

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-231, Chistol Maxim**

**Raport**

**pentru lucrarea individuală**

***la cursul de “Programarea Interactivă”***

Verificat:

**Alexei Victoria,** asistent.universitar.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**Introducere:**

**Python** este un limbaj de programare de nivel înalt, recunoscut pentru simplitatea și claritatea sa. Datorită sintaxei intuitive, Python este folosit în multiple domenii, precum dezvoltarea de aplicații, automatizarea sarcinilor, inteligență artificială și analiza de date. Printre principalele sale avantaje se numără: sintaxa simplă, portabilitatea pe mai multe platforme și o colecție vastă de biblioteci care facilitează dezvoltarea rapidă a proiectelor.

În contextul dezvoltării de **Telegram Bots**, Python oferă o bibliotecă puternică numită **Aiogram 3**. Aiogram este o bibliotecă asincronă modernă, care permite gestionarea eficientă a mesajelor și interacțiunilor cu utilizatorii. Aceasta oferă funcționalități precum gestionarea comenzilor, implementarea tastaturilor inline și personalizate, callback-uri și middleware pentru adăugarea unor funcționalități suplimentare.

Utilizarea limbajului Python împreună cu Aiogram 3 aduce următoarele avantaje:

1. **Dezvoltare rapidă** datorită simplității codului Python.
2. **Performanță ridicată**, deoarece Aiogram utilizează programarea asincronă pentru a gestiona mai mulți utilizatori simultan.
3. **Flexibilitate**, oferind opțiuni de integrare cu baze de date, API-uri externe și notificări automate.
4. **Documentație detaliată**, care facilitează implementarea rapidă a funcțiilor.
5. **Scalabilitate**, fiind potrivit atât pentru proiecte mici, cât și pentru aplicații complexe.

Astfel, Python și Aiogram reprezintă o alegere ideală pentru dezvoltatorii care doresc să creeze **Telegram Boturi** funcționali, eficienți și ușor de întreținut.

* 1. Teorie:Limbajul Python si bibliotecele folosite în lucrare

**Python** este un limbaj de programare interpretat, dinamic și de nivel înalt, dezvoltat pentru a fi simplu și ușor de utilizat. Datorită sintaxei sale clare și a flexibilității, Python este folosit într-o gamă variată de domenii, precum dezvoltarea aplicațiilor web, automatizarea proceselor, analiză de date și crearea de **bots** pentru diferite platforme.  
Unul dintre principalele avantaje ale Python este ecosistemul său vast de biblioteci, care permit rezolvarea rapidă și eficientă a unor probleme complexe.

În cadrul acestei lucrări, am utilizat următoarele **biblioteci și instrumente**:

### **Aiogram:**

Aiogram este o bibliotecă Python asincronă, special creată pentru dezvoltarea de **Telegram Boturi**. Aceasta oferă o abordare modernă și flexibilă pentru gestionarea interacțiunilor cu utilizatorii.

**Caracteristici principale**:

* Gestionarea mesajelor, comenzilor și callback-urilor.
* Suport pentru **tastaturi inline** și personalizate.
* Utilizarea programării **asincrone** (async/await) pentru performanță optimă.
* Middleware-uri pentru procesarea mesajelor și gestionarea logicii complexe.

Această bibliotecă permite dezvoltarea unor bots eficienți, scalabili și ușor de întreținut, fiind alegerea perfectă pentru proiecte care necesită interacțiuni rapide cu utilizatorii.

### **SQLAlchemy:**

SQLAlchemy este o bibliotecă Python utilizată pentru **maparea obiectelor la baze de date relaționale** (ORM - Object Relational Mapping). Aceasta simplifică interacțiunea cu bazele de date, oferind o metodă intuitivă și programatică de gestionare a datelor.

**Caracteristici principale**:

* Permite definirea **tabelelor** și a relațiilor folosind clase Python.
* Suportă mai multe baze de date, precum SQLite, PostgreSQL și MySQL.
* Oferă o gestionare eficientă a interogărilor și tranzacțiilor.
* Facilitează dezvoltarea prin eliminarea necesității scrierii directe a interogărilor SQL.

### **SQLite Studio:**

SQLite Studio este un instrument dedicat gestionării bazelor de date **SQLite**, o soluție simplă, ușor de integrat și foarte utilizată pentru aplicații mici și medii. SQLite funcționează local, fără necesitatea unui server, fiind potrivit pentru aplicații care necesită stocarea locală a datelor.

**Caracteristici principale ale SQLite Studio**:

* Interfață grafică intuitivă pentru gestionarea bazelor de date.
* Crearea și editarea **tabelelor** și a **datelor** într-un mod simplu și vizual.
* Generarea automată a interogărilor SQL.
* Exportul și importul datelor din diverse formate (CSV, JSON etc.).
  1. Codul:

1. **Run.py:**
2. import aiogram
3. import asyncio
4. from aiogram import Bot, Dispatcher
5. from Config import TOKEN
6. from app.handlers import router
7. from app.database.models import async\_main
9. bot = Bot(token=TOKEN)
10. dp = Dispatcher()
12. async def main():
13. await async\_main()
14. dp.include\_router(router)
15. await dp.start\_polling(bot)
17. if \_\_name\_\_ == '\_\_main\_\_':
18. try:
19. asyncio.run(main())
20. except KeyboardInterrupt:
21. print("Bot Dezactivat")
22. **Handlers:**
23. from aiogram import F, Router
24. from aiogram.filters import CommandStart, Command
25. from aiogram.types import Message, CallbackQuery
26. import app.keyboards as kb
27. import app.database.requests as rq
28. from aiogram.utils.keyboard import InlineKeyboardMarkup,InlineKeyboardButton
29. router = Router()
30. @router.message(CommandStart())
31. async def cmd\_start(message: Message):
32. await rq.set\_user(message.from\_user.id)
33. await message.answer('Salut.Te pot ajuta în căutarea unui film ?!')
34. @router.message(Command('help'))
35. async def help(message: Message):
36. await message.answer('-------FilmaxBot------\nAcest bot ți permite sa deschuți si sa cauti film ori serial pentru vezionare\n--------------------------------------------\nComenzile Precepute de bot sunt: \n---/start----Activiază botul\n---/help---- Instrucțiuni pentru bot și posibilitățile botului\n---/info----informații adiționale\n\n-------Ce precepe botul------\n-----da-----\n-----nu-----\n-----hm-----\n-----recomanda un film-----\n\n\nMultumesc pentru Atenție!!!\n')
37. @router.message(Command('info'))
38. async def info(message: Message):
39. await message.answer('Sunt creația unui proiect menit să vă ușureze căutarea filmelor, serialelor și desenelor animate, aducându-vă mai aproape de divertismentul care vă inspiră.\n Mulțumesc că mă alegeți pentru a vă ghida în această lume a poveștilor.🥺❤️')
40. @router.message()
41. async def handle\_message(message: Message):
42. text = message.text.strip().lower()
44. if text == "da":
45. message\_text = "Mă Bucur, îți doresc vizionare plăcută!!!🥺❤️"
46. reply\_markup = kb.nan
47. elif text == "nu":
48. message\_text = "Îți recomand cu drag să te așezi confortabil❤️, să îți pregătești un bol de popcorn și să te bucuri de o experiență mai plăcută alături de mine🥺❤️"
49. reply\_markup = kb.nan
50. elif text == "hm":
51. message\_text = "Te gândești ce film să vizionezi? "
52. reply\_markup = kb.nan
53. elif text == "salut":
54. message\_text = "Salut, esti gata sa vezionezi un film sau un serial?❤️🎬🍿\niți pot recomanda ceva?!"
55. reply\_markup = kb.nan
56. elif text == "recomanda un film":
57. message\_text = "Sigur iti voi recomda  niste filme din baza mea de date"
58. reply\_markup = None
59. elif text == 'filme❤️🎬🍿seriale':
60. await catalog(message)
61. return
62. else:
63. message\_text = "Nu am înțeles, te rog să spui mai clar."
64. reply\_markup = None
65. await message.answer(message\_text, reply\_markup=reply\_markup)
66. async def catalog(message: Message):
67. await message.answer('Alegeți categoria dorită!',
68. reply\_markup=await kb.categories())
70. @router.callback\_query(F.data.startswith('category\_'))
71. async def handle\_category(callback: CallbackQuery):
72. try:
73. category\_id = callback.data.split('\_')[1]
74. print(f"Category ID: {category\_id}")
75. await callback.answer("Alegeți un Film sau Serial din această categorie!")
76. await callback.message.answer("Selectează un Film:",
77. reply\_markup=await kb.film(category\_id))
78. except IndexError:
79. await callback.answer("Eroare: ID-ul categoriei nu este valid.", show\_alert=True)
80. @router.callback\_query(F.data.startswith('film\_'))
81. async def film(callback: CallbackQuery):
82. try:
83. film\_id = callback.data.split('\_')[1]
84. film\_data = await rq.get\_film(film\_id)
86. if film\_data:
87. await callback.answer("Film sau Serial selectat!")
88. await callback.message.answer(
89. f"Denumirea: {film\_data.name}\n---------------------------------\nActorii: {film\_data.actor}\n\nAnull Producerii: {film\_data.age}\n---------------------------------\nTipul: {film\_data.tip}\n---------------------------------\nDescrierea: {film\_data.description}\n---------------------------------\nAnii Recomandți: {film\_data.age\_rec}\n---------------------------------\nCategoria:{film\_data.category}\n---------------------------------\nLink spre film:{film\_data.link}\n---------------------------------\nYouTube link: {film\_data.linkyotube}")
90. reply\_markup=await kb.film(film\_id)
91. else:
92. await callback.answer("Eroare: Film sau Serial nu a fost găsit în baza de date.", show\_alert=True)
93. except IndexError:
94. await callback.answer("Eroare: ID-ul Filmui sau Serialui nu este valid.", show\_alert=True)
95. except Exception as e:
96. await callback.answer(f"Eroare: {str(e)}", show\_alert=True)
97. 1. **Keyboards:**
98. from aiogram.types import (ReplyKeyboardMarkup, KeyboardButton,
99. InlineKeyboardMarkup,InlineKeyboardButton)
100. from aiogram.utils.keyboard import InlineKeyboardBuilder
101. from app.database.requests import get\_categories,get\_film\_category
102. nan = ReplyKeyboardMarkup(
103. keyboard=[
104. [KeyboardButton(text='Filme❤️🎬🍿Seriale')]
105. ],
106. resize\_keyboard=True,
107. input\_field\_placeholder='Alegeti! Ce doriți să priviți'
108. )
109. async def categories():
110. all\_categories = await get\_categories()
111. keyboard = InlineKeyboardBuilder()
112. for category in all\_categories:
113. keyboard.add(InlineKeyboardButton(text=category.name, callback\_data=f"category\_{category.id}"))
114. keyboard.add(InlineKeyboardButton(text='Main',callback\_data='to\_main'))
115. return keyboard.adjust(3).as\_markup()
116. async def film(category\_id):
117. all\_films = await get\_film\_category(category\_id)
118. keyboard = InlineKeyboardBuilder()
119. for film in all\_films:
120. keyboard.add(InlineKeyboardButton(text=film.name, callback\_data=f"film\_{film.id}"))
121. keyboard.add(InlineKeyboardButton(text='Main',callback\_data='to\_main'))
122. return keyboard.adjust(3).as\_markup()  
     1. **requests:**
123. from app.database.models import async\_session
124. from app.database.models import User,Category,Film
125. from sqlalchemy import select,update,delete
126. from sqlalchemy.orm import joinedload
127. from sqlalchemy.future import select
128. async def set\_user(telegram\_id):
129. async with async\_session() as session:
130. user = await session.scalar(select(User).where(User.telegram\_id == telegram\_id))
131. if not user:
132. session.add(User(telegram\_id=telegram\_id))
133. await session.commit()
134. async def get\_categories():
135. async with async\_session() as session:
136. return await session.scalars(select(Category))
138. async def get\_film\_category(category\_id):
139. async with async\_session() as ssession:
140. return await ssession.scalars(select(Film).where(Film.category == category\_id))
141. async def get\_film(film\_id):
142. async with async\_session() as session:
143. film = await session.scalar(
144. select(Film)
145. .options(joinedload(Film.category\_obj))
146. .where(Film.id == film\_id)
147. )
148. return film

**5. models:**

from sqlalchemy.orm import DeclarativeBase, Mapped, mapped\_column

from sqlalchemy.ext.asyncio import AsyncAttrs, async\_sessionmaker, create\_async\_engine

import asyncio

from sqlalchemy.orm import relationship

from sqlalchemy import Column, String, Integer, ForeignKey,BigInteger

engine = create\_async\_engine(url='sqlite+aiosqlite:///Bazadedate.sqlite3')

async\_session = async\_sessionmaker(engine)

class Base(AsyncAttrs, DeclarativeBase):

    pass

class User(Base):

    \_\_tablename\_\_ = 'users'

    id: Mapped[int] = mapped\_column(primary\_key=True)

    telegram\_id: Mapped[int] = mapped\_column(BigInteger)

class Category(Base):

    \_\_tablename\_\_ = 'categories'

    id = Column(Integer, primary\_key=True)

    name = Column(String(32))

    films = relationship("Film", back\_populates="category\_obj")

    def \_\_repr\_\_(self):

        return f"<Category(id={self.id}, name='{self.name}')>"

class Film(Base):

    \_\_tablename\_\_ = 'films'

    id = Column(Integer, primary\_key=True)

    name = Column(String(32))

    category = Column(Integer, ForeignKey('categories.id'))

    tip = Column(String(32))

    age = Column(String(16))

    age\_rec = Column(String(4))

    actor = Column(String(256))

    description = Column(String(1048))

    link = Column(String(1000))

    linkyotube = Column(String(1000))

    category\_obj = relationship("Category", back\_populates="films")

    def \_\_repr\_\_(self):

        return f"<Film(id={self.id}, name='{self.name}',actori='{self.actor}',anu='{self.age}',tip='{self.tip}',ani recomandati='{self.age\_rec}', description='{self.description}',link='{self.link}',link\_yotube='{self.linkyotube}')>"

async def async\_main():

    async with engine.begin() as conn:

        await conn.run\_sync(Base.metadata.create\_all)

    print("Tabelele au fost create cu succes!")

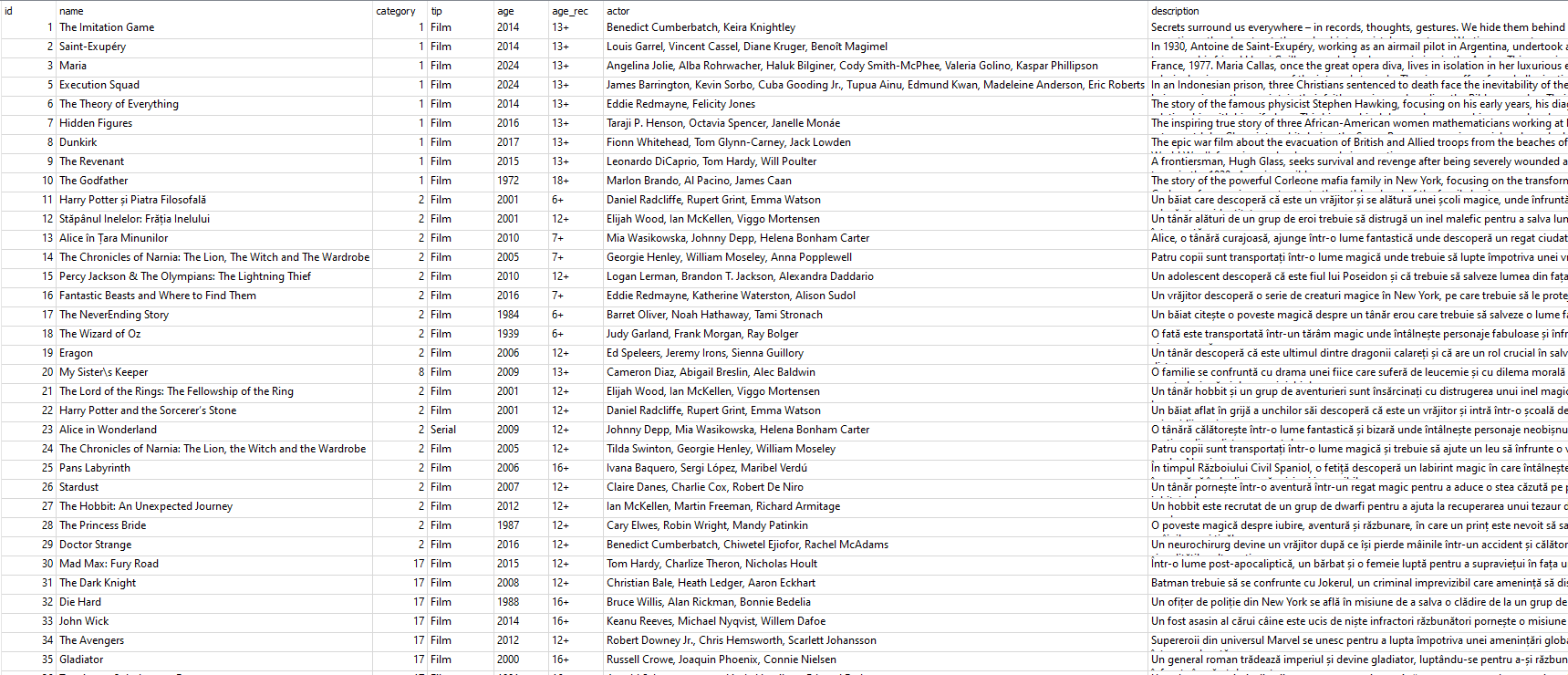
if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(async\_main())

**6. Tabela Category, din baza de date inserată cu date:**



**Figura 1: Tabela category.**

**7. Tabela films:Inserată cu date care le foloseste botul:**

**Figura 2: Tabela films.**

# **Concluzii:**

În această lucrare, am demonstrat eficiența limbajului **Python** și a bibliotecilor moderne în dezvoltarea unui **Telegram Bot**. Python, datorită simplității și flexibilității sale, a permis implementarea rapidă și clară a funcționalităților necesare.

Biblioteca **Aiogram** a fost utilizată pentru gestionarea mesajelor, comenzilor și interacțiunilor asincrone, asigurând o performanță optimă prin procesarea simultană a cererilor utilizatorilor. Pentru gestionarea datelor, am folosit **SQLAlchemy**, care simplifică interacțiunea cu baza de date **SQLite,** oferind o structură organizată și ușor de întreținut. Instrumentul **SQLite Studio** a completat procesul, facilitând vizualizarea și gestionarea datelor într-un mod intuitiv.

Prin utilizarea acestor tehnologii, am obținut o aplicație funcțională, eficientă și scalabilă. Soluția prezentată demonstrează avantajele Python și a bibliotecilor sale în dezvoltarea rapidă de aplicații moderne, capabile să răspundă nevoilor utilizatorilor și să gestioneze datele într-un mod structurat și performant.

**Bibliografie:**

1. **freeCodeCamp**. How to Create a Telegram Bot using Python. Disponibil la: <https://www.freecodecamp.org/news/how-to-create-a-telegram-bot-using-python/>. Accesat la: 10 decembrie 2024.
2. **aiogram Documentation**. aiogram 3.15.0 documentation. Disponibil la: <https://docs.aiogram.dev/>. Accesat la: 10 decembrie 2024.
3. **Toptal**. How to Create a Telegram Bot. Disponibil la: <https://www.toptal.com/python/telegram-bot-tutorial-python>. Accesat la: 10 decembrie 2024.
4. **YouTube**. How To Create A Telegram Bot In Python For Beginners. Disponibil la: <https://www.youtube.com/watch?v=vZtm1wuA2yc>. Accesat la: 10 decembrie 2024.
5. **Medium**. Building a Telegram Bot with Python: A Step-by-Step Guide. Disponibil la: <https://medium.com/@rubyabdullah14/building-a-telegram-bot-with-python-a-step-by-step-guide-5ca305bea6c0>. Accesat la: 10 decembrie 2024.