**New Feature Outline**

**The feature: Actor retrieval by age.**

**Introduction**

The implementation increases the abilities of the project by allowing the user to add an age parameter when adding actors. This opens up the possibility to filter out the actors by their age. So with the new function the user can enter the an age and all of the actors with that specific age will be returned to the user.

1. New addActor including the age
   1. Endpoint: /api/v1/addMovie
   2. Request Method: PUT
   3. Body: {“name”: “John Doe”,

“actorId” : “123”,

“age”: “35”}

* 1. Response: usual status code
  2. How we have joined our feature with the existing addActor endpoint is by not making it mandatory to add an age of the actor. If the age of the actor is not included the age property will not be created for that node and if the age property is included in the body than the age will be included in the node. This way if the user wants to filter the actors by age they are able to do so.

1. New getActorsByAge function
   1. Endpoint: /api/v1/getActorsByAge
   2. Request Method: GET
   3. The age is required in the parameters when sending request
      1. **For example:** GET /api/v1/getActorsByAge?age=30
   4. Response: array of actor names
      1. **For example:** {“actors”:[“John Doe”, “John Cena”]} and the status code

**Testing**

The new feature is created and manually tested through CLI. But in order to thoroughly test the feature we use the robot framework with extensive tests designed to make sure the feature is working with no errors and bugs included.

So there are series of test cases to test for functionality and in this section we will go through the names of the test cases and a quick overview of what it provides so it is easier to track them down in the file itself.

1. Adding Actors with age
   1. addActorswithAgeMariya
   2. addActorswithAgeFrank
   3. addActorswithAgePass
      1. The tests above test for the PUT functionality with the add age feature included, so the name of actor, id, and age with none of the values being empty is submitted in the body and the response should be of 200 status code.
   4. addActorswithEmptyAgePass
      1. In the following test, the age property is left empty because even with the age property being empty the user should be able to add an actor since the age is not mandatory. The system should respond with code 200.
   5. addActorwithAgeFail
      1. With this test, a duplicate actor is being attempted to add into the database to see if the system handles duplicates correctly with a status code of 400.
2. Getting Actors with age
   1. getActorsbyAgePass
      1. The test runs the get endpoint for the feature and expects to retrieve actors with he age specified. In this test the age specified is 27 and the actors Mariya and Frank are age 27, so those two actors should appear in the response along with a successful status code of 200.
   2. getActorsbyAgeFail
      1. This test runs the get endpoint for the feature and tests how the system handles invalid inputs. By leaving the age property blank it is expected that the system communicates invalid inputs to the user with return of status code
   3. getActorsbyAgeFail404
      1. For this test, it is required the system communicates when an age is specified but no actor with that age is stored in the database. For this situation the system should return a status code of 404 to communicate this to the user.

**Conclusion**

In conclusion, by including the actors age in the adActor function we allow the user a convenient way to find actors by a specific age. Our feature is further testing using the robot framework which comprehensively tests all the features and cases for our function.