Pràctica 3 - UF2 M06: L’app de pel·lícules

Per dur a terme aquesta activitat necessitem:

* Escriptori isard ubuntu amb MariaDB i Visual Studio Code instal·lats
* Disposar de la base de dades de pel·lícules que ja has emprat a la pràctica 1
* Instal·lar postgreSQL a l’escriptori o bé fer ús d’un contenidor docker amb postgreSQL.

# Objectiu

A la sessió anterior vam explicar els patrons DAO i factoria abstracta, de tal manera que vam veure com es poden separar els mètodes de lògica d’aplicació (amb interfícies) del que és la seva implementació (segons si és MariaDB, Mongo, Redis, etc.).

En el següent repositori de github teniu una aplicació de consola incompleta, la qual interacciona amb la base de dades de pel·lícules. Els objectius de la pràctica seran:

* Implementar els mètodes de cerca per un camp, inserir una nova pel·lícula i modificar una existent.
* Afegir un nou mètode de selecció que permeti “paginar” els resultats. Es tracta d’una variant del mètode existent de llegir totes en la qual es mostren els resultats de 10 en 10.
* Crear una classe de persistència per postgreSQL amb implementació similar a la que teniu per mySQL/mariaDB. L’aplicació haurà de ser capaç d’interaccionar amb una o una altra base de dades.

Es valorarà i tindrà en compte com és habitual:

* Codi net i pulit, amb comentaris que expliquin que fan determinades instruccions.
* Fer ús de mètodes i funcions sempre que sigui possible i utilitzar-les adequadament.
* Refactoritzar adequadament el codi.

# Enunciat i passos

Disposeu del motlle al repositori de github següent:

<https://github.com/ToniTalens/films_app_practica/tree/main>

Com podeu veure podeu fer el git clone i se us crearà un fork del projecte, el qual podreu fer vostre i desplegar els canvis als vostres repositoris.

NOTA IMPORTANT: El fitxer de configuracio.yml no té usuari ni password per raons òbvies (encara que tampoc estem protegint cap informació que sigui vital). L’usuari i el password són els ja coneguts.

**Implementació de mètodes a la classe persistencia\_pelicula\_mysql.py**

Els mètodes **desa** i **canvia** estan buits. Respectivament insereixen i modifiquen pel·lícules. Tenen com a retorn un objecte pel·lícula, que fonamentalment és l’objecte que s’ha inserit o modificat segons sigui el cas.

Per la seva banda el mètode **llegeix** rep com a paràmetre un any i retorna les pel·lícules d’aquell any.

Per provar el seu funcionament, des d’aquesta mateixa classe podeu fer servir un main i veure que fan el que espereu que facin. Un cop funcionin els utilitzarem més endavant.

**Paginació a partir del mètode totes**

**Primera part: canvis a la classe persistencia\_pelicula\_mysql.py**

A la classe de persistència tenim el mètode totes, el qual fa una SELECT incondicional i retorna tots els registres de la taula. La forma amb la qual es crida és mitjançant el mètode llegeix\_de\_disc que trobareu a la classe llistapelis. Com de fet haureu comprovat, per comoditat emprem una classe llistapelis que funciona com un contenidor de llistes de pel·lícules, però compte, no és l’únic atribut que té!

Així, l’objectiu serà omplir el mètode totes\_pag per tal que mostri les pel·lícules de 10 en 10. Per fer-ho, cal que la SELECT a la taula sols mostri 10 resultats cada cop que es crida el mètode, seguint un ordre per fer les SELECT i mostrar els resultats successius. Considereu el paper que juga la id de la pel·lícula, ja que us serà útil per poder reprendre la cerca i mostrar le següents pel·lícules.

Per entendre-ho millor, quan s’arrenca l’aplicació i es tria l’opció 1, ens mostrarà els 10 primers registres començant per la id més baixa. A l’acabar, ens preguntarà si volem sortir (opció 0) o seguim pels 10 següents:

| Primera crida |
| --- |
| {  "pelicules": [  {  "id": 1,  "titol": "Star Wars",  "any": 1977,  "puntuacio": 8.9,  "vots": 14182  },  {  "id": 2,  "titol": "Pulp Fiction",  "any": 1994,  "puntuacio": 8.4,  "vots": 11693  },  {  "id": 3,  "titol": "Blade Runner",  "any": 1982,  "puntuacio": 8.6,  "vots": 8665  },  {  "id": 4,  "titol": "Titanic",  "any": 1997,  "puntuacio": 9.2,  "vots": 8129  },  {  "id": 5,  "titol": "Braveheart",  "any": 1995,  "puntuacio": 8.4,  "vots": 8074  },  {  "id": 6,  "titol": "Empire Strikes Back, The",  "any": 1980,  "puntuacio": 8.5,  "vots": 8050  },  {  "id": 7,  "titol": "Shawshank Redemption, The",  "any": 1994,  "puntuacio": 8.8,  "vots": 7850  },  {  "id": 8,  "titol": "Independence Day",  "any": 1996,  "puntuacio": 7.0,  "vots": 7138  },  {  "id": 9,  "titol": "Usual Suspects, The",  "any": 1995,  "puntuacio": 8.7,  "vots": 6981  },  {  "id": 10,  "titol": "Raiders of the Lost Ark",  "any": 1981,  "puntuacio": 8.4,  "vots": 6488  }  ]  }  0.- Surt de l'aplicació.  2.- Mostra les següents 10 pel·lícules  Selecciona una opció: |

Així, si triem l’opció 2, aquest hauria de ser el resultat. Fixeu-vos que segueix per la id amb valor 11:

| Segona crida |
| --- |
| {  "pelicules": [  {  "id": 11,  "titol": "2001: A Space Odyssey",  "any": 1968,  "puntuacio": 8.4,  "vots": 6413  },  {  "id": 12,  "titol": "Forrest Gump",  "any": 1994,  "puntuacio": 7.8,  "vots": 6269  },  {  "id": 13,  "titol": "Aliens",  "any": 1986,  "puntuacio": 8.3,  "vots": 5811  },  {  "id": 14,  "titol": "Silence of the Lambs, The",  "any": 1991,  "puntuacio": 8.3,  "vots": 5715  },  {  "id": 15,  "titol": "Princess Bride, The",  "any": 1987,  "puntuacio": 8.4,  "vots": 5522  },  {  "id": 16,  "titol": "Terminator 2: Judgment Day",  "any": 1991,  "puntuacio": 8.0,  "vots": 5513  },  {  "id": 17,  "titol": "Casablanca",  "any": 1942,  "puntuacio": 8.7,  "vots": 5489  },  {  "id": 18,  "titol": "Monty Python and the Holy Grail",  "any": 1974,  "puntuacio": 8.4,  "vots": 5319  },  {  "id": 19,  "titol": "Star Trek: First Contact",  "any": 1996,  "puntuacio": 8.2,  "vots": 5298  },  {  "id": 20,  "titol": "Fargo",  "any": 1996,  "puntuacio": 8.2,  "vots": 5293  }  ]  }  0.- Surt de l'aplicació.  2.- Mostra les següents 10 pel·lícules  Selecciona una opció: |

**Segona part: canvis a les altres classes i fitxers**

* Caldrà que modifiqueu la classe llistapelicula i feu que el mètode llegeix\_de\_disc cridi al mètode totes\_pag.
* Caldrà que feu alguns canvis o directament fiqueu codi a l’app\_film.py, concretament:
  + Canvis a bucle\_principal, per tal que permeti la paginació. Actualment sols fa la lectura per totes.
  + A partir dels mètodes que ja hi ha implementats, al mètode database\_read té mancances de codi i caldrà que les solventeu.
  + Implementeu el que sigui necessari dins el mostrar\_menu i altres parts del codi per tal que l’usuari pugui inserir una nova pel·lícula, modificar una existent o seleccionar algunes per any. Caldrà doncs aprofitar i/o cridar el que heu implementat a la primera part de la pràctica.

Responeu a aquestes preguntes:

1. Què fan els mètodes **get\_configuracio** i **get\_persistencies**?

Els mètodes get\_configuracio i get\_persistencies obtenen la configuració de l'aplicació des d'un fitxer de configuració i recuperen les credencials necessàries per connectar-se a la base de dades, respectivament.

1. A **procesa\_opcio** veureu instruccions com aquestes:

return {

"0": lambda ctx : mostra\_lent("Fins la propera"),

"1": lambda ctx : mostra\_llista(ctx['llistapelis'])

}

Què fa lambda? Com es podria reescriure el codi sense utilitzar lambda? Quina utilitat hi trobeu a utilitzar lambda?

En aquest context, lambda és una funció anònima que es pot passar com a argument a altres funcions. Es podria reescriure el codi utilitzant funcions definides per separat en lloc de funcions lambda. L'ús de lambda és útil per a funcions senzilles i pot fer que el codi sigui més concís.

1. Penseu que s’ha desacoblat suficientment la lògica de negoci de la lògica d’aplicació? Raoneu la resposta i digueu si hi ha cap millora que es pugui fer.

La lògica de negoci s'ha desacoblat prou de la lògica de l'aplicació. Els mètodes tenen funcions específiques i clares, el que facilita la comprensió i el manteniment del codi. Una millora podria ser utilitzar noms de funcions més descriptius en lloc de funcions lambda per millorar la llegibilitat del codi. També es podria considerar l'ús de disseny de patrons com ara el patró Estratègia per millorar la flexibilitat i la reutilització del codi.

**Implementació amb un altre SGDB: postgreSQL**

A imatge i semblança de la classe **persistencia\_pelicula\_mysql.py** cal que feu la classe **persistencia\_pelicula\_pgSQL.py**, la qual tindrà els mateixos mètodes però adaptats per postgreSQL.

Com ho haureu de provar caldrà que feu el següent:

* Instal·lar postgreSQL dins aquest escriptori isard i la corresponent extensió a VisualStudioCode. Una alternativa és utilitzar el contenidor docker que ja disposeu per altres mòduls com M10.
* Tant si instal·leu localment (a l’escriptori isard) com si feu ús de docker, se us donarà un script de pel·lícules que carregarà la BD postgreSQL.
* Caldrà que us genereu un fitxer de credencials per tal de poder accedir a la base de dades, semblant al que ja hi ha per mySQL.

# Avaluació

Se us indicarà quina és la rúbrica d’avaluació al moodle.

Els lliuraments seran parcials, és a dir, hi haurà una data de lliurament definitiva, però caldrà complir fites i fer els commits als vostres repositoris per anar lliurant les diferents parts, en les dates límits que us indiqui el professor.

# Resultats d’aprenentatge i criteris d’avaluació

RA1. Desenvolupa aplicacions que gestionen informació emmagatzemada en bases de dades relacionals identificant i utilitzant mecanismes de connexió.

1.1 Valora els avantatges i inconvenients d’utilitzar connectors.

1.2 Utilitza gestors de bases de dades incrustats i independents.

1.3 Utilitza el connector idoni en l’aplicació.

1.4 Estableix la connexió.

1.5 Defineix l’estructura de la base de dades.

1.6 Desenvolupa aplicacions que modifiquen el contingut de la base de dades.

1.7 Defineix els objectes destinats a emmagatzemar el resultat de les consultes.

1.8 Desenvolupa aplicacions que fan consultes.

1.9 Elimina els objectes un cop finalitzada la seva funció.

1.10 Gestiona les transaccions