EECS 3311 - A

Project Phase 1 - Rest API and Neo4j

Bilal Mian - [redacted]

Contents

1. **Introduction**

Bilal Mian - My name is Bilal Mian and I am excited to learn how to develop REST API endpoints using Java and Neo4j, which is a graph database. Though I've used relational databases (MySQL, Postgres) and external API's before, this will be my first time using Neo4j to fully develop and test an API from scratch. I also want to further my effectiveness with Git and Github in a team setting, as our group will be using this for version tracking our software project.

2. **Summary Description of Project**

This project, we believe, uses the law of "six degrees of separation" to calculate a "bacon number", which will show the relationship(s) that actor Kevin Bacon has to other actors through his movies. This phenomenon is when two people, say John and Alice, can be shown to have a relationship between each other through mutual relationships with other people. For example, John is friends with Bob, Bob with Jill, Jill with Rachel, Rachel with Sam, and Sam with Alice. Each person in this example can be represented by a node in a graph, and Alice has a "John number" of 4, because they are connected by 4 intermediate nodes, or people, in the graph. The original law states that "any two people can be shown to have a relationship in six degrees or less".

To demonstrate the "six degrees of Bacon", we will be using Java (with maven) to build and test a REST API, Neo4j graph database to store relationships, and a client-server application model that will allow for creating, reading, updating, and deleting (CRUD) data in the database. We are using Kevin Bacon, a Hollywood actor, as an example to see how he relates to other actors (Bacon number) through movies. In addition to the endpoints given in the project handout, we will develop three new features to extend the functionality of our app, and those are described below in our submission.

3. New Features for Development

Feature 1

- Add an endpoint on the movies on the API called “**getAllActorsInMovie**” that retrieves a list of all actors in a given movie. The movie name will be provided in the request body.

Feature 2

- Add an endpoint called “**getAllMoviesInYear**” to retrieve all movies that were released in a given year. The year will be provided in the request body.

Feature 3

- Add an endpoint called “**getAllMoviesWithActor**” that retrieves a list of all movies that an actor has acted in. The actor name will be provided in the request body.

4. Endpoints and Testing New Features

Feature 1

- GET /api/v1/getAllActorsInMovie
  - Description: This endpoint will return a list of all the actors in a specified movie.
  - Body Parameters:
    - movieId: String
  - Body Example:

```
{
  "movieId": "nm1113481"
}
```

  -
  - Response:
    - actorId: String
    - name: String
  - Response Body Example:

```
{
    [
       {
              "actorId": "nm1113231",
              "name":    "Keanu Reeves"
       },
       {      “actorId:  “nm1114432”,
              “name”:    “Donnie Yen”
       }
    ]
}
```

- Expected Response:
  - **200 OK** - For a successful retrieval from database
  - **400 BAD REQUEST** - If the request body is improperly formatted or missing required information
  - **404 NOT FOUND** - If there is no movie in the database that exists with that movieId.
  - **500 INTERNAL SERVER ERROR** - If retrieval was unsuccessful (Java Exception Thrown)
- Edge Cases:
  - N/A

Feature 2

- GET /api/v1/getAllMoviesInYear
    - Description: This endpoint is to get a list of all movies released in a given year.
    - Body Parameters:
        - movieId: String
    - Body Example:

```
{
  "release_year": "2019"
}
```

    -
    - Response:
        - movieId: String
        - movie: String
    - Response Body Example:

```
{
    [
       {
              "movieId": "nm1113632",
              "movie":    "The Gentlemen"
       },
       {      “movieId:  “nm1113499”,
              “movie”:    “The Irishman”
       }
    ]
}
```

- Expected Response:
    - **200 OK** - For a successful retrieval from database
    - **400 BAD REQUEST** - If the request body is improperly formatted or missing required information
    - **404 NOT FOUND** - If there is no movie in the database that exists with that release year.
    - **500 INTERNAL SERVER ERROR** - If retrieval was unsuccessful (Java Exception Thrown)
- Edge Cases:
    - If there were no movies released in a given year, return an empty list as a response.
    - Example:
        - { [] }

Feature 3

- GET /api/v1/getAllMoviesWithActor
    - Description: This endpoint will return a list of all movies that a given actor has acted in.
    - Body Parameters:
        - actorId: String
    - Body Example:

```
{
  "actorId": "nm1113632"
}
```

    -
    - Response:
        - movieId: String
        - movie: String
    - Response Body Example:

```
{
    [
       {
              "movieId": "nm1113632",
              "movie":    "Forrest Gump"
       },
       {      “movieId:  “nm1113499”,
              “movie”:    “The Green Mile”
       },
       {      “movieId:  “nm1113477”,
              “movie”:    “Sully”
       }
    ]
}
```

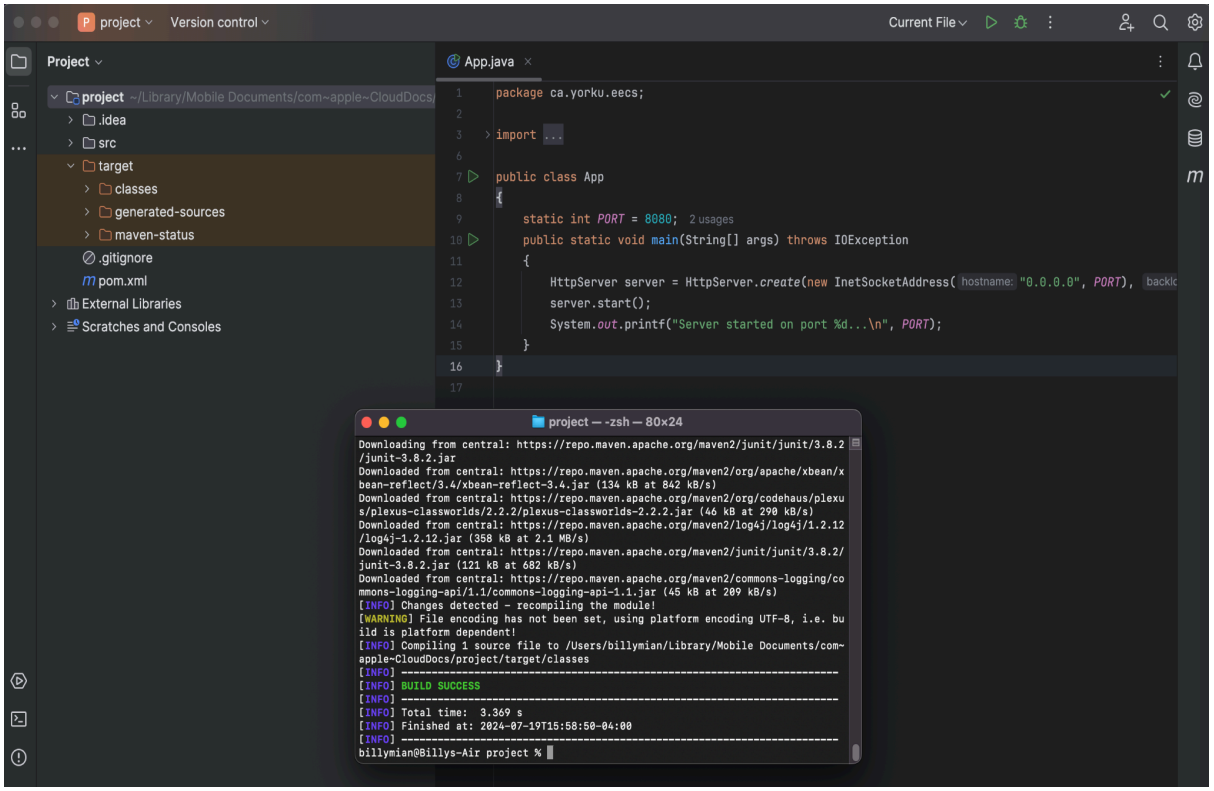
- Expected Response:
    - **200 OK** - For a successful retrieval from database
    - **400 BAD REQUEST** - If the request body is improperly formatted or missing required information
    - **404 NOT FOUND** - If there is no movie in the database that the specified actor has acted in.
    - **500 INTERNAL SERVER ERROR** - If retrieval was unsuccessful (Java Exception Thrown)
- Edge Cases:
    - If an actor did not act in any movie but they exist in the database, then return an empty list as a response.
    - Example:
        - { [] }

5. How We Will Test Our Features

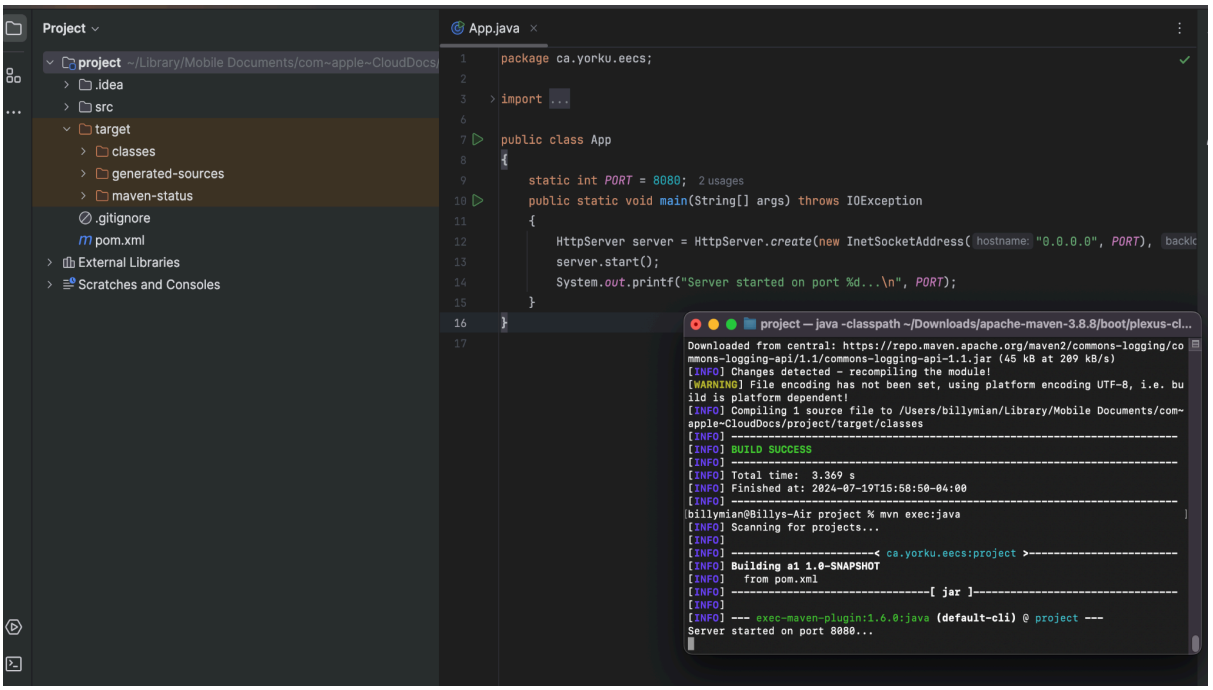
We will be using Postman desktop and Swagger to test our API requests and generate API documentation to validate it and test our features. Additionally, we will make use of some manual testing to ensure that all of the expected responses work as required by changing the response body as needed. In our Java app, we will investigate testing libraries to test our features, specifically unit testing frameworks such as JUnit.

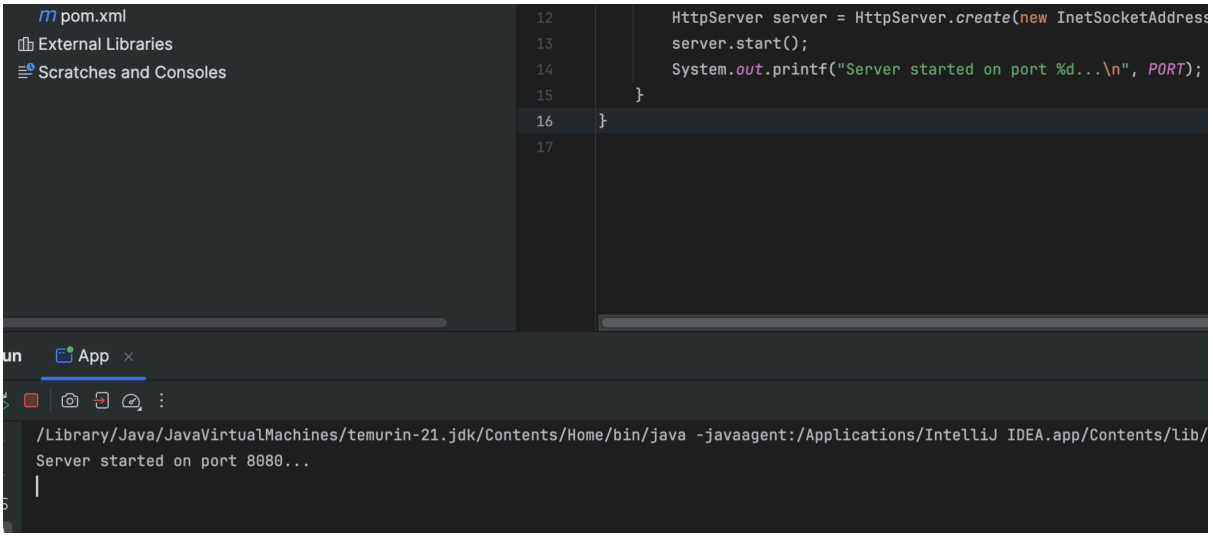
6. Project Setup with Screenshots

We used Java 1.8, Neo4J 4+, Maven and IntelliJ for our main tech stack. We have included a few screenshots of successful compilation of the project and the folder structure after running the project.

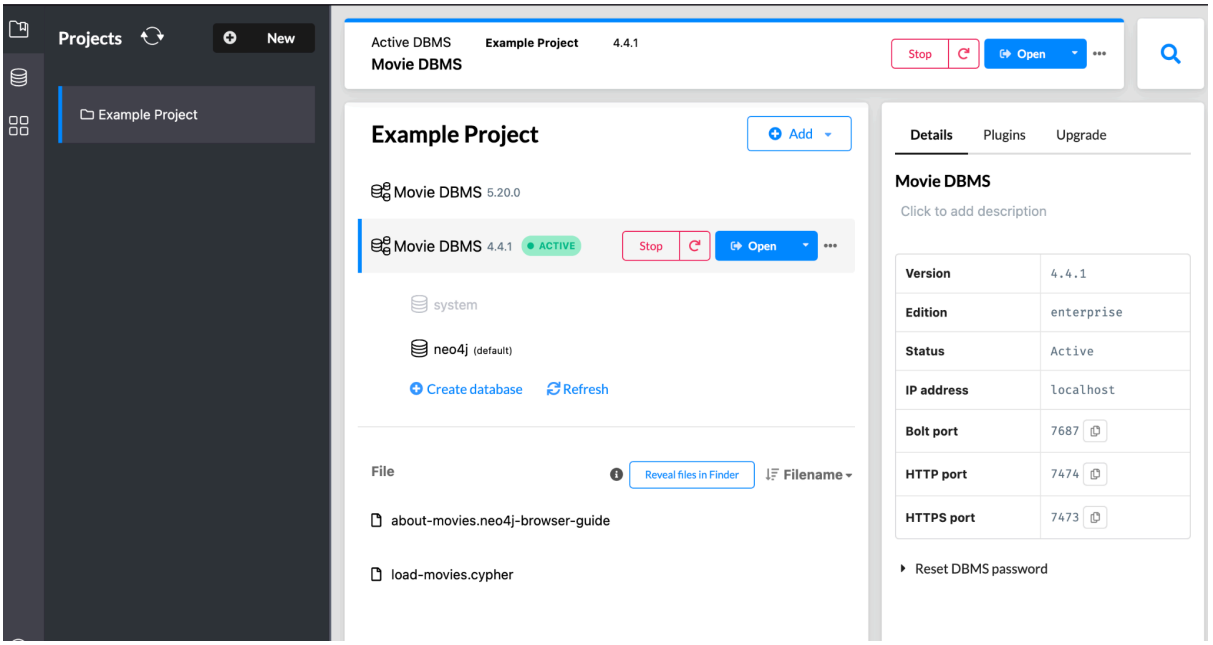


Compiling the project successfully.





Building the project successfully with the server started.



Starting the database successfully and loaded the cypher file.