Assignment 2 - SQLAlchemy

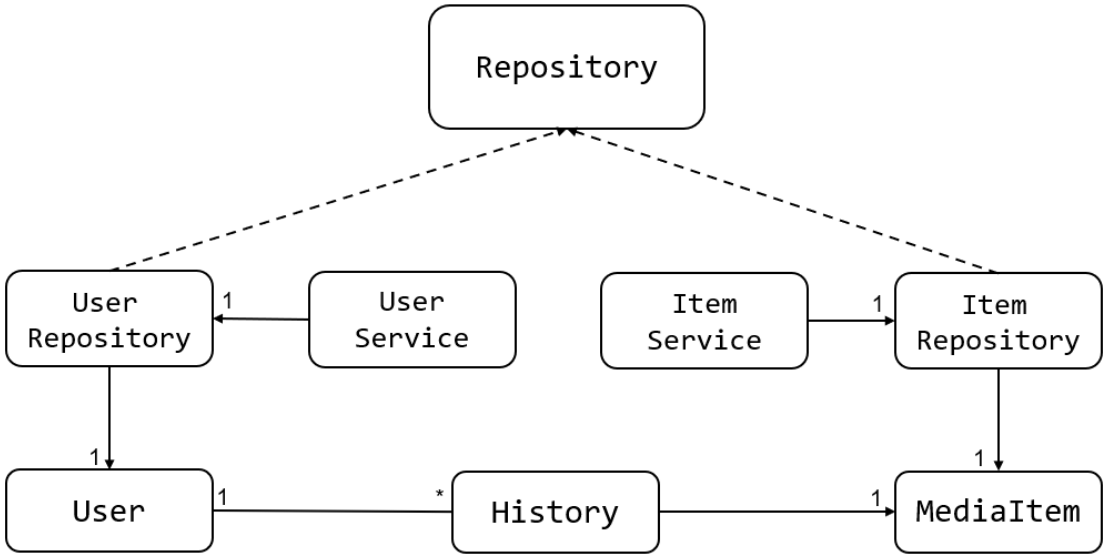
Submission instructions:

- The assignment should be submitted in pairs.
- **In this assignment, All DB functions should use SQLAlchemy.**

The assignment should be submitted in pairs. You should submit one zip file. The archived zip file should contain 1 file:

- A Python file named hw2.py contains all the functions and comments- use it.
- No need to submit the main function but using it for the internal tests is recommended.

In this assignment, you are required to build a system as described in the following diagram:



The entry point to the system is through the services, and the user acts as the aggregate root. When the user is loaded, their associated histories, along with the media related to them, will also be loaded.

- At the end of the attached Python file, there is a comment explaining how to open a session - this can be used for in-depth testing. Please delete/leave it in a comment before submitting.
- The session is accepted open in services - no need to open and close in functions, just commit after a write operation

The archive file name should be of form id1_id2.zip.

Deadline: 19/12/2024

Assignment Instructions:

1. Entities

a. Create User class with the following columns:

table name Users

- id (String(255)) – primary key, defined as the username
- password (LargeBinary)- the password is obtained from a string but must be securely stored in an encrypted format using the bcrypt library
- first_name (String)
- last_name (String)
- date_of_birth (DateTime)
- registration_date (DateTime)
- histories (History list)

Write a Python function add_history in User class:

def add_history(self, media_item_id) -> None

- The function adds to the histories a new object of type History that contains the current user, the ID of the received media item, and the current time

Write a Python function sum_title_length in User class:

def sum_title_length (self) -> int

- The function takes no arguments and returns the total length of all MediaItem titles associated with the user's histories

b. Create MediaItem class with the following columns:

table name MediaItems

- id (Integer) – primary key and autoincrement
- title (String)
- prod_year (Integer)
- title_length (Integer)

c. Create History class with the following columns:

table name History

- id (Integer) – primary key and autoincrement
- user_id (Sring) – ForeignKey
- user – relationship with User
- media_item_id (Integer)- ForeignKey
- mediaitem – relationship with MediaItem
- viewtime (DateTime)

2. Repository

Each class inherits from a Repository class.

In UserRepository class:

Write a Python function validateUser:

```
def validateUser (self, session, username: str, password: str) -> bool
```

- The function compares received values with existing ones in the database.
- The function returns True if the password is matched (using bcrypt) to the password of the user stored in the table with the corresponding username, otherwise returns false.
- If no user exists with the provided username, return false.

Write a Python function getNumberOfRegistredUsers:

```
def getNumberOfRegistredUsers(self, session, n: int) -> int
```

- The function receives an integer number n
- The function retrieves from the table Users the number of registered users in the past n days
- The function returns an integer number

In ItemRepository class:

Write a Python function getTopNItems :

```
def getTopNItems (self, session, top_n: int) -> list
```

- The function retrieves from the table MediaItems first top_n items (id ascending order).

3. **Services**

These classes are the entry points to the system. Each function in a service class corresponds to a use-case scenario. Note that each use case (function) should involve a single **transaction**.

In UserService class:

Write a Python function create_user :

```
def create_user(self,username,password,first_name, last_name, date_of_birth)->None
```

▪ The function receives all the details required to create a user, creates it with the current time for registration_date, and adds it to the table using the UserRepository

Write a Python function add_history_to_user:

```
def add_history_to_user (self, username, media_item_id)->None
```

▪ The function adds an object of type history with the id of the media item to the user whose username was received. If there is no user with this username- a ValueError("User not found") error will be thrown

Write a Python function validateUser:

```
def validateUser (self, username: str, password: str) -> bool
```

▪ The function calls the corresponding method from the UserRepository.

Write a Python function getNumberOfRegistredUsers:

```
def getNumberOfRegistredUsers(self, n: int) -> int
```

▪ The function calls the corresponding method from the UserRepository.

Write a Python function sum_title_length_to_user:

```
def sum_title_length_to_user (self, username) -> int
```

▪ A function receives a username as input and returns the total length of all MediaItem titles associated with the history of the user corresponding to the provided username. If there is no user with this username- a ValueError("User not found") error will be thrown

Write a Python function get_all_users_to_user:

```
def get_all_users (self)
```

▪ The function returns all existing users in the system (UserRepository must be used)

In ItemService class:

Write a Python function create_item:

```
def create_item(self,title, prod_year)->None
```

▪ The function receives a title and prod_year, creates a MediaItem object with the received title and prod_year along with the length of the received title, and adds it to the table using the ItemRepository

Good Luck,
Moria Cohen.