body Example

        -
          ```
          {
                “moveId”: “nm1119903”
                “name”: “TheGodFather”
                “rating”: 9.5
          }
          ```
    - Expected Response:
        - 200 Ok - For a successful add
        - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
        - 500 INTERNAL SERVER ERROR - If save or add was unsuccessful

Box Office Performance

- GET/api/v1/boxOfficePerformance
    - Description: This endpoint returns the box office performance of movies
    - Body Parameters:
        - ‘name’: String
        - ‘movieId’: String
    - Body Example

        -
          ```
          {
                “movieId”: “nm1119903”
                “name”: “TheGodFather”
          }
          ```
    - Response
        - ‘profit’: int
    - Response example

        -
          ```
          [
                {“name”: “TheGodFather”, “profit”: 100000000},
                {“name”:  “Avatar”, “profit”: 250000000}
          ]
          ```
    - Expected Response:
        - 200 OK - For a successful query

        - **400 BAD REQUEST** - If the Request parameter is improperly formatted or missing required information
        - **500 INTERNAL SERVER ERROR**- For a Unsuccessful query
- PUT/api/v1/addProfit
    - Description: This endpoint is to add a profit node into the dataBase
    - Body Parameters:
        - movieId: String
        - profit: int
    - Body Example

```
{
    "moveId":“nm1119903”,
    "profit":10000000
}
```

    - Expected Response:
        - **200 OK** - For a successful add
        - **400 BAD REQUEST** - If the request body is improperly formatted or missing required information
        - **500 INTERNAL SERVER ERROR** - If save or add was unsuccessful

Testing

Describe how you will test each feature.

Note that the examples below are based on the endpoint scenarios from above. For the purpose of a simple and straightforward testing description, we have shown the testing code via text. However, for the actual implementation in phase II, to test these endpoints effectively, we will use tools like Postman.

Popularity Testing:

- **Testing Plan (based on the three possibilities for the expected response):**
    - **Possibility #1:** Successful Query
        - **Test Name:** moviesByPopularityPass
        - **Status:** 200 OK
        - **Description**: Checks if the endpoint correctly returns the list of movies sorted by their ratings on IMBD when the query is successful.
        - **Request**: POST /api/v1/moviesByPopularity
        - **Expected Response**:

```
{
        "Movies": [
                {"movieID": "nm1119903", "movieName": "The
        GodFather", "rating": 8.3},
                {"movieID": "tt0499549", "movieName": "Avatar",
        "rating": 9.5}
            ]
}
```

- **Possibility #2**: Unsuccessful Query
    - **Test Name**: moviesByPopularityFail
    - **Status**: 500 INTERNAL SERVER ERROR
    - **Description**: Checks if the endpoint returns 500 INTERNAL SERVER ERROR when the query is unsuccessful.
    - **Request**: POST /api/v1/moviesByPopularity
    - Expected Response:
      ```
      {
              "error": "Internal Server Error"
      }
      ```

- **Possibility #3:** Parameter improperly formatted or missing required information
    - **Test Name**: moviesByYearFail
    - **Status**: 400 BAD REQUEST
    - **Description**: Checks if the endpoint returns 400 status code when the request parameter is missing or improperly formatted.
    - **Request**: POST /api/v1/moviesByYear
    - **Body**: {"Yr": "nineteen ninety-four"}
    - **Expected Response:**
      ```
      {
              "error": "Bad Request - Missing or Improperly Formatted Parameter"
       }
      ```

Date Testing:

- **Testing Plan (based on the three possibilities for the expected response):**
    - **Possibility #1:** Successful Query
        - **Test Name**: moviesByYearPass
        - **Status**: 200 OK
        - **Description**: Checks if the endpoint correctly returns a list of movies released in the specified year when the query is successful.
        - **Request**: POST /api/v1/moviesByYear
        - **Body**: {"Year": 1994}
        - **Expected Response**
          ```
          {
                  "Movies": [
                                  {"movieID": "nm1119903", "movieName": "The
                          GodFather", "rating": 8.3},
                                  {"movieID": "tt0499549", "movieName": "Avatar",
                          "rating": 9.5}
                             ]
          }
          ```

    - **Possibility #2:** Parameter improperly formatted or missing required information
        - **Test Name**: moviesByYearFail
        - **Status**: 400 BAD REQUEST
        - **Description**: Checks if the endpoint returns 400 status code when the request parameter is missing or improperly formatted.

        - **Request**: POST /api/v1/moviesByYear
        - **Body**: {"Yr": "nineteen ninety-four"}
        - **Expected Response:**
          {
          "error": "Bad Request - Missing or Improperly Formatted Parameter"
          }
    - **Possibility #3:** Unsuccessful Query
        - **Test Name**: moviesByYearServerFail
        - **Status**: 500 INTERNAL SERVER ERROR
        - **Description**: Checks if the endpoint returns a 500 status code when the query is unsuccessful.
        - **Request**: POST /api/v1/moviesByYear
        - **Body**: {"Year": 1994}
        - **Expected Response:**
          {
          "error": "Internal Server Error" }

Box Office Performance:

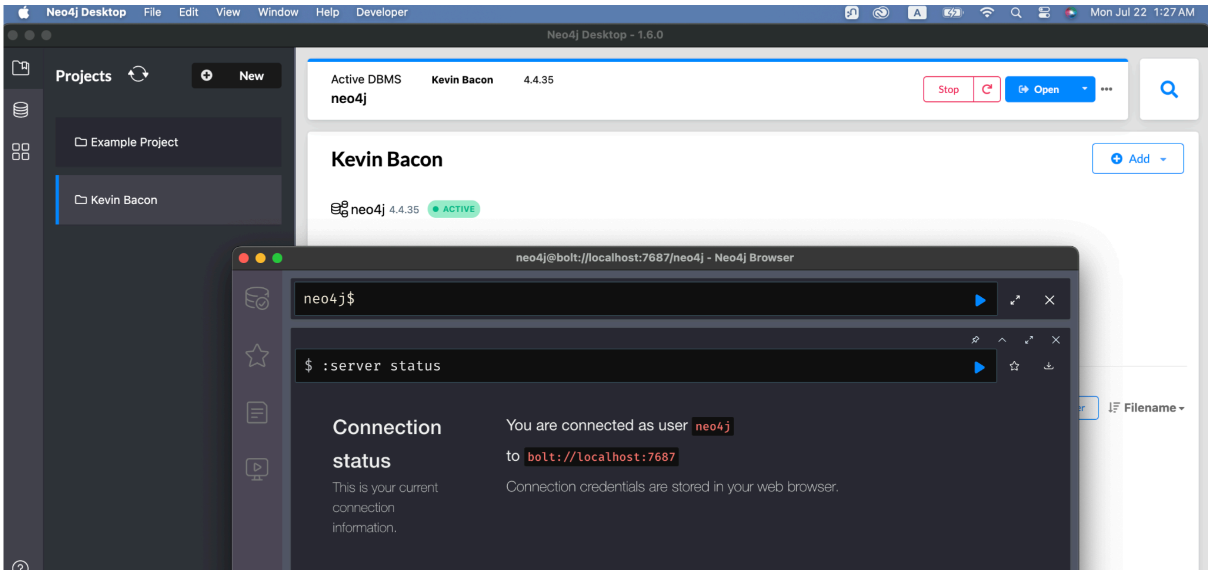
- **Testing Plan (based on the three possibilities for the expected response):**
    - **Possibility #1:** Successful Query
        - **Test Name**: boxOfficePerformancePass
        - **Status**: 200 OK
        - **Description**: Checks if the endpoint correctly returns the box office performance of the specified movie when the query is successful.
        - **Request**: GET /api/v1/boxOfficePerformance
        - **Parameters**: Name = Avatar movieID = tt0499549
        - **Expected Response:**
          {
          "Profit": 1500000000
          }
    - **Possibility #2:** Parameter improperly formatted or missing required information
        - **Test Name**: boxOfficePerformanceFail
        - **Status**: 400 BAD REQUEST
        - **Description**: Verify that the endpoint returns a 400 status code when the request parameter is missing or improperly formatted.
        - **Request**: GET /api/v1/boxOfficePerformance
        - **Parameters**: movieNm = Avatar
        - **Expected Response**
          {
          "error": "Bad Request - Missing or Improperly Formatted Parameter"
          }
    - **Possibility #3:** Unsuccessful Query
        - **Test Name**: boxOfficePerformanceServerFail
        - **Status**: 500 INTERNAL SERVER ERROR

- **Description**: Checks if the endpoint returns a 500 status code when the query is unsuccessful.
- **Request**: GET /api/v1/boxOfficePerformance
- **Parameters**: Name = Avatar movieID = tt0499549
- **Expected Response**:
```
{
        "error": "Internal Server Error"
}
```

Project setup screenshots

Complete your work by reporting about your setup, including screenshots of your project setup in your chosen IDE

- Neo4j database was created with the 4.4.35 version



```
neo4j$
```

```
$ :server status
```

Connection status

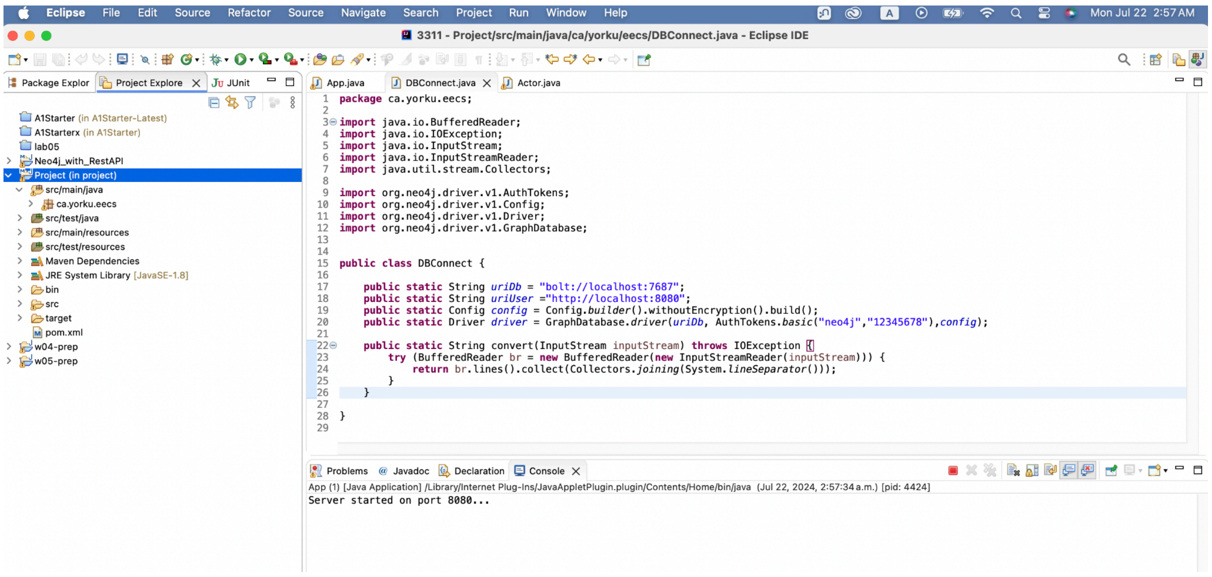
This is your current connection information.

You are connected as user `neo4j`

to `bolt://localhost:7687`

Connection credentials are stored in your web browser.

- Database connection created

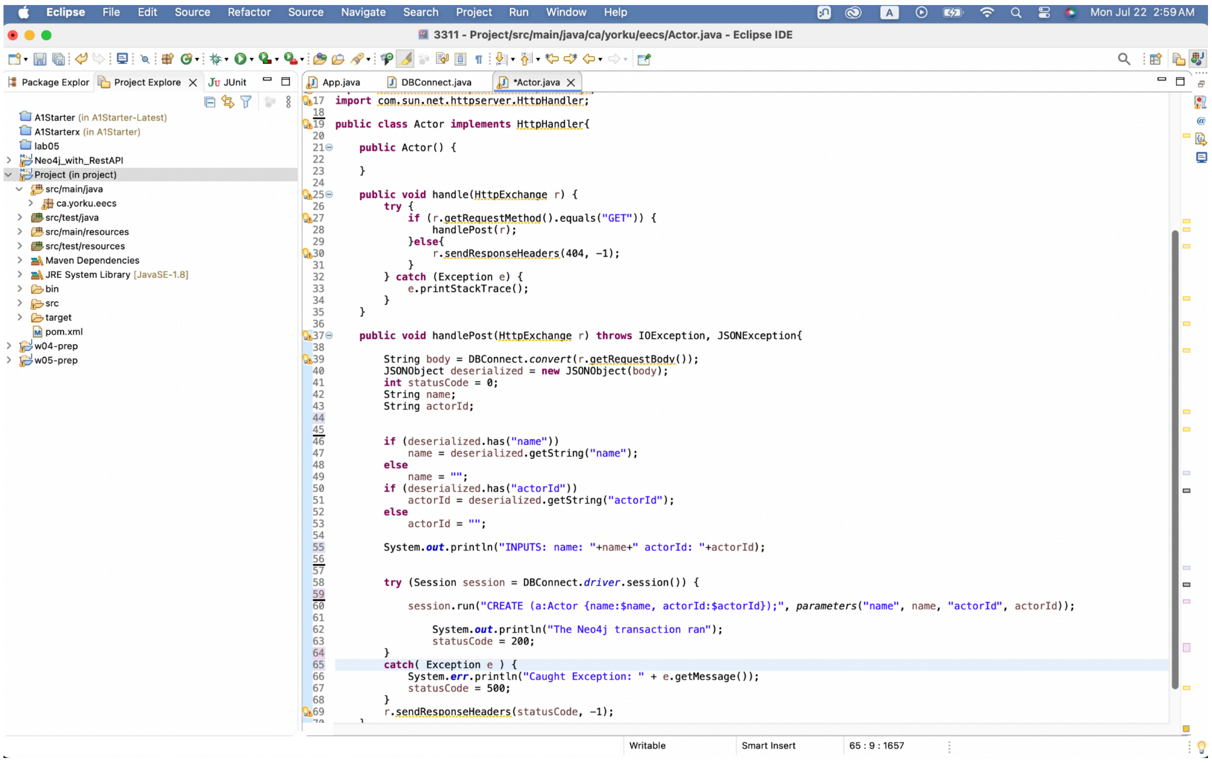


3311 - Project/src/main/java/ca/yorku/eecs/DBConnect.java - Eclipse IDE

```java
package ca.yorku.eecs;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.stream.Collectors;

import org.neo4j.driver.v1.AuthTokens;
import org.neo4j.driver.v1.Config;
import org.neo4j.driver.v1.Driver;
import org.neo4j.driver.v1.GraphDatabase;


public class DBConnect {

    public static String uriDb = "bolt://localhost:7687";
    public static String uriUser ="http://localhost:8080";
    public static Config config = Config.builder().withoutEncryption().build();
    public static Driver driver = GraphDatabase.driver(uriDb, AuthTokens.basic("neo4j","12345678"),config);

    public static String convert(InputStream inputStream) throws IOException {
        try (BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
            return br.lines().collect(Collectors.joining(System.lineSeparator()));
        }
    }

}
```

```
App (1) [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Jul 22, 2024, 2:57:34 a.m.) [pid: 4424]
Server started on port 8080...
```

- Actor class to add actors to the database

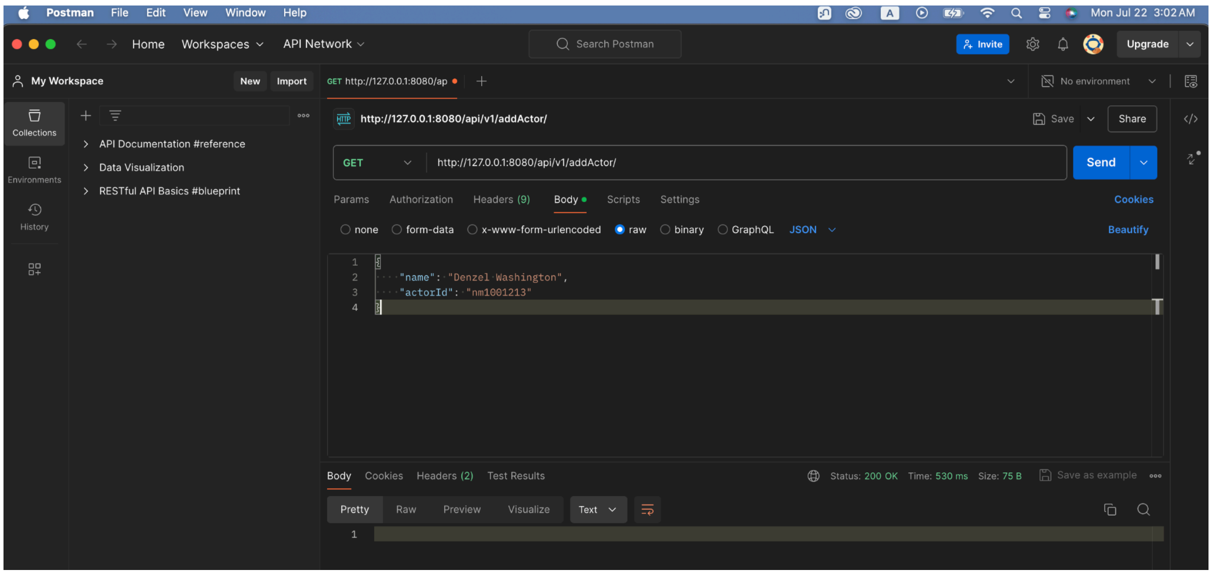


3311 - Project/src/main/java/ca/yorku/eecs/Actor.java - Eclipse IDE

```java
import com.sun.net.httpserver.HttpHandler;

public class Actor implements HttpHandler{

    public Actor() {

    }

    public void handle(HttpExchange r) {
        try {
            if (r.getRequestMethod().equals("GET")) {
                handlePost(r);
            }else{
                r.sendResponseHeaders(404, -1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void handlePost(HttpExchange r) throws IOException, JSONException{

        String body = DBConnect.convert(r.getRequestBody());
        JSONObject deserialized = new JSONObject(body);
        int statusCode = 0;
        String name;
        String actorId;


        if (deserialized.has("name"))
            name = deserialized.getString("name");
        else
            name = "";
        if (deserialized.has("actorId"))
            actorId = deserialized.getString("actorId");
        else
            actorId = "";

        System.out.println("INPUTS: name: "+name+" actorId: "+actorId);


        try (Session session = DBConnect.driver.session()) {

            session.run("CREATE (a:Actor {name:$name, actorId:$actorId});", parameters("name", name, "actorId", actorId));

                System.out.println("The Neo4j transaction ran");
                statusCode = 200;
        }
        catch( Exception e ) {
            System.err.println("Caught Exception: " + e.getMessage());
            statusCode = 500;
        }
        r.sendResponseHeaders(statusCode, -1);
```

Writable   Smart Insert   65 : 9 : 1657

- Postman used to test the GET request, 200 OK received



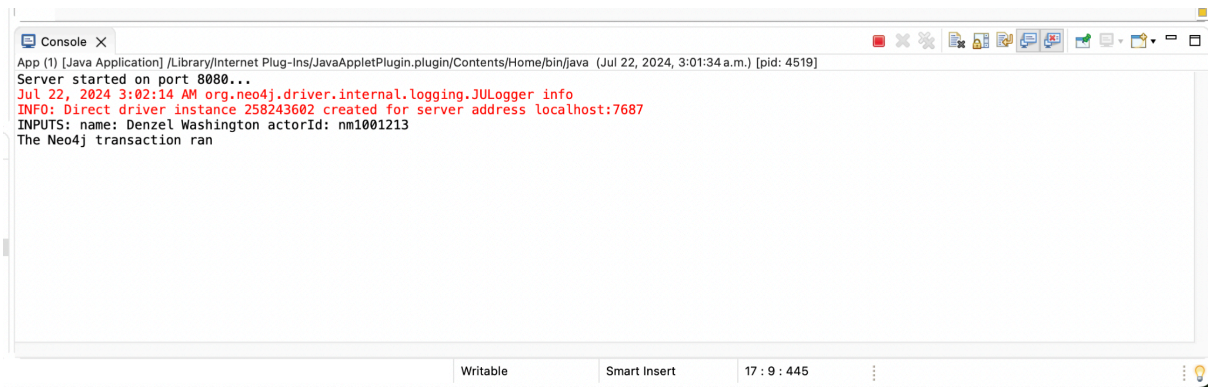
```
GET http://127.0.0.1:8080/api/v1/addActor/
```

Body (raw, JSON):

```
{
    "name": "Denzel Washington",
    "actorId": "nm1001213"
}
```

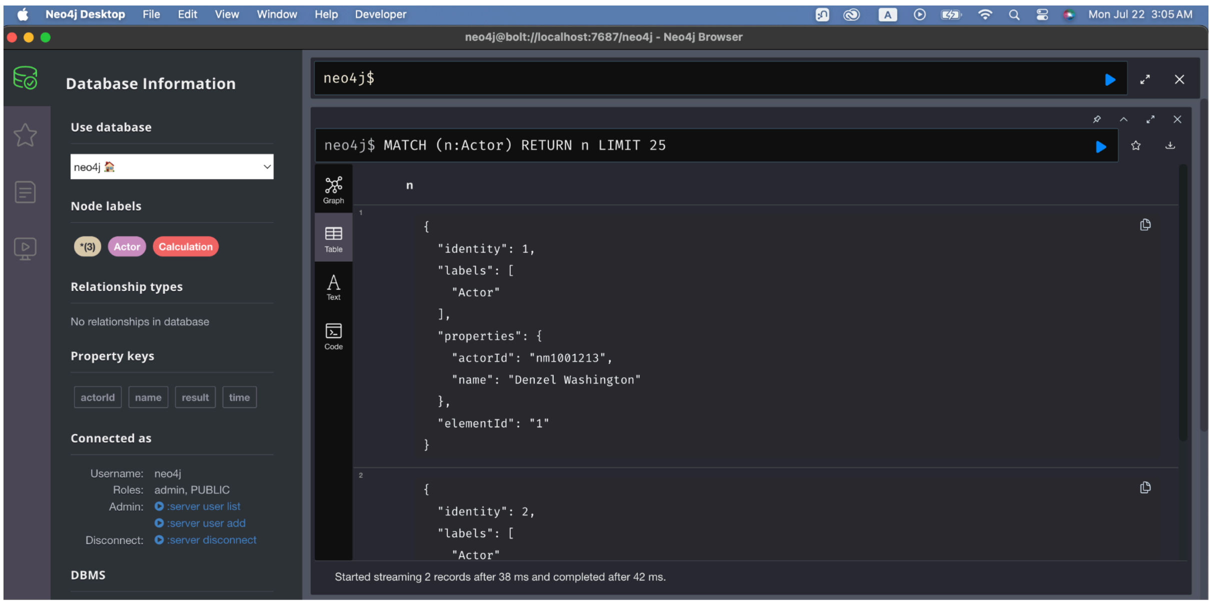
Status: 200 OK Time: 530 ms Size: 75 B

- Eclipse console showing the input received



```
App (1) [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Jul 22, 2024, 3:01:34 a.m.) [pid: 4519]
Server started on port 8080...
Jul 22, 2024 3:02:14 AM org.neo4j.driver.internal.logging.JULogger info
INFO: Direct driver instance 258243602 created for server address localhost:7687
INPUTS: name: Denzel Washington actorId: nm1001213
The Neo4j transaction ran
```

- Neo4j showing the actor was added to the database.



```
neo4j$

neo4j$ MATCH (n:Actor) RETURN n LIMIT 25

n
1
{
  "identity": 1,
  "labels": [
    "Actor"
  ],
  "properties": {
    "actorId": "nm1001213",
    "name": "Denzel Washington"
  },
  "elementId": "1"
}
2
{
  "identity": 2,
  "labels": [
    "Actor"

Started streaming 2 records after 38 ms and completed after 42 ms.
```