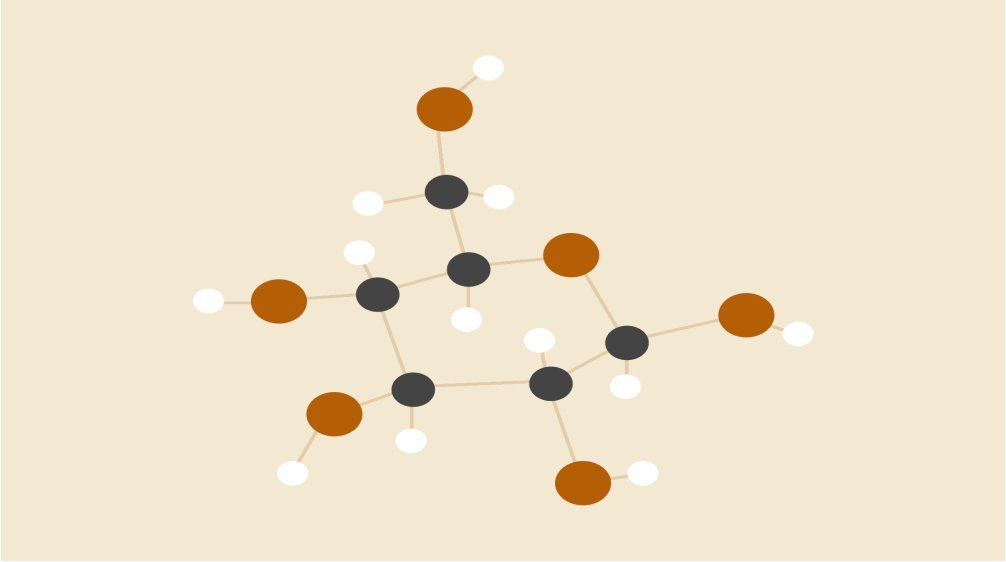
PROJECT REPORT

Movie-Book Recommender System



Pratyush Agarwal - 180050078

Raaghav Raaj - 180050082

Vrinda Jindal - 180050120

Yash Gupta - 180050121

INTRODUCTION

The **Movie-Book Cross-Domain Recommender System** is a Database based application that provides the user with recommendations of movies on the basis of the movies, books and genres explored and rated by him.

Domains here refer to Books and Movies and the recommender system will provide suggestions for the movies including the user's rating on other Movies and Books and similarly it suggests books.

Commonly, the books do not have much of the rating database. Hence, we will use the movie choices, and the movie ratings to suggest books. Also, the available book ratings will be used to provide movie suggestions.

The database used in the application is a Graph Database as the backend computational framework and React JS for the frontend which involves a login/registration system and contains a database of a large number of movies, books and existing users.

REQUIREMENTS

1. **USER DETAILS**
    a. The user is asked to provide credentials while signing up. These include a **unique** username and password.
    b. User details are not editable. (We prevent username and password modification for simplicity, and since we are working with localhost only)

2. **SOCIAL NETWORK:**
    a. A user can follow/unfollow another user, and it is one way i.e. A follows B, does not imply B follows A.
    b. A user can search for other users and see their rated movies/books and the users they follow (ie user A can search for user B and see who all B follows and B's rated movies/books)
    c. Users also get recommendations based on the users they are following (movies/books they've rated)
3. **PROVIDING MOVIE/BOOK RECOMMENDATIONS**
    a. Users would be provided with recommendations of movies on the basis of their previously provided ratings, the

genres they like, and the overall ratings of books/movies

b. As mentioned in social network , users also get recommendations based on the users they are following (movies/books they've rated)

4. **RATING A BOOK/MOVIE**
    a. Users can update the database by providing a number rating (on a scale of 0-5) to a book/movie
5. **LIKING/DISLIKING A GENRE**
    a. Users can like/dislike any GENRE and see related movies/books and see other related genres to a particular genre
6. **SEARCHING FOR A MOVIE/BOOK/GENRE**
    a. Users can search for a particular movie/book which would take them to the details page of that. The page also contains related movies and books to that particular movie/book.
7. **RELATED PEOPLE**
    a. While searching for movie/book, user can see cast (writers, producers) of that movie/book, and visit that person profile page and see other related movies/books and related people (ie people who he/she has worked with etc)
8. **CROSS-DOMAIN RECOMMENDATIONS**
    a. Users will receive suggestions for books based on the movies they explore, like and similarly they receive suggestions of movies based on the books rated by them.
9. **GETTING DETAILS AND GENERAL INFORMATION**
    a. Users can see details of a movie/book like the Plot, Cast, Director, Writer, etc.
10. **ANALYTICS**
    a. Spotlight feature where user can see popular movies, book and genres

USE CASES

There are 2 roles in our system.

1. The administrator (admin)
2. The target user

We now describe the roles classified on the basis of primary actors (the admin or the user).

The primary actor is selected on the basis of a 2 choice question we ask of whether the person is admin (directed to a login screen) or a consumer of the service (directed to a signup/login screen).

Primary Actor - ADMIN

- Logging on to the system
    - description
        - supply the admin (already preset) username and password to login into the system
        - takes the admin to the “main” page (dashboard) of the service
    - triggers
        - None
    - precondition
        - an admin user of the service who knows the login credentials
    - main success path
        - after the entering of the required data, the user is directed to the “main” admin page of the service
    - constraints & exceptions
        - single pair of username/password (preset) for the admin of the system. if no matches are found, we reject the login attempt.
    - postconditions
        - the user is in control of the full database

The following use cases are do-able from the main admin page (where we reach upon login) of the web application.

- Modifying the contents of the Database. For eg, adding/removing users/movies from the system
    - description

        - make fundamental changes to the database
        - add or remove users from the database
        - Add remove movies/books/reviews from the database
    - triggers
        - If a user is removed, the tuples corresponding to the user is removed after removing the user
        - If movie/book is deleted : all tuples corresponding to that movie deleted
    - precondition
        - an admin user of the service who knows the login credentials
    - main success path
        - The full schema is available to the admin. He can edit/query for whatever he wishes too.
        - can also add new parameters (for example new analytics) to the database
    - constraints & exceptions
        - single pair of username/password (preset) for the admin of the system. if no matches are found, we reject the login attempt
    - postconditions
        - whatever edits admin makes to the database
- Logging out of the system
    - description
        - logs out of the dashboard when done with the service
    - triggers
        - none
    - precondition
        - an admin of the service who is logged in and is on the admin page
    - main success path
        - the admin selects the ‘logout’ on the main page
        - is directed back to the login page
    - constraints & exceptions
        - none
    - postconditions
        - logged out. cannot make any changes before logging in again.

Primary Actor - TARGET USER

- Signing up for the service
    - description

        - select a username and password for accessing the service in the future
    - triggers
        - Password is hashed after accepting it from the user
    - precondition
        - the user is new to the service (no personal data of the user is currently in the database)
    - main success path
        - after the entering of the required data, the user is directed to the “main” page of the service
    - constraints & exceptions
        - INSERT authorization required on the user relation
        - username is rejected if already present in the database and the user is asked to try again. if no matches are found, the username and hashed password is stored.
    - postconditions
        - user details are present in the database. to be precise, the username, password, preferred genres and language data is inserted.
        - now user can log in (not sign up) into the service with the credentials
- Logging on to the system
    - description
        - supply username and password to login into the system
        - takes the user to the “main” page (dashboard) of the service
    - triggers
        - Password is hashed after accepting it from the user and matched with the stores hashed password in database
    - precondition
        - an existing user of the service who knows his login credentials
    - main success path
        - after the entering of the required data, the user is directed to the “main” page of the service
    - constraints & exceptions
        - SELECT authorization required on the user relation
        - if no matches are found, we reject the login attempt
    - postconditions
        - none

All further use-cases can be accessed using the “main” page (dashboard) of the service.

- Insert his/her ratings data i.e. rating a movie/book
    - description

        - insert movie/book rating
        - Or update the rating of previously reviewed book/movie
    - triggers
        - None
    - precondition
        - an existing user of the service who is logged in and is on the dashboard
    - main success path
        - the user selects the ‘rate movies/books’ on the main page
        - then update the desired fields i.e. book (or movie), a 0-5 star rating
        - saves his changes and is directed back to the dashboard
    - constraints & exceptions
        - INSERT/UPDATE authorization required on the movies rating relation
        - cannot update anything related to other users’ data
    - postconditions
        - the database now carries the desired updates of the user
- Following(Follow/Unfollow) Users(Like/Unlike Genre)
    - description
        - follow/unfollow another user (like/unlike a genre)
    - triggers
        - none
    - precondition
        - an existing user of the service who is logged in and is on the dashboard (An existing Genre in DB)
    - main success path
        - search for a person by name and follow/unfollow him(Search for a genre and like/unlike it)
    - constraints & exceptions
        - UPDATE/INSERT authorization required on follower relation(on like_genre relation)
        - cannot update anything related to other users’ data
    - postconditions
        - the database now carries the desired followers for the user(the desired likings of genres of the user)
        - he can now see their preferences, reviews on a priority basis(Get recommendations based on genres liked)
- Searching for a movie/book
    - description
        - Search for a known movie or book
        - get to know similar movie/books and the friends’ numerical ratings

    - triggers
        - none
    - precondition
        - an existing user of the service who is logged in and is on the dashboard
    - main success path
        - the user selects the ‘search movie/book’ on the main page
        - then provides a name to search for the movie.
        - the user views the shown details of the movie
        - goes back and is directed back to the dashboard
    - constraints & exceptions
        - book and movie should already be in the database. otherwise, the keyword matches with nothing.
    - postconditions
        - the database now carries the desired updates of the user
- Querying for movie/book recommendations
    - description
        - the user wants the service to recommend him some books or movies
    - triggers
        - none
    - precondition
        - an existing user of the service who is logged in and is on the dashboard
    - main success path
        - the user goes to the profile page.
        - he is directed to a page that shows recommendations based on the data he has provided
        - notes down the output and is directed back to the dashboard
    - constraints & exceptions
        - SELECT authorization required on the movies relation
        - cannot update anything related to other users’ data
    - postconditions
        - no change on the underlying database if the user just sees the recommendations
- Logging out of the system
    - description
        - logs out of the dashboard when done with the service
    - triggers
        - none

- precondition
  - an existing user of the service who is logged in and is on the dashboard
- main success path
  - the user selects the 'logout' on the main page
  - is directed back to the login/signup page
- constraints & exceptions
  - none
- postconditions
  - logged out. cannot make any changes before logging in again.

TEST CASES (RESULTS ARE COLOR CODED)

Requirement it is testing	Use case it is testing	Inputs to the test	Expected outputs	Expected backend changes (change to DB state for example)	Test procedure (which screen to load, what action to take etc.)
Welcome Page	login	click on "login" button	go to **Login Page**	no changes to the database.	login page screen should be loaded
	signup	click on "signup" button	go to **Signup Page**	no changes to the database.	signup page screen should be loaded
Login Page	Home button	Click on home page	Go to **Welcome Page**	No change to database	Loads the **Welcome Page**
	Submit (User)	Username Eg "pratyush1019" Password Eg	Success: Proceed to **successful login page**(see screen design) if the username is present	No change to database-	Success: Proceed to **successful login page** Failure : Clear the username and password filled and reload the same Login page with error

		“popcorn”	and the password is correct Failures : Username can’t be “admin” for a user, Username not found(account not created for the person), password incorrect for the given username		message
	Submit(admin)	Username : “admin” Password : admin’s password(hardcoded beforehand by db admin)	Success: Proceed to **successful admin login page**(see screen design and notes, if the user is admin, an admin hyperlink appears on each page which leads to the admin page) if username is “admin” and the password is correct Failures : Username not equal to “admin”, incorrect password	No change to database	Success: Proceed to successful admin login page(same as for normal user, but an additional admin hyperlink appears on each page which leads to the admin page) Failure : Clear the username and password filled and reload the same page with error msg
	Go to Create a new account	Username, password can be provided but they are ignored	Go to the **Signup page**	No change to database	Load the **Signup page**

Signup Page	Creating a new account	Username Eg “pratyush1019” Password Eg “popcorn” Confirm password Eg “popcorn”	Success : If the username is not already taken (and not “admin”). Also if the password and confirm password are the same, then create a new account and go to **Sign Up Successful Page**. Failure : If username is already taken or it is “admin” ,and/ or the password and confirm password don’t match, display error message and stay on same page	Success: Update the USERS table to add the new username and store its hashed password Failure : No change	Success: go to **Sign Up Successful Page**. Failure: Clear the username and password fields filled and reload the same page with error msg
Successful Login Page	Search - by name, by genre	-type in name/genre -click on search	Success: If the genre or the movie/book name exists in the DB, Go to the page of the movie/book/genre Failure : If the name is not present in the DB, display error page	No changes in the database	Success: Loads the corresponding entity(movie/book) page or genre page Failure : Display error message and stay on the same page
	Recommended entities	-click the link of the entity	Go to the page of the movie/book	No changes in the database	Loads the corresponding entity(movie/book) page

	Profile Button	-click on the profile button on top right corner	Go to the profile of the user	No change in the database	Loads the profile page of the user.
	Spotlight	-click the link of the entity	Go to the page of the movie/book	No changes in the database	Loads the corresponding entity(movie/book) page
Signup Successful Page	Profile button	-click on the profile button	Go to the **User Profile Page** for the current user	No changes	Loads the **User Profile Page**
	Logout button	-click on the logout button	Logout and go to the **Welcome Page**	No changes	Loads the **Welcome Page**
	Homepage button	-click on the homepage button	Go to the **Successful Login Page**	No changes	Loads the **Successful Login Page**
Entity Page	rating - (logged in) the highlighted button among the 5 reflects the rating	click on one of the 5 star buttons	set the rating according to the star clicked. the highlighted star (which shows the current rating) now reflects the new rating	sets/updates the rating for that user and entity in the DB relationship	reloads the same **Entity Page** with the new rating
	rating - (not logged in)	button won't appear, so no action possible	button won't appear	No change	button won't appear
	genre - for that	click on the	go to that specific	no change to the	Loads the corresponding **Genre**

	entity whose page you're on	genre button	**Genre Page**	database	**Page**.
	written/directed by	click on one of the shown person buttons	go to that specific **Person Page**	no change to the database	Loads the corresponding **Person Page**.
	cast	click on one of the shown person buttons	go to that specific **Person Page**	no change to the database	Loads the corresponding **Person Page.**
	Related Movies/book (**Analytics**)	Get movies and books similar to the given entity(book/movie) Note : If the entity is a movie, we get similar books(cross-domain) as well as movies. Similarly if the entity is a book, we show similar books and movies(cross-domain)	Go to the specific **Entity Page**	No change	Loads the corresponding **Entity Page**
	Subjective Reviews	Enter your subjective review	The entity page should update with the review	Review, movie id, user id is added in the database	Same page, with the added review.

Genre Page	Like/Dislike (logged in)	-click on "Like/Dislike"	"Like" button changes to "Dislike", and vice-versa.	<userID,genreID> is added to the relation **LIKES** when Likes is clicked and <userID,genreID> is removed from **LIKES** when Dislikes is clicked	The screen remains the same. The button "Like" changes to "Dislike", and vice-versa.
	Like/Dislike (logged out)	The option won't appear in this case, so no action possible	The option won't appear in this case	No change	The option won't appear in this case.
	Recommended entities of the genre	-click the link of the entity	Go to the **Entity Page** of the movie/book	No changes in the database	Loads the corresponding **Entity Page**.
	Recommended genres related to this genre(**Analytics**)	-click the link of the genre	Go to the **Genre Page** of selected genre	No changes in the database	Loads the corresponding **Genre Page**.
Person Page	directed	click on one of the directed movie buttons	go to that specific **Movie Page**	no change to the database	Loads the corresponding **Entity Page**.
	acted in	click on one of the acted in movie buttons	go to that specific **Movie Page**	no change to the database	Loads the corresponding **Entity Page**.
	written	click on one of the written movie/book	go to that specific **Entity Page**	no change to the database	Loads the corresponding **Entity Page**.

		buttons			
	Related People	Click on the link of the related persons	Go to that **Person Page**	No change in the database	Loads the corresponding Person Page
My Profile Page	Rated movies/books	Click on the link to the movie/book	Go to that **Entity Page**	No change in the database	Loads the corresponding Entity Page
	Update rating	Select the number of stars out of 5	That many stars are highlighted	The tuple in the RATED relation is updated.	The rating changes and page does not change
	Recommendations (**ANALYTICS**)	Click on the link to the movie/book	Go to that **Entity Page**	No change in the database	Loads the corresponding **Entity Page**
	What Friends are Enjoying -this is the top rated movies/books of the users we follow	Click on the link to the movie/book	Go to that **Entity Page**	No change in the database	Loads the corresponding **Entity Page**
User Profile Page	home	-click the “home” button	Go to **Successful Login Page**	No change to the database.	Loads the **Successful Login Page**
	Follow/Unfollow button	-click the “follow/unfollow” button	“Follow” button changes to “Unfollow” and vice-versa	**FOLLOWS** relation is updated accordingly. Follow - <user1,user2> is added	Screen does not change and the “follow” button changes to “unfollow” and vice-versa.

			(only when the user is on some other users page)	Unfollow - <user1, user2> is removed	
	Search user	-input Username Eg “pratyush1019” - click “search”	Success: Go to the **User Profile Page** of the username searched if it exists in the DB admin(username : “admin”) can also be searched Failure: If the name does not exist, it displays an error page.	No change to the database. (Success/Failure)	Success: Loads the User Profile Page of the corresponding user. Failure: Display error msg and reload the same User Profile Page
	Users that the given user follows	-click on the username	Go to the **User Profile Page** of the username.	No change to the database.	Loads the **User Profile Page** of the corresponding user.
	Movies/books rated by the user	-click on the link to the movie/book	Go the **Entity Page**	No change to the database	Loads the **Entity Page** of the corresponding movie/book.
NOTE: The admin page is reached after clicking on an admin button which appears on every page if the username is admin. The button details are given below.					
-		click on the admin button	Go to the **Admin Page.** No failure possible as button presence is conditional on the username	No change to the database	Load the **Admin Page**

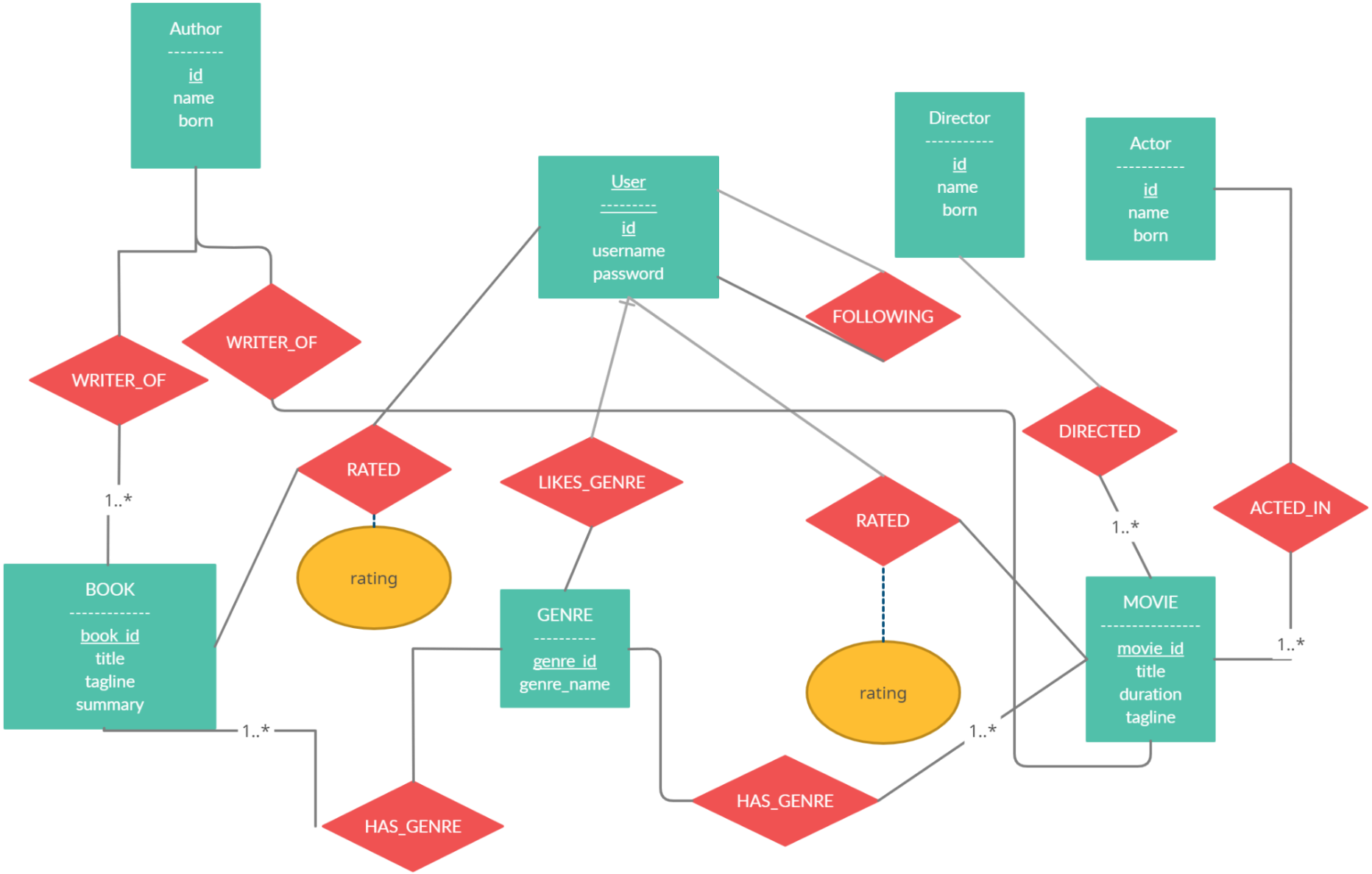
Admin Page - only when username is admin	Add movie/book	Entity name Entity cast Entity genre	Success: If the corresponding cast people and genre name exists in the DB already, then add movie to DB and reload the same Admin Page Failure: If any of the above conditions is not met : ie if one or more persons of the cast don't exist in DB or the genre name doesn't exist in DB, then reload the same page	Success : add the node of the movie to the DB and add the corresponding relations (cast, genre) to the DB Failure: No change	Success: Load the **Admin Page** again Failure : Display error msg and reload the **Admin Page** again
	Delete movie/book	Entity name	Success: If the movie/book name exists, then delete the movie/book from the DB and reload the same page. Failure: If the movie/book does not exist, then reload the same page with an error message.	Success : Delete the node of the entity from the DB and the corresponding relations (cast, genre) from the DB Failure: No change	Success: Load the **Admin Page** again Failure : Display error msg and reload the **Admin Page** again
	Remove user	username	Success: If the user name exists, then delete the user from	Success : Delete the node of the entity from the DB and the	Success: Load the **Admin Page** again

			the DB and reload the same page. Failure: If the user does not exist, then reload the same page with an error message	corresponding relations from the DB Failure: No change	Failure : Display error msg and reload the **Admin Page** again
	others	run any arbitrary neo4j query against the database.	Success: the query is valid and error free, it is executed directly against the database. Failure: If the query is invalid, an error message is displayed.	Success: whatever the query dictates Failure: No change	Success: Load the **Admin Page** again and display the result Failure : Display error msg and reload the **Admin Page** again

Concurrency Control : By design choice we haven't modelled concurrent access of two users to the database. However we can use two different tabs but on refreshing one tab, we will get to the latest state (current or the other tab). We have made this choice since we are working with localhost (and not remote server). Note that if this is hosted as a website the multiple users setting is expected to work perfectly.

DESIGN

Normalisation: The following is the normalised graph DB. Our graph DB in neo4j is essentially the same, only there we first populate the nodes and then the edges. Kindly treat the boxes as nodes with their attributes and -diamonds- as edges (relationships) of the graph with their associated attributes(in some cases):



1. As given in the article, our graph DB schema is already in 3NF form i.e. edge normalised.

2. Since the **relations** are only binary, and for any relationship between nodes A and B, the primary key of these two combined forms the primary key of the relationship (intuitively speaking for graph DB as well).
   We don't normalise our db schema further since being in 3NF(edge normalised) is sufficiently normalised for our purpose as it is already quite less redundant.

3. We don't normalise it further to AA-normalised or vertex-normalised since that isn't needed and we don't wish to create separate nodes(global-value singletons) for each attribute, because that takes a performance and efficiency hit for our purpose.

4. Our attributes, say, movie_title, summary, tagline are generally linked with one movie only (generally, in very few cases , incase of sequels tagline/title can be the same). Hence, creating a separate node for their values and then establishing edges between movies(/books) and these attributes gets inefficient and so we keep our schema as edge-normalised only.

5. Ratings for books are normally distributed (as asked for in the feedback) because we could not find them in the original database.

⇨ We keep all the edges in separate relationships.

Examples

DIRECTED: This is a relationship between movies and director and has no other property, hence this is 3NF trivially since it has primary key (<id> , <movie_id>)

Similar for ACTED_IN, WRITER_OF

FOLLOWING : It is one-way characterised by (<id1>,<id2>). By one-way we mean, if user A follows user B, then B doesn't automatically follow A. A would get recommendations based on B's choices as well, but if B doesn't follow A, then A's choices would have no effect on recommendations given to B.

We are computing similarity on the fly i.e. by taking into account the ratings, genre, and crew/author(incase of movies/books). **We aren't maintaining explicit edges/relationships for it.**

See Getting list of similar entities (movie/books) in point4 below to see how we are providing similar movies/books.

For book_movie_similarity we are taking into account the similarity in genres, and rating of the users towards that particular book and movie, ie if a user rates the book and the movie with a similar rating, the similarity score would be higher for that book and movie.

We aren't explicitly maintaining book_movie_similarity, we are computing that on the fly by using similar genres and similar ratings of users.

USERS: This has two unique constraints ie username and userid, both of these are superkeys and hence this is still in 3NF/BCNF.

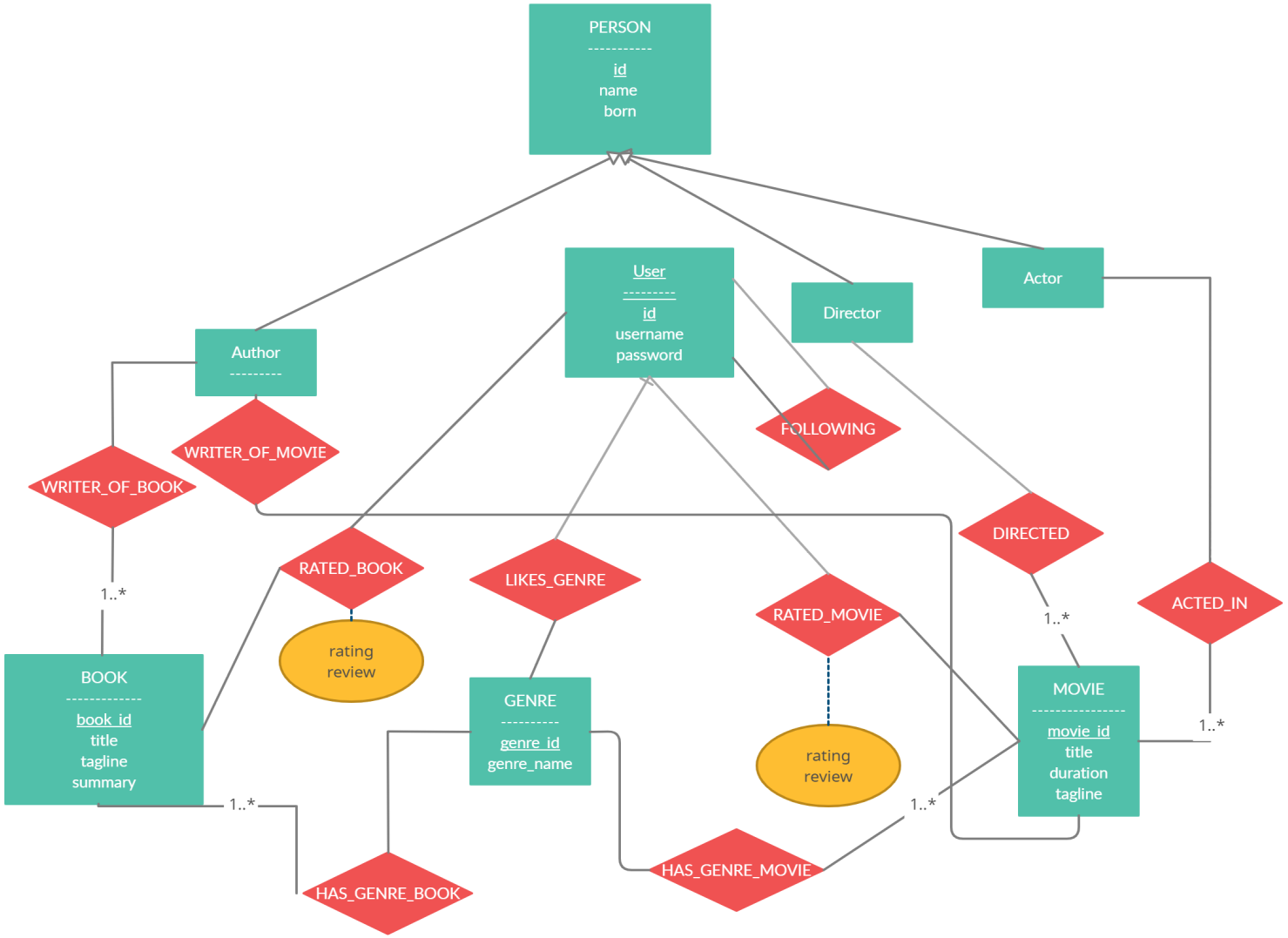
LIKES_GENRE: This is binary with (<user_id>, <genre_id>) as the primary key and is formed by these two attributes only

RATED: This is (<user_id>, <book_id>/<movie_id> , stars). Again (<user_id>, <book_id>/<movie_id>) form the primary key and the stars is the associated properties. Hence this is in 3NF/BCNF.

Similarly it can be shown for all edges and nodes, hence the graph schema is edge normalised i.e. 3NF.

Note that specialisation of Person(in the ER) to Author, director , actor is overlapping i.e. the same person (i.e. with the same id) can fulfill multiple such roles.

Before normalisation:


PERSON
id
name
born
User
id
username
password
Author
Director
Actor
WRITER_OF_MOVIE
WRITER_OF_BOOK
FOLLOWING
DIRECTED
ACTED_IN
RATED_BOOK
LIKES_GENRE
RATED_MOVIE
rating
review
rating
review
1..*
1..*
1..*
1..*
1..*
BOOK
book_id
title
tagline
summary
GENRE
genre_id
genre_name
MOVIE
movie_id
title
duration
tagline
HAS_GENRE_BOOK
HAS_GENRE_MOVIE

Technology Choices:

We have used the following technologies:

1. Neo4j Enterprise Edition - has support for multiple databases, and additional constraints. The state of the art graph database for the current generation.
2. Javascript (React) - Very good tutorials and exposure from outlab.

Indices:

Index Name	Type	Uniqueness	EntityType	LabelsOrTypes	Properties	State
book_name	BTREE	NONUNIQUE	NODE	["Book"]	["title"]	ONLINE
constraint_3a764435	BTREE	UNIQUE	NODE	["Movie"]	["id"]	ONLINE
constraint_47e7f15d	BTREE	UNIQUE	NODE	["User"]	["id"]	ONLINE
constraint_50656458	BTREE	UNIQUE	NODE	["Person"]	["id"]	ONLINE
constraint_860aead3	BTREE	UNIQUE	NODE	["Genre"]	["id"]	ONLINE
constraint_99d662d3	BTREE	UNIQUE	NODE	["User"]	["username"]	ONLINE
constraint_a58e82c6	BTREE	UNIQUE	NODE	["Book"]	["id"]	ONLINE
genre_name	BTREE	NONUNIQUE	NODE	["Genre"]	["name"]	ONLINE
movie_name	BTREE	NONUNIQUE	NODE	["Movie"]	["title"]	ONLINE

Indices

1. These are the indices in our database.

2. These indices correspond to the uniqueness constraints and were created automatically as a by-product of the DDL statements.

3. We create custom indices on **movie_name**, **book_name**, **genre_name** and **username** to enable faster search for these in our search pages.

4. Creating indexes comes at the cost of slower writes, but since there will be few writes to movies/genres/books (maybe none at all), this doesn't harm us, on the other hand searching becomes quite efficient.

Constraints:

```
//NODES
CREATE CONSTRAINT ON (n:Person) ASSERT n.id IS UNIQUE;

CREATE CONSTRAINT ON (n:Movie) ASSERT n.id IS UNIQUE;

CREATE CONSTRAINT ON (n:Book) ASSERT n.id IS UNIQUE;

CREATE CONSTRAINT ON (n:Genre) ASSERT n.id IS UNIQUE;

CREATE CONSTRAINT ON (n:User) ASSERT n.id IS UNIQUE;

CREATE CONSTRAINT ON (n:User) ASSERT n.username IS UNIQUE;

//RELATIONSHIPS
CREATE CONSTRAINT ON ()-[r:RATED_MOVIE]-() ASSERT exists(r.rating);

CREATE CONSTRAINT ON ()-[r:RATED_BOOK]-() ASSERT exists(r.rating);
```

We have uniqueness (in graph db) constraints on the nodes for specific attributes (including for all primary keys in the schema). For **USER**, we have the constraint that both user_id and username should be unique.

We have a constraint on **RATED** relationship(edge) that the "rating" attribute should exist i.e. it should not be null.

Currently neo4j allows:

3. Unique node property
4. Node property existence
5. Relationship property existence and
6. Node key constraints.

There are no relationship uniqueness constraints in neo4j. Between two nodes, only one relationship can exist of a particular type.

Conclusions:

We could complete everything we set out to do. Some minor functionality like subjective reviews and ability to edit user details were left out for the better use of time adding useful features. Additional features which weren't reported earlier were added like "Friends are Enjoying" and "Spotlight Genre" Section.

Git Link

https://github.com/yashgupta-7/rec_sys