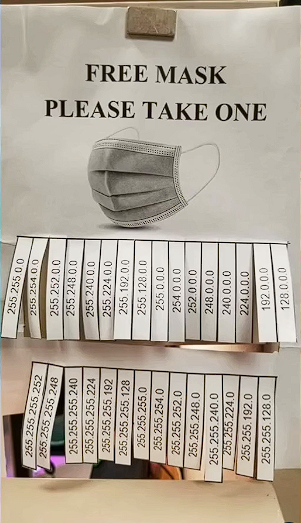
T.P. Integrador

Redes de Datos IA3.5

T.U.I.A. UNR FCEIA



Borgo Elgart, Iair.

Introducción y descripción de la base de datos,

Para este trabajo se creó una API utilizando el módulo FastAPI de Python con la versión 3.11.9 en una PC con Windows 10. También es necesario Pandas para la manipulación del dataset. Otros requerimientos a instalar se encuentran en ´requirements.txt´. Se tomó la fuente de datos[[1]](#footnote-1) de los 100 mejores libros de ficción de la historia.

En ella se encuentran las siguientes columnas:

* Author: El escritor del libro (puede ser desconocido en ciertos casos). Str
* Country: El país de origen. Str
* imageLink: Es la ruta relativa donde se encuentran guardada la imagen correspondiente al libro en cuestión. Str
* Language: Lenguaje en el que fue escrito. Str
* Link: Link a la Wikipedia en inglés, siendo necesario aclarar que todas terminan con un carácter /n al final, luego eliminado en la limpieza del dataset. Str
* Pages: Cantidad de páginas del libro. Int
* Title: Nombre de la obra. Str
* Year: Año publicada (puede ser un estimado). Int

DESCRIPCIÓN DEL ENTORNO DE TRABAJO DEL SERVIDOR

El servidor se encuentra en el archivo ‘main.py’ tiene las siguientes funcionalidades:

* /diccionario-[libros | autores] : Retorna lista ordenada de los nombres de los libros o autores.
* /busqueda-[libro | autor]/{nombre} : Dado un nombre presente en el dataset, realiza una búsqueda y devuelve toda la información relacionada a este. Acepta que se escriba tanto en minúscula como mayúscula, y que sea separado por %20% (espacio en la url) o guion bajo ( \_ )
* /búsqueda-imagen/{nombre}: Dado el nombre de algún libro, retorna la imagen de su tapa.
* /actualizar-link/{nombre}: Dado el nombre de algún libro, y el final el subdirectorio de Wikipedia, actualiza el link en el dataset. Argumento opcional : Idioma de la wiki utilizada, default ‘es’.
* /nuevo-libro: Dado los varios atributos que poseen los libros, crea una nueva entrada en la base de datos.
* /eliminar-libro: Dado el título de un libro, lo elimina la base de datos

Todas estas poseen códigos de respuestas apropiados.

Además de estas funciones, el servidor cuenta con autenticación, por lo que para poder usarlo primero se debe crear un usuario con /nuevo-usuario e ingresar para tener un token que permita usar la API durante 60 minutos. Todos los usuarios creados cuentan con la posibilidad de utilizar cualquiera de las funciones antes mencionadas ya que todos tienen la máxima autorización. Las contraseñas de los usuarios son encriptadas con el algoritmo bcrypt y guardadas junto a su nombre de usuario en ´usuarios.json´.

Crear el servidor fue bastante simple, la parte que más se complico fue al querer implementar autenticación, pero con una simple búsqueda a la documentación de FastAPI se pudo resolver rápido.

DESCRIPCIÓN DEL ENTORNO DE TRABAJO DEL CLIENTE E INSTRUCCIONES

Para poder ingresar a la parte del cliente se ejecuta ´cliente.py´ en CMD. En este archivo se encuentras diversas funciones que utilizando el módulo *requests* para poder utilizar la API sin tener conocimiento de las llamadas en el backend.

Al ejecutar pide el IP y el puerto del servidor (en mi caso, la PC que usé como servidor tenía asignado la IP 192.168.0.13 por DHCP, y uvicorn tenia el puerto 8000). El código utiliza estas 2 para crear el link que luego será pasado a todas las funciones al ser ejecutadas.

Luego se procede a preguntar si se quiere crear un nuevo usuario (que se hace tocando cualquier carácter seguido de intro) o si se desea acceder (tocar intro sin escribir nada), por lo que pide usuario y contraseña. El código del servidor procede a hacer la verificación y dar una Key valida durante 1 hora, que el código del cliente guarda y las pasa como ¨HEADER¨ en todas las llamadas a realizar.

Por último, aparece un menú donde da números para las distintas funciones y cada una de ellas se encarga de pedir lo requerido y realizar la llamada al servidor, devolviendo las respuestas.

El problema que mas tiempo me llevo fue resolver los headers, ya que de la manera que se pasaban era un json y se necesitaba realizar una combinación larga de comillas simples y comillas dobles para que el resultado pueda ser leído por el servidor.

REFERENCIAS

1. <https://fastapi.tiangolo.com/tutorial/>
2. <https://www.youtube.com/watch?v=7t2alSnE2-I&ab_channel=Bitfumes>

Link del repositorio:

<https://github.com/iairborgo/RestAPI_Redes>

Apéndice A REQUIERE DESCARGAR modelos.py DE GITHUB

from fastapi import FastAPI, status, Response, Depends

import pandas as pd

import re

from fastapi.responses import StreamingResponse

import modelos

from fastapi.security import OAuth2PasswordRequestForm

# Iniciamos fastapi y cargamos el json de los datos

app = FastAPI()

#users = pd.DataFrame(columns = ['User', 'Password'])

#users.to\_json('users.json')

# Use este regex para eliminar \n que aparecian al final de los links

# dbooks['link'] = dbooks['link'].str.replace('\\n', '', regex=True)

# Separo la columna que contiene los link de donde tengo guardada la imagen

#books\_images = dbooks[['title','imageLink']].copy()

#books\_images.to\_json('booksimagepath.json')

#dbooks.drop(columns='imageLink', inplace=True)

#dbooks.to\_json('books.json')

@app.post('/nuevo-usuario', status\_code=status.HTTP\_201\_CREATED, tags = ['Autenticacion'])

def nuevo\_usuario(usuario : modelos.Usuario, response : Response):

dbooks, books\_images, users = modelos.load\_data()

if usuario.user in users['User'].unique(): # Checkea si existe en la ''''base de datos'''''

response.status\_code = status.HTTP\_409\_CONFLICT

return('Nombre de usuario en uso')

users.loc[len(users)] = [usuario.user, modelos.Hash.bcrypt(usuario.password)] # Bcrypt es una funcion hash

users.to\_json('users.json')

return(f'Usuario {usuario.user} creado')

@app.post('/login', tags = ['Autenticacion'])

def login(response : Response, log\_credential: OAuth2PasswordRequestForm = Depends()):

dbooks, books\_images, users = modelos.load\_data()

user\_found = users[users['User'] == log\_credential.username]

if user\_found.empty:

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('Usuario o contraseña incorrecta')

hashedpass = user\_found['Password'].values[0]

if not modelos.Hash.autenticar(log\_credential.password, hashedpass): # checkea si la contraseña dada coincide con la encriptada

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('Usuario o contraseña incorrecta')

access\_token = modelos.create\_access\_token(data={"sub": log\_credential.username}, expires\_delta = None)

return modelos.Token(access\_token=access\_token, token\_type="bearer")

@app.get('/diccionario-autores', tags = ['Lista'])

def diccionario\_autores(current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Retorna los autores ordenados alfabeticamente

return(sorted(dbooks['author'].unique()))

@app.get('/diccionario-libros', tags = ['Lista'])

def diccionario\_autores(current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Retorna los títulos de los libros ordenados alfabeticamente

return(sorted(dbooks['title'].unique()))

@app.get('/busqueda-autor/{autor}', tags = ['Busqueda'])

def busqueda\_autor(autor : str, response : Response,

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Dado un autor (no es case sensitive y se puede separar nombre con espacios o \_) devuelve todos los libros suyos

autor = re.sub('%20', ' ', autor)

autor = re.sub('\_', ' ', autor)

autor = autor.lower()

query = dbooks[dbooks['author'].str.lower() == autor]

if query.empty :

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('Autor no encontrado, podes ver una lista ordenada en /diccionario-autores')

else:

return query.to\_dict(orient="records")

@app.get('/busqueda-libro/{libro}', tags = ['Busqueda'])

def busqueda\_libro(libro : str, response : Response,

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Dado un titulo, devuelve los datos de este. No es case-sensitive y se puede separar con espacios o \_

libro = re.sub('%20', ' ', libro)

libro = re.sub('\_', ' ', libro)

libro = libro.lower()

query = dbooks[dbooks['title'].str.lower() == libro]

if query.empty :

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('Libro no encontrado, podes ver una lista ordenada en /diccionario-libros')

else:

return query.to\_dict(orient="records")

@app.get('/busqueda-imagen/{titulo}', tags = ['Busqueda'])

def busqueda\_imagen(titulo : str, response : Response,

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Dado un título, devuelve la imagen

path = books\_images[books\_images['title'] == titulo.title() ]['imageLink']

if path.empty :

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('No existe imagen en la base de datos')

file = open(path.values[0], mode="rb")

return StreamingResponse(file, media\_type="image/png")

@app.put('/actualizar-link/{libro}', status\_code=status.HTTP\_202\_ACCEPTED, tags = ['Modificacion de Datos'])

def actualizar\_link(libro : str, path : str, response : Response, idioma : str = 'es',

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

# Dado un título, idioma del link (ej: ingles = en) y el subdirectorio, actualiza la lista

libro = re.sub('%20', ' ', libro)

libro = re.sub('\_', ' ', libro)

libro = libro.lower()

if libro not in [x.lower() for x in dbooks['title'].unique()]:

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('No encontramos el título en la base de datos')

link = f'https://{idioma}.wikipedia.org/wiki/{path}'

dbooks.loc[dbooks['title'].str.lower() == libro , 'link'] = link

dbooks.to\_json('books.json')

return(f'Link de {libro.title()} actualizado a: {link}')

@app.delete("/eliminar-libro", status\_code=status.HTTP\_202\_ACCEPTED, tags = ['Modificacion de Datos'])

def eliminar\_libro(libro: str, response : Response,

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

libro = re.sub('%20', ' ', libro)

libro = re.sub('\_', ' ', libro)

libro = libro.lower()

query = dbooks[dbooks['title'].str.lower() == libro].index

if query.empty:

response.status\_code = status.HTTP\_404\_NOT\_FOUND

return('No se ha encontrado el libro en la base de datos')

dbooks = dbooks.drop(index = query)

books\_images = books\_images.drop(index = books\_images[books\_images["title"].str.lower() == libro].index)

dbooks.to\_json("books.json")

books\_images.to\_json("booksimagepath.json")

return(f"{libro.title()} ha sido eliminado correctamente de la base de datos")

@app.post('/nuevo-libro', status\_code=status.HTTP\_201\_CREATED,tags = ['Modificacion de Datos'])

def nuevo\_libro(libro : modelos.Libro, response : Response,

current\_user : modelos.Usuario = Depends(modelos.get\_current\_user)):

dbooks, books\_images, users = modelos.load\_data()

if libro.title.title() not in dbooks['title'].unique():

dbooks.loc[len(dbooks)] = [libro.author.title(),

libro.country.title(),

libro.language.title(),

libro.link,

libro.pages,

libro.title.title(),

libro.year

]

books\_images.loc[len(books\_images)] = [libro.title.title(),libro.imageLink]

books\_images.to\_json('booksimagepath.json')

dbooks.to\_json('books.json')

else:

response.status\_code = status.HTTP\_406\_NOT\_ACCEPTABLE

return('El título se encuentra en la base de datos.')

return(f'Se ha insertado {libro.title.title()} a la lista.')

Apéndice B

import requests

import sys

import time

from os import system

from PIL import Image

import shutil

import re

def usuario(link : str):

ingresar = None

if not ingresar:

ingresar = input('\nDesea acceder? (presionar cualquier letra para crear nuevo usuario, enter para ingresar) \n')

user\_input = input('\nIngrese el nombre de Usuario: \n')

pass\_input = input('\nIngrese contraseña: \n')

while ingresar:

create\_payload = '{"user": "'+user\_input+'", "password": "'+pass\_input+'"}'

create = requests.post(link + '/nuevo-usuario',

headers={'Content-Type': 'application/json',

'accept': 'application/json'},

data=create\_payload)

if create.status\_code == 201:

print('\n\n\n Usuario Creado.')

ingresar = None

elif create.status\_code == 409:

print(f'\n\n{create.json()}')

user\_input = input('\nIngrese otro nombre de Usuario: \n')

pass\_input = input('\nIngrese contraseña: \n')

else:

print('\n\n\n Ha ocurrido un error. El programa se cerrará')

time.sleep(3)

sys.exit()

while not ingresar:

payload = f"grant\_type=&username={user\_input}&password={pass\_input}&client\_id=&client\_secret="

log = requests.post(link + '/login',

headers={'Content-Type': 'application/x-www-form-urlencoded',

'accept': 'application/json'},

data=payload)

if log.status\_code == 200:

print('\n\nIngreso exitoso. Acceso a la API durante 60 minutos.\n')

ingresar = 1

else:

print(f'\n\n{log.json()}')

user\_input = input('\nIngrese el nombre de Usuario: \n')

pass\_input = input('\nIngrese contraseña: \n')

headers = {'Authorization': log.json()['token\_type'].title() + ' ' + log.json()['access\_token'],

'accept' : 'application/json'}

return headers

def diccionario\_autores(sesion : requests.Session, link):

for autor in sesion.get(link+'/diccionario-autores').json():

print(autor)

print('\n\n')

def diccionario\_libros(sesion : requests.Session, link):

for libro in sesion.get(link+'/diccionario-libros').json():

print(libro)

print('\n\n')

def busqueda(sesion : requests.Session, link, tipo):

query = input('\nIngrese el nombre buscado:\n\n')

for encontrado in sesion.get(link+f'/busqueda-{tipo}/{query}').json():

print('\n')

for item, value in encontrado.items():

print(f'{item}: {value}')

print('\n\n')

def busqueda\_autor(sesion : requests.Session, link):

return busqueda(sesion, link, 'autor')

def busqueda\_libro(sesion : requests.Session, link):

return busqueda(sesion, link, 'libro')

def busqueda\_imagen(sesion : requests.Session, link):

query = input('\nIngrese el nombre buscado:\n\n')

response = sesion.get(link+f'/busqueda-imagen/{query}', stream=True)

with open('img.png', 'wb') as out\_file:

shutil.copyfileobj(response.raw, out\_file)

del response

img = Image.open('img.png')

img.show()

print('\n\n')

def actualizar\_link(sesion : requests.Session, link):

libro = input('\nIngrese el nombre del libro al que va a realizar la modificacion:\n')

path = input('Ingrese el subdirectorio de la wiki (ej https://es.wikipedia.org/wiki/Ficciones ingreso Ficciones):\n')

idioma = input('Ingrese idioma de la wiki (si es en español puede tocar enter).\n')

idioma = f'?idioma={idioma}' if idioma else ''

response = sesion.put(link+f'/actualizar-link/{libro}?path={path}{idioma}')

print(f'\n{response.json()}\n')

def nuevo\_libro(sesion : requests.Session, link):

post\_headers={'Content-Type': 'application/json'}

print('Para ingresar libro a la base de datos se requiere que ingrese los siguientes campos:\n')

autor = input('Autor (Unknown si desconoce): ').title()

pais = input('Pais de origen: ').title()

lengua = input('Lenguaje escrito: ').title()

wiki = input('Link Wikipedia: ')

paginas = input('Cantidad de páginas: ')

titulo = input('Nombre del libro: ').title()

anio = input('Año de lanzamiento: ')

payload = ('{"author": "'+autor+

'","country": "'+pais+

'","language": "'+lengua+

'","link": "'+wiki+

'","pages": '+paginas+

',\"title":"'+titulo+

'","year": '+anio+'}')

response = sesion.post(link + '/nuevo-libro', data = payload, headers=post\_headers)

print(f'\n{response.json()}\n')

def eliminar\_libro(sesion : requests.Session, link):

libro = input('\nIngrese el nombre del libro que quiere eliminar:\n')

response = sesion.delete(link+f'/eliminar-libro?libro={libro}')

print(f'\n{response.json()}\n')

def salir(sesion, link):

system('cls')

print("Adios")

time.sleep(1)

sys.exit()

def display\_menu(menu):

print('\n')

for k, function in menu.items():

print(k, re.sub('\_', ' ',function.\_\_name\_\_).capitalize())

print('\n')

def main():

ip = input('Ingrese la ip del servidor: \n')

puerto = input('\nIngrese el puerto al que accede: \n')

link = f'http://{ip}:{puerto}'

headers = usuario(link)

sesion = requests.Session()

sesion.headers.update(headers)

functions\_names = [diccionario\_autores, diccionario\_libros,

busqueda\_autor, busqueda\_libro, busqueda\_imagen,

actualizar\_link, eliminar\_libro, nuevo\_libro,

salir]

menu\_items = dict(enumerate(functions\_names, start = 1))

while True:

display\_menu(menu\_items)

select = int(input('Ingresa el numero de la llamada que quiere realizar: '))

selected = menu\_items[select]

selected(sesion, link)

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

main()

1. <https://github.com/benoitvallon/100-best-books/blob/master/books.json> [↑](#footnote-ref-1)