tp_2_mongodb_with_python

In this TP we'll have to write python code to implement some features to our Netflix application. All the code you'll have to write will be into the file `mflix/db.py` of the project **Mflix**. Some functions will be to write from scratch and some are to complete. The features to will answer specific use cases that are user stories.

Let's start!

1- Projection:

#fromscratch

User Story

"As a user, I'd like to be able to search movies by country and see a list of movie titles. I should be able to specify a comma-separated list of countries to search multiple countries."

Task

Implement the **get_movies_by_country** method in db.py to search movies by country and use projection to return the title and `_id` field. The `_id` field will be returned by default.

You can find examples in notebooks/your_first_read.ipynb.

MFlix Functionality

Once you complete this ticket, the UI will allow movie searches by one or more countries.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m projection
```

After passing the relevant unit tests, what is the validation code for **Projection**?

2- Text and Subfield Search:

#tocomplete

User Story

"As a user, I'd like to be able to search movies by cast members, genre, or perform a text search of the plot summary, full plot, and title."

Task

For this ticket, you will need to modify the method **build_query_sort_project** in db.py to allow the following movie search criteria:

- genres: finds movies that include any of the wanted genres.
  Already, the **build_query_sort_project** method is able to return results for two different types of movie search criteria:
- text: performs a text search in the movies collection
- cast: finds movies that include any of the wanted cast
  You just need to construct the query that queries the movies collection by the genres field.
  Hint

> Check the implementation of similar formats of search criteria - the genres query should be similar.

MFlix Functionality

Once you complete this ticket, the UI will allow movie searches by members of the cast, movie genres, movie title, and plot summary.

A text index was created for you when you restored the collections with **mongorestore**, so these queries will be performant once they are implemented.

Testing

You can run the unit tests for this ticket with:

```
pytest -m text_and_subfield_search
```

After passing the relevant tests, what is the validation code for **Text and Subfield Search**?

3- Paging:

#tocomplete

User Story

"As a user, I'd like to get the next page of results for my query by scrolling down in the main window of the application."

Task

Modify the method **get_movies** in db.py to allow for paging. You can see how the page is parsed and sent in the **api_search_movies** method from movies.py.

You can find examples of using cursor methods in notebooks/cursor_methods_agg_equivalents.ipynb.

MFlix Functionality

The UI is already asking for infinite scroll! You may have noticed a message stating "paging not implemented" when scrolling to the bottom of the page.

Once this ticket is completed, this message will go away, and scrolling to the bottom of the page will result in a new page of movies.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m paging
```

After passing the relevant tests, what is the validation code for **Paging**?

4- User Management:

#fromscratch

User Story

"As a user, I should be able to register for an account, log in, and logout."

Task

For this Ticket, you'll be required to implement all the methods in db.py that are called by the API endpoints in user.py. Specifically, you'll implement:

- **get_user**
- **add_user**
- **login_user**
- **logout_user**
- **get_user_session**
- **delete_user**

For this ticket, you will need to use the find_one(), update_one() and delete_one() methods. You can find examples in the following notebooks:

- notebooks/deletes.ipynb
- notebooks/your_first_write.ipynb

MFlix Functionality

Once this ticket is completed, users will be able to register for a new account, log in, logout, and delete their account.

Registering should create an account and log the user in, ensuring an entry is made in the **sessions** collection. There is a unique index on the user_id field in **sessions**, so we can efficiently query on this field.

Testing

```
pytest -m user_management
```

After passing the relevant tests, what is the validation code for **User Management**?

5-Durable Writes:

#tocomplete

Task

For this ticket, you'll be required to increase the durability of the **add_user** method from the default write concern of w: 1.

When a new user registers for MFlix, their information must be added to the database before they can do anything else on the site. For this reason, we want to make sure that the data written by the **add_user** method will not be rolled back.

We can completely eliminate the chances of a rollback by increasing the write durability of the **add_user** method. To use a non-default write concern with a database operation, use Pymongo's with_options flag when issuing the query.

You can find examples of write concerns in notebooks/write_concerns.ipynb.

Which of the following write concerns are more durable than the default?

- `w:1`
- `w:majority`
- `w:0`

6- User preferences:

#fromscratch

User Story

"As a user, I want to be able to store preferences such as my favorite cast member and preferred language."

Task

For this Ticket, you'll be required to implement one method in db.py, **update_prefs**. This method allows updates to be made to the "preferences" field in the users collection.

MFlix Functionality

Once this ticket is completed, users will be able to save preferences in their account information.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m user_preferences
```

After passing the relevant tests, what is the validation code for **User Preferences**?

7- Get comments:

#tocomplete

User Story

"As a user, I want to be able to view comments for a movie when I look at the movie detail page."

Task

For this ticket, you'll be required to extend the **get_movie** method in db.py so that it also fetches the comments for a given movie.

The comments should be returned in order from most recent to least recent using the date key.

Movie comments are stored in the comments collection, so this task can be accomplished by performing a $lookup. Refer to the Aggregation Quick Reference for the specific syntax.

You can find examples of Aggregation with the Python driver in notebooks/basic_aggregation.ipynb.

MFlix Functionality

Once this ticket is completed, each movie's comments will be displayed on that movie's detail page.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m get_comments
```

After passing the relevant tests, what is the validation code for **Get Comments**?

Hint: We need to sort the comments in the $lookup stage

---

8 - Create/Update Comments:

#fromscratch

User Story

"As a user, I want to be able to post comments to a movie page as well as edit my own comments."

Task

For this ticket, you'll be required to implement two methods in db.py, **add_comment** and **update_comment**.

Ensure that **update_comment** only allows users to update their own comments, and no one else's comments.

MFlix Functionality

Once this ticket is completed, users will be able to post comments on their favorite (and least favorite) movies, and edit comments they've posted.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m create_update_comments
```

After passing the relevant unit tests, what is the validation code for **Create/Update Comments**?

9- Delete Comments:

#fromscratch

User Story

"As a user, I want to be able to delete my own comments."

Task

For this ticket, you'll be required to modify one method in db.py, **delete_comment**. Ensure the delete operation is limited so only the user can delete their own comments, but not anyone else's comments.

You can find examples of delete_one() in notebooks/deletes.ipynb.

MFlix Functionality

Once this ticket is completed, users will be able to delete their own comments, but they won't be able to delete anyone else's comments.

Testing

You can run all the tests for this ticket by running:

```
pytest -m delete_comments
```

After passing the relevant tests, what is the validation code for **Delete Comments**?

---

10 User Report:

#tocomplete

User Story

"As an administrator, I want to be able to view the top 20 users by their number of comments."

Task

For this ticket, you'll be required to modify one method in db.py, **most_active_commenters**. This method produces a report of the 20 most frequent commenters on the MFlix site.

Hint

> This report is meant to be run from the backend by a manager that is very particular about the accuracy of data. Ensure that the read concern used in this read, avoids any potential document rollback.

Remember to add the necessary changes in the pipeline to meet the requirements. More information can be found in the comments of the method.

You can find examples of Aggregation with the Python driver in notebooks/basic_aggregation.ipynb.

MFlix Functionality

Once this ticket is completed, users with database access can make users administrators. Administrators will be able to generate a user report listing top commenters.

Testing

You can run the unit tests for this ticket by running:

```
pytest -m user_report
```

After passing the relevant tests, what is the validation code for **User Report**?

Homework:

User Story

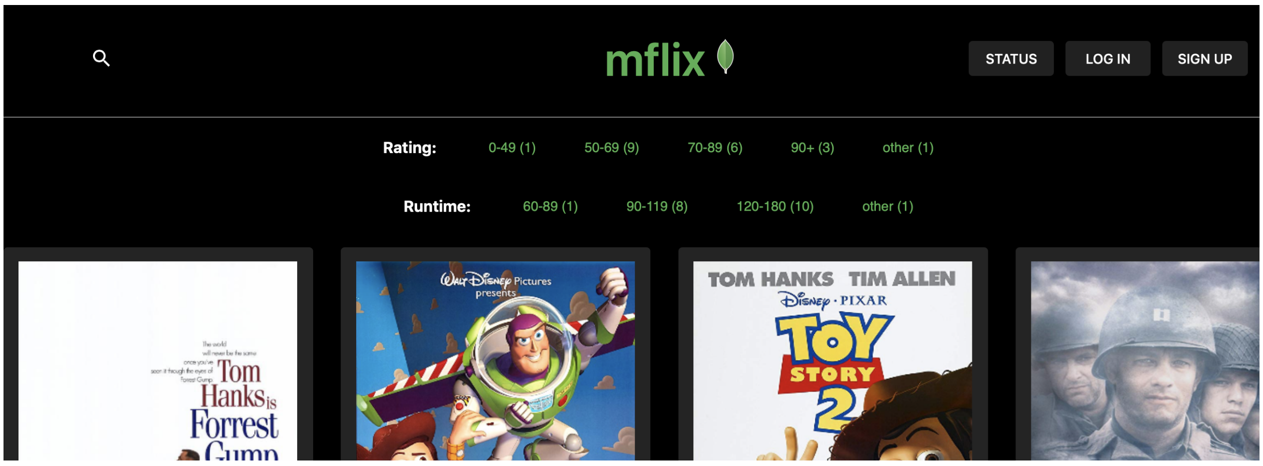
"As a user, I want to be able to filter cast search results by one facet, **metacritic** rating."

Task

For this Ticket, you'll be required to implement one method in db.py, **get_movies_faceted**, so the MFlix application can perform faceted searches.

MFlix Functionality

Once the change is implemented for this ticket, the user interface will reflect this change when you search for cast (e.g. "Tom Hanks"), then additional search parameters will be added as shown below:



What is a Faceted Search?

Faceted search is a way of narrowing down search results as search parameters are added. For example, let's say MFlix allows users to filter movies by a rating from 1 to 10, but Kate Winslet has only acted in movies that have a rating of 6 or higher.

If we didn't specify any other search parameters, MFlix would allow us to choose a rating between 1 and 10. But if we first search for Kate Winslet, the application would only let us choose a rating between 6 and 10, because none of the movie documents in the result set have a rating below 6.

If you're curious, you can read more about Faceted Search here.

Faceted Search in MFlix

Faceted searches on the MFlix site cannot be supported with the basic search method **get_movies**, because that method uses the Mongo query language. For faceted searches, the application must use the Aggregation Framework.

The method **get_movies_faceted** uses the Aggregation Framework, and the individual *stages* in the pipeline have already been completed. Follow instructions in the db.py file to append the required stages to the pipeline object.

Testing and Running the Application

Look in the test_facets.py file in your **tests** directory to view the unit tests for this ticket.

You can run the unit tests for this ticket by running:

```
pytest -m facets
```

Once the unit tests are passing, run the application with:

```
python run.py
```

Now proceed to the status page to run the full suite of integration tests and get your validation code.

After passing the relevant tests, what is the validation code for **Faceted Search**?