CS235 - A2 & A3 Design Report - nbel113

Github Link:

https://github.com/nbel113/CS235-A3

Assignment 2 Info:

Cool New Feature:

- The ability for a user to edit and delete their own reviews.
    - The user can edit the movie reviewed and/or the review text and/or the score.
        - If a field is left blank in the edit review form (on the edit review page), the original review's value for that field is kept.
    - The user can delete a review.
        - The user is prompted in the delete review page if they want to delete the review, with two buttons (Yes and No).
    - After deleting a review, if the user tries to go back to the delete page by clicking on the back button in their browser, they will be redirected back to the homepage. This is done in order to prevent the web app from trying to access a deleted review and therefore crashing. I have not implemented this redirecting for the edit review page, since the review is still accessible without crashing.
    - If the user tries to create a review for someone else's account (by changing the user, replacing the value set for the 'user' query parameter (their username) with someone else's usually name), or tries to edit or delete another person's reviews (by attempting the mentioned approach), they will be redirected back to their own reviews listing page (aka their own 'reviews').

Key Design Decisions:

- I made several additions/changes to A1's code:
    - I consolidated the individual Actor, Director, Genre, Movie, Review, User and Watchlist classes into one file: 'domain/model.py.'. (the original .py files for each class are available at 'domain/indiv/')
        - This was done for better file organization, and allows several classes to be imported with just one line (`from movie_web_app.domain.model import ...`)
    - In movie_file_csv_reader.py, I added several lines to attach a movie's actors, genres, director, description and runtime to the relevant movie.
    - Within the model file, I added setters for: Movie's non-parameter properties (except for properties that can contain several objects within a list, such as 'actors' which can hold a list of Actor objects), and most of Review's properties (except for review_id, which is a property that we don't want to change).
        - The Movie setters are primarily used within the movie_file_csv_reader when processing each line (movie).
            - Using the line `movie.director = row['Director']` as an example, the value at the 'Director' column gets assessed as the movie's value for the director attribute. This allows a movie's director to be displayed in other parts of the web app (such as the list_movies page). For actors and genres, loops are used to append each Actor/Genre.
        - The setters in Review are used in the edit_review function within review_services, whenever a valid edit to a property needs to be obtained from the TempReview 'new_review' and applied to the original review.
    - In Review, I added:
        - a review_id property in order to individualize a review. Instead of storing the review as a query parameter, the id can be used. This also allows the web app to make reviews with the same movie title, review text and rating.

        - A ‘latest_edit’ property, which collects the time of the last edit of a review. If a review gets edited, this time (along with the time the review was created) gets displayed within the reviews listing page.
- For the edit feature, I added another class called TempReview, which is based on the Review class, but with some changes such as no type handling. It is designed to store input from the edit review form. If input for a property (eg review_text) is valid and not None or a blank field (eg "" or " "), the original review will have that property updated with the input.
    - Using this class reduces the number of parameters required for the edit_user function within review_services.py to just two parameters; the original review, and the temporary ‘new’ review. It also makes the function easier to modify if more properties need to be passed into TempReview (since you don’t need to change the number of parameters).
- The decision to have the adapters (repositories and services) and blueprints in their own dedicated folders (a_adapters and a_blueprints) was made so that they weren’t buried alongside the folders for the non-adapters or blueprints. This allows for more convenient access to said files while developing the web app.
- Along with reading from a .csv file of movies, I have also created a .csv file that contains one user. This is used primarily to help with several web app unit tests, and also allows for easier testing when physically interacting with the web app in a browser.
- In the register, create_review and edit_review pages, I have displayed the field requirements (e.g. a review’s text must not contain profanity), so that it is clearer as to what valid input should be like.
- For the search feature, I:
    - return the intersection of all field results with entries containing terms as substrings (eg ‘star’ in ‘Star Trek’),
    - allow searching to be case-insensitive
    - and allow input to contain several search terms by being comma separated (eg “the, end” for movie titles results in movies with either ‘the’ or ‘end’ in the title)

  in order to obtain as many relevant movies as possible.

Design Principles, Patterns Applied and Benefits Provided:

- I broke up my development over several days, with the goal of completing or at least implementing the majority of a few important features each day. If I get stale with the development of a feature (and if there are other features to implement within the grade level), I switch over to another feature, depending on how important the feature I am struggling with is. Important features included:
    - The listing of movies
    - The navigation
    - The search feature and it’s search capabilities
    - Implementing the authentication feature (register, login, logout)
    - Displaying and creating reviews
    - Editing reviews
    - Deleting reviews

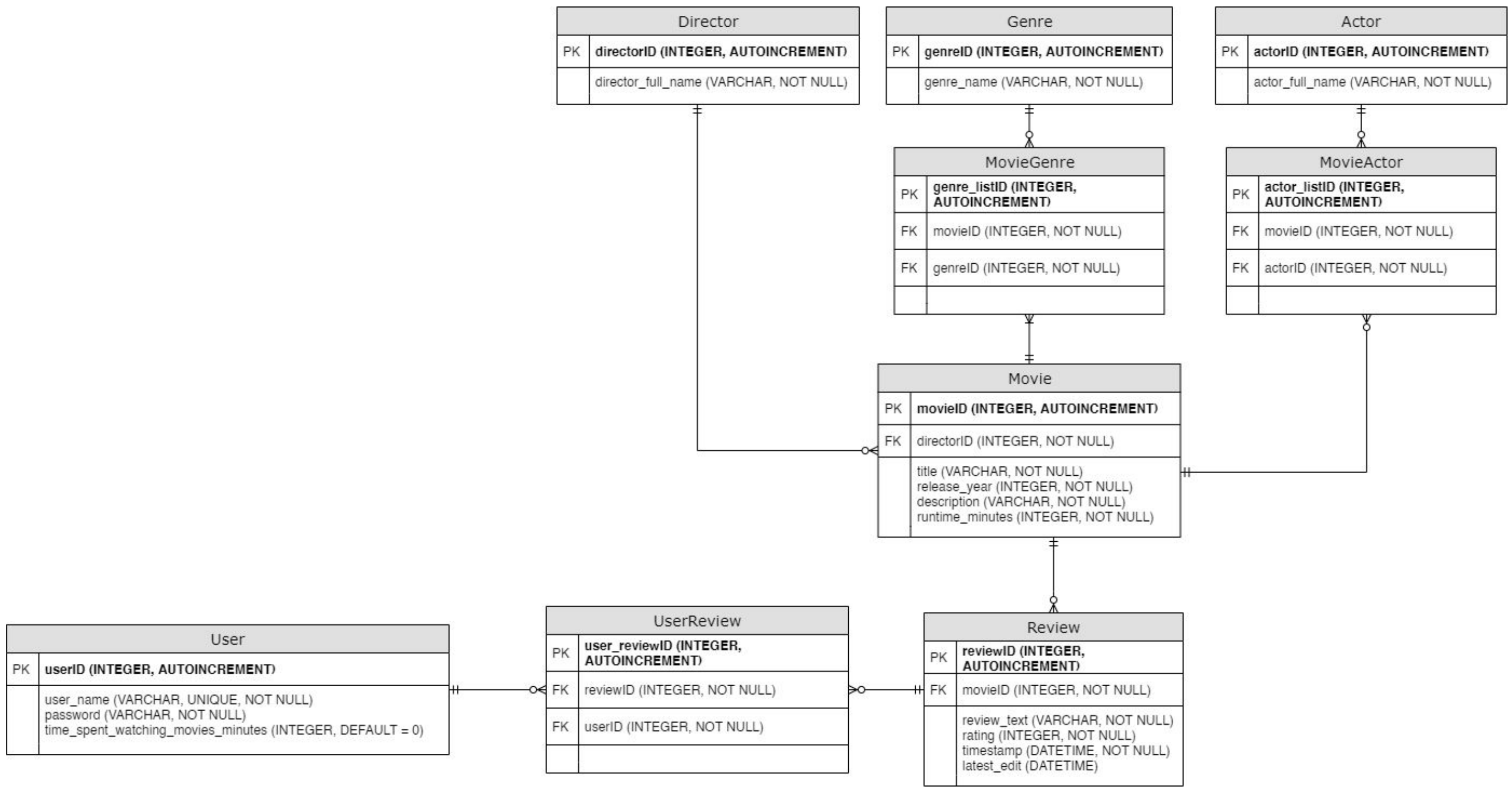
  This made development far less overwhelming, while ensuring that a significant amount of work was made each day.
- I did implement test-driven-development. This is where I went through the requirements from the brief, and created several tests (unit tests and end-to-end tests). With these tests, I had to improve my code with the goal of passing these tests.
    - It made me think more in terms of the brief, and helped with creating goals that are more tangible.
    - It also helped in ensuring that I didn’t deviate significantly from the brief.
    - However, I have to constantly ensure that the tests are written properly and with enough depth (via the assert statements that I write). This is so that they actually test the aspect to be tested and so that I can have confidence in the code I’ve written (in that I’m not getting a false positive).

        - Sometimes it was more helpful to create a list of tests to implement, implement a few, and then write the code to fulfill those tests, so that the process of ensuring the tests work properly is easier.
    - Note that in the commit ‘Grade B recommit’, tests were written for it beforehand, but didn’t get pushed. They were pushed in the commit ‘Grade A submission, with all tests’.
- I iteratively built up the program piece by piece (with tests to ensure each piece functions properly):
    - starting with the basic requirements for C:
        - displaying movies within the web app into a paginated table with 15 movies/pg
        - allowing the user to navigate through pages to access different movies.
        - Creating the requirements.txt and readme file
        - Creating the basic UI (HTML, CSS, Jinja), the basic web interface (entry points, use of HTTP protocol)
        - Create the movie data repository
    - Then requirements for B:
        - The search feature to search by actor, genre and or director. I added searching by movie title.
        - The authentication system (register, login, logout)
        - The ability for users to create movie reviews (along with displaying said movies), with signed cookies to ensure a logged in user can only write reviews
        - Use of Blueprints for the pages of the application
        - Use of WTForms for the authentication pages and the create review page.
    - Then the special feature for A (the editing and deletion of reviews)

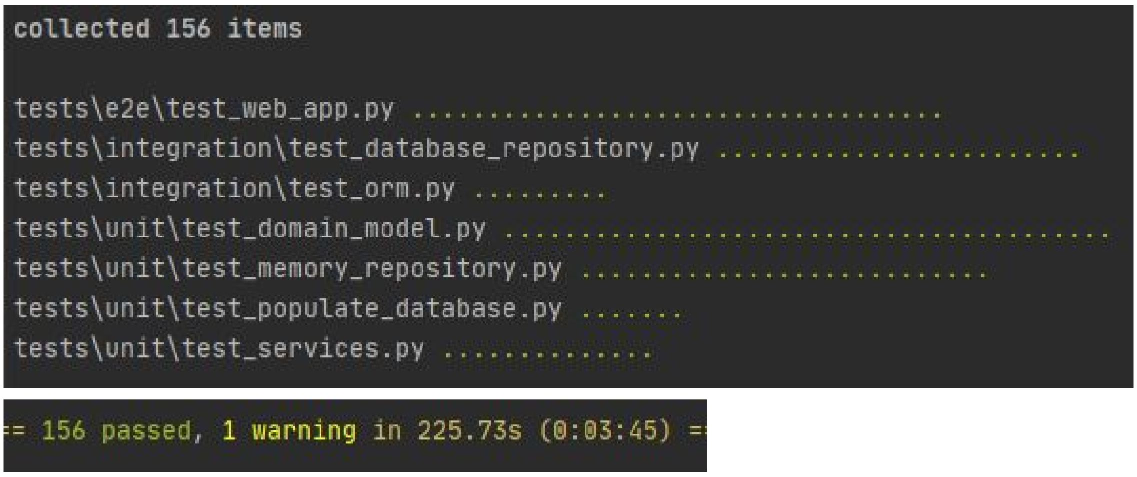
  Once a piece was completed, it was marked off as complete.

Assignment 3 Info:

<u>Database Schema Diagram (image available in folder)</u>


Director
PK directorID (INTEGER, AUTOINCREMENT)
director_full_name (VARCHAR, NOT NULL)
Genre
PK genreID (INTEGER, AUTOINCREMENT)
genre_name (VARCHAR, NOT NULL)
Actor
PK actorID (INTEGER, AUTOINCREMENT)
actor_full_name (VARCHAR, NOT NULL)
MovieGenre
PK genre_listID (INTEGER, AUTOINCREMENT)
FK movieID (INTEGER, NOT NULL)
FK genreID (INTEGER, NOT NULL)
MovieActor
PK actor_listID (INTEGER, AUTOINCREMENT)
FK movieID (INTEGER, NOT NULL)
FK actorID (INTEGER, NOT NULL)
Movie
PK movieID (INTEGER, AUTOINCREMENT)
FK directorID (INTEGER, NOT NULL)
title (VARCHAR, NOT NULL)
release_year (INTEGER, NOT NULL)
description (VARCHAR, NOT NULL)
runtime_minutes (INTEGER, NOT NULL)
User
PK userID (INTEGER, AUTOINCREMENT)
user_name (VARCHAR, UNIQUE, NOT NULL)
password (VARCHAR, NOT NULL)
time_spent_watching_movies_minutes (INTEGER, DEFAULT = 0)
UserReview
PK user_reviewID (INTEGER, AUTOINCREMENT)
FK reviewID (INTEGER, NOT NULL)
FK userID (INTEGER, NOT NULL)
Review
PK reviewID (INTEGER, AUTOINCREMENT)
FK movieID (INTEGER, NOT NULL)
review_text (VARCHAR, NOT NULL)
rating (INTEGER, NOT NULL)
timestamp (DATETIME, NOT NULL)
latest_edit (DATETIME)

Results from final testing - Memory

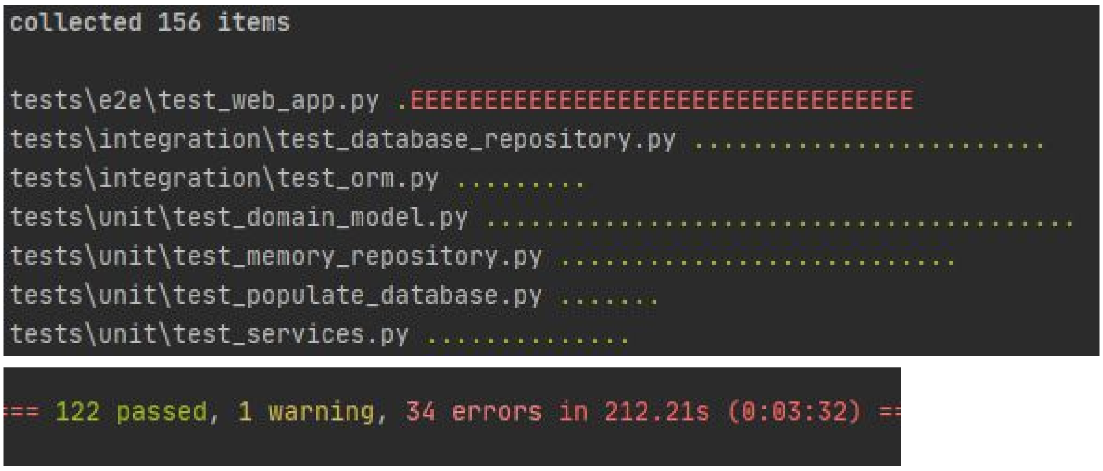


```
collected 156 items

tests\e2e\test_web_app.py ...................................
tests\integration\test_database_repository.py ........................
tests\integration\test_orm.py .........
tests\unit\test_domain_model.py ........................................
tests\unit\test_memory_repository.py ...........................
tests\unit\test_populate_database.py .......
tests\unit\test_services.py ..............
```

```
== 156 passed, 1 warning in 225.73s (0:03:45) ==
```

Results from final testing - Database



```
collected 156 items

tests\e2e\test_web_app.py .EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
tests\integration\test_database_repository.py ........................
tests\integration\test_orm.py .........
tests\unit\test_domain_model.py ........................................
tests\unit\test_memory_repository.py ...........................
tests\unit\test_populate_database.py .......
tests\unit\test_services.py ..............
```

```
=== 122 passed, 1 warning, 34 errors in 212.21s (0:03:32) ==
```

Further Notes:

- Searching by Director, Actor(s) and/or Genre(s) using 'Search movies' will be slower due to how it is implemented.
- When attempting the tests within tests\e2e\test_web_app, if: `'REPOSITORY': 'database'` and `'TEST_DATA_PATH': TEST_DATA_PATH_DATABASE` in conftest's client function, then the following error message occurs for all but it's first test:

`
sqlalchemy.exc.ArgumentError: Class '<class 'movie_web_app.domain.model.Director'>' already has a primary mapper defined. Use non_primary=True to create a non primary Mapper. clear_mappers() will remove *all* current mappers from all classes.
`

    - It essentially means that since the there is already a primary mapper created (in movie_web_app/a_adapters/orm.py) by the first test's client, then we can't make duplicate primary mappers.
    - All these tests still pass when the web app is set to memory.
    - When testing with the database, each test was individually tested, and does pass. The comment #DB-PASS is put next to tests that individually passed.
    - This issue also occurs in the COVID web app's test_web_app tests too.
- Even though the relationship between users and reviews is more naturally one-to-many (since a review is created by one user, and one user can make many reviews), I decided to add a user-review table to handle the relation separately, so that the user foreign key doesn't appear within the 'review' table.
    - This is more faithful to the Review class in the model code, as it also lacks a variable/data structure relevant to the user.
    - It also opens up the opportunity to further expand this web app to allow users to collaborate with reviews (eg users can have their copy of the review they collaborated in with other users).

- In UserReview, reviewID is the first foreign key, since it is usually more unique than userID.
- Testing takes roughly 3 minutes and 45 seconds.

Additional Changes

- Fixed issue from A2 where pressing 'List all movies' points to a url with no page number (from A2 testing, this functioned the same as having /list/ in the url).