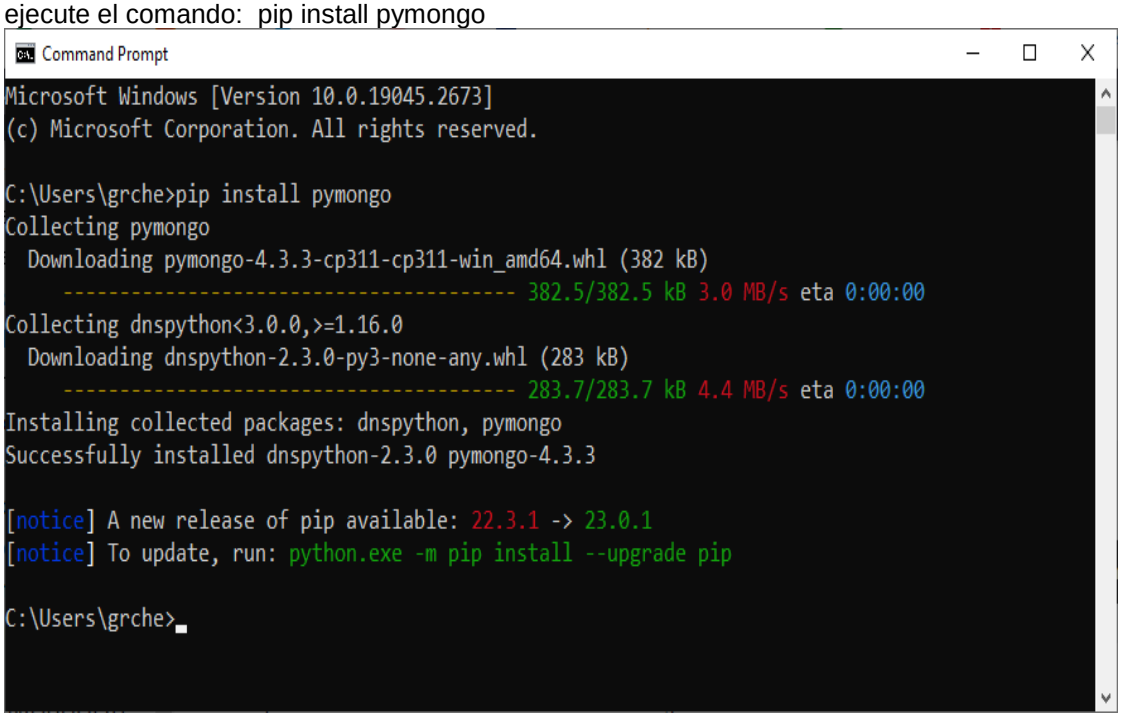
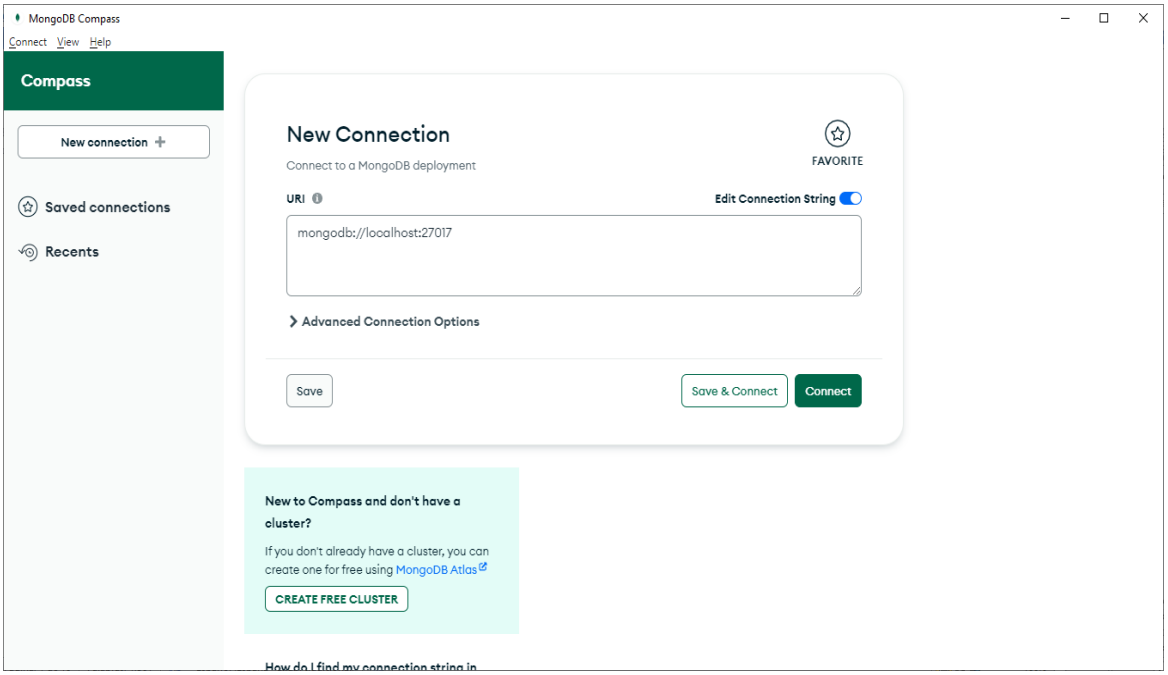
TRABAJO PRACTICO VI – Desarrollo de Aplicación Client-Server con Python y Mongodb

Ingrese a: https://www.grch.com.ar/docs/ap/

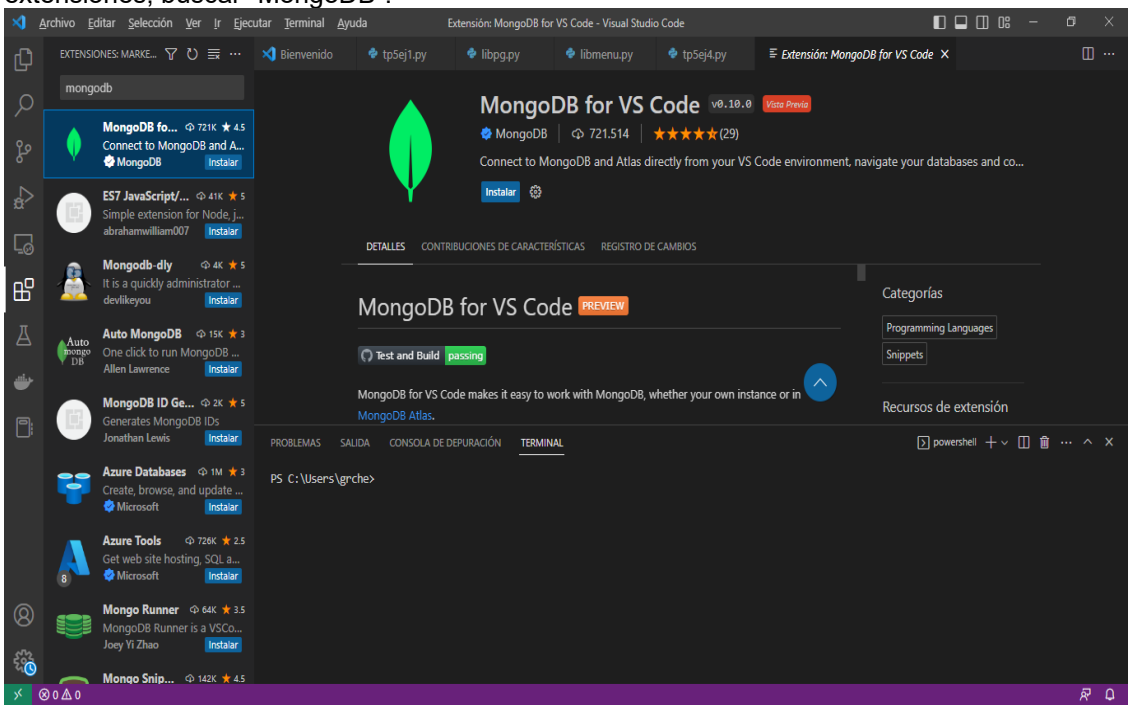
- Instalar MongoDB Community Edition (Community Server) para Windows, también instalar la aplicación cliente Compass.
- Instalar el driver python para mongoDB (pymongo), abra una consola Windows (cmd.exe) y ejecute el comando: pip install pymongo



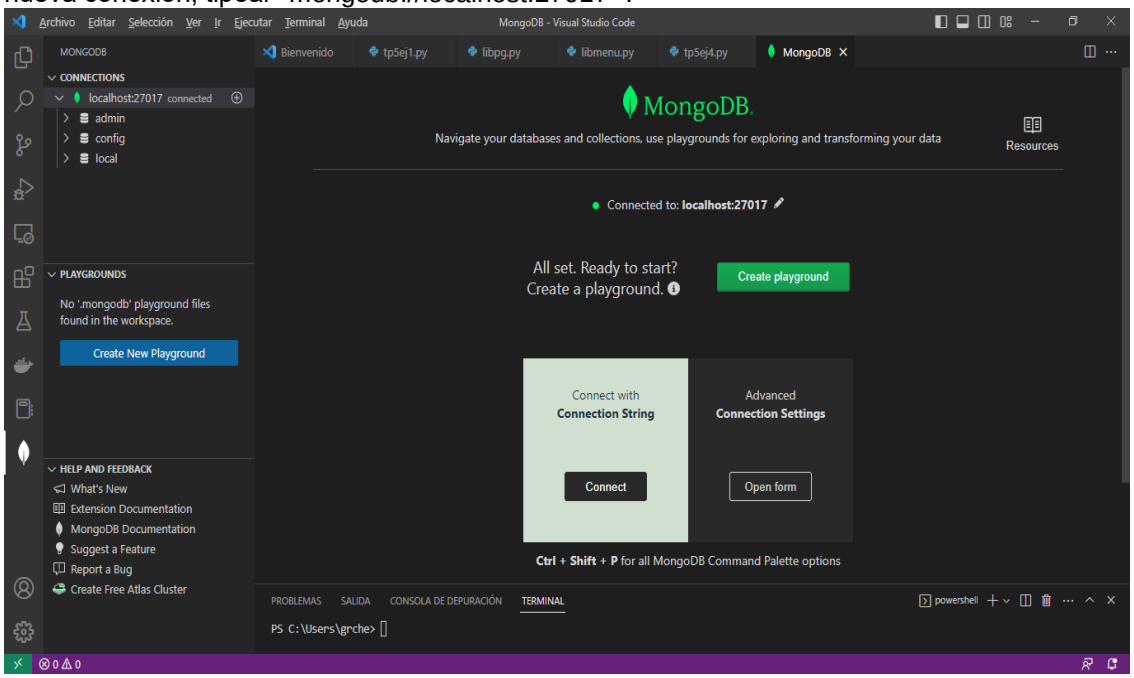
- Puede gestionar su servidor mongoDB desde la aplicación MongoDB Compass:



- También puede gestionar su servidor mongoDB desde Visual Studio Code (VSC), click en extensiones, buscar “MongoDB”:

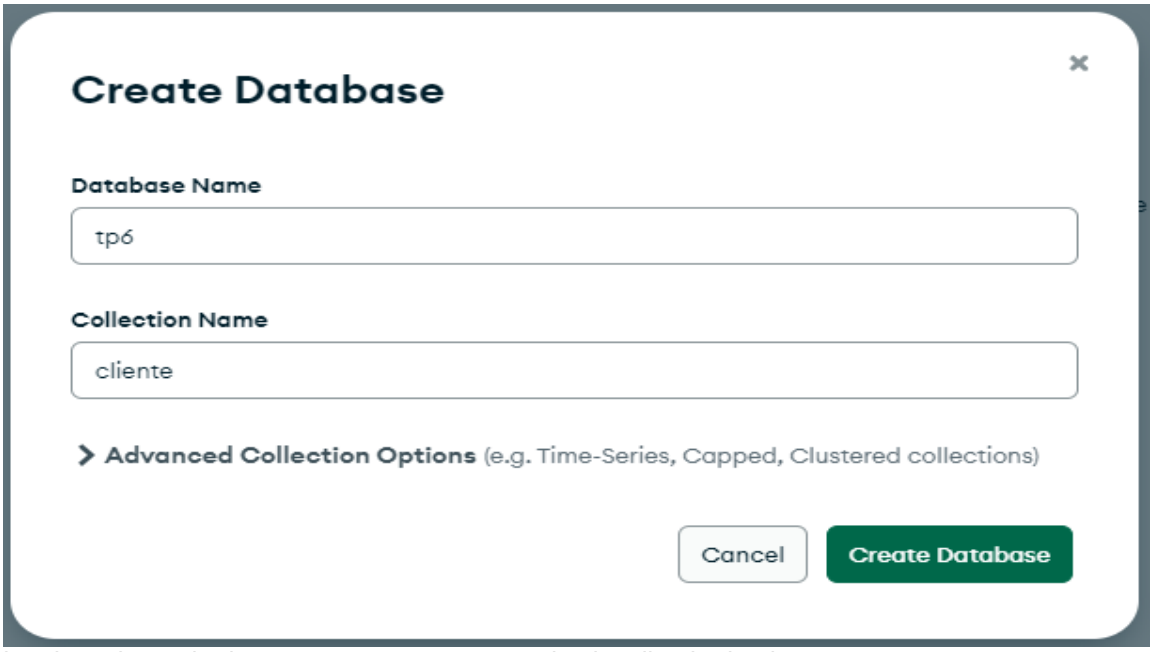


Instalar “MongoDB for VS Code”, luego hacer click en barra lateral, icono MongoDB, crear nueva conexión, tipear “mongodb://localhost:27017” :

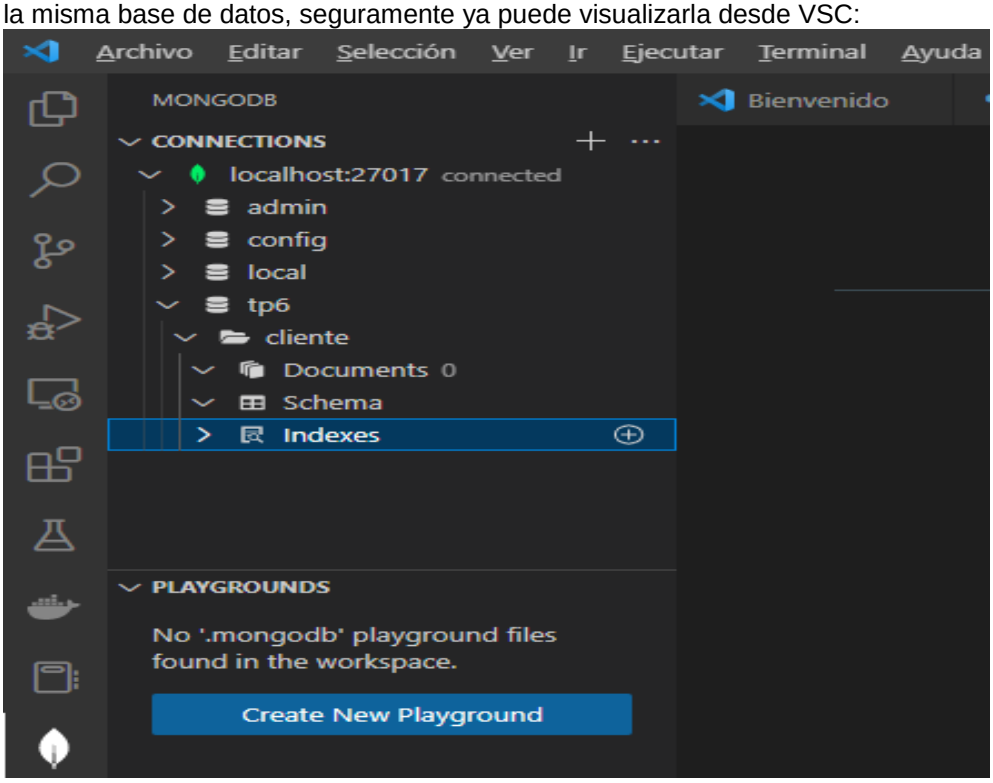


Programas fuentes mencionados aquí, disponibles en https://www.grch.com.ar/docs/ap/tp6/

1) Utilizamos la aplicación mongoDB Compass, creamos una nueva base de datos: **tp6** y la colección: **cliente**



la misma base de datos, seguramente ya puede visualizarla desde VSC:



2) Creamos el programa tp6ej2.py para utilizar y probar el driver mongoDB para python pymongo (ver https://pymongo.readthedocs.io/en/stable/tutorial.html) , el cual nos permite conectarnos, obtener acceso a la colección cliente, mostrar su contenido, agregar un nuevo cliente a la colección, etc:

```
import pymongo
from pymongo import MongoClient, errors
from pymongo.errors import ConnectionFailure
import pprint

client = None
try:
    # si el server o el puerto es incorrecto no lanza excepcion
    client = MongoClient('localhost', 27017)
    # aqui si lanza excepcion si hay algo mal
    client.admin.command('ismaster')
    # ahora si puedo decir que estoy conectado!
    print("Conectado a mongoDB server!")

    # conecto a la base de datos tp6 dentro del servidor
    db = client['tp6']
    print("Conectado a base de datos tp6!")

    # obtengo referencia a coleccion cliente
    coleccion = db['cliente']
    print("Accedi a coleccion cliente!")

    # muestro el contenido de la coleccion cliente
    clis = coleccion.find()

    # cuantos clientes hay en la coleccion
    cantidad = coleccion.count_documents({})
    print(f"Cantidad de Clientes: {cantidad}")

    # muestro los clientes de la coleccion usando
    # print y pretty print que muestra objetos json
    # de una forma mas amigable
    # ver https://docs.python.org/3/library/pprint.html
    for cli in clis:
        print(cli)
        print('-lo mismo pero mas bonito-')
        pprint.pprint(cli)

    # agrego un nuevo cliente a la coleccion
    codigo = int(input('Codigo:'))
    razon = input('Razon Social:')
```

```
    saldo = 0
    nuevo_cli = { "codigo" : codigo, "razon" : razon, "saldo" : saldo }
    nuevo_cli_id = coleccion.insert_one(nuevo_cli).inserted_id
    print(f"id cliente insertado: {nuevo_cli_id}")

except ConnectionFailure:
    print("Error en conexion a servidor mongoDB!")
```

ejecutando reiteradas veces este programa, podemos aprender que mongoDB genera un *object Id distinto, único*, por cada elemento que agregamos a la colección y dicho *object id* se guarda en la propiedad *_id* que mongoDB agrega automáticamente a nuestra colección. MongoDB crea un índice *unique* (que no admite repetidos) sobre la propiedad *_id* . No obstante, desde el punto de vista del usuario, el *_id* puede que no sea una clave primaria muy apropiada o representativa y por otro lado, mongoDB no controla la no repetición de los valores de ninguna otra propiedad.

3) Creamos el programa tp6ej3.py (a partir de tp6ej2.py), vamos a agregar un índice a la colección cliente por la propiedad codigo. Si la colección cliente tiene códigos repetidos, la creación del índice dará error, en tal caso, deberá borrar los documentos que duplican los valores en el campo codigo usando mongoDB Compass o bien la extensión para mongoDB de VSC que también permite borrar documentos. Este programa cada vez que lo ejecutamos vuelve a crear el índice, lo cual no es necesario, pero sirve para ver que sucede cuando intentamos repetir valores en campo codigo:

```
import pymongo
from pymongo import MongoClient, errors
from pymongo.errors import ConnectionFailure
import pprint

client = None
try:
    # si el server o el puerto es incorrecto no lanza excepcion
    client = MongoClient('localhost', 27017)
    # aqui si lanza excepcion si hay algo mal
    client.admin.command('ismaster')
    # ahora si puedo decir que estoy conectado!
    print("Conectado a mongoDB server!")

    # conecto a la base de datos tp6 dentro del servidor
    db = client['tp6']
    print("Conectado a base de datos tp6!")

    # obtengo referencia a coleccion cliente
    coleccion = db['cliente']
    print("Accedi a coleccion cliente!")

    # creo indice sobre la coleccion cliente, propiedad codigo, ascendente y
sin repeticiones
```

```
    result = coleccion.create_index([('codigo',
pymongo.ASCENDING)],unique=True)
    print('Creacion de indice por campo codigo ',result)

    # muestro el contenido de la coleccion cliente
    clis = coleccion.find()

    # cuantos clientes hay en la coleccion
    cantidad = coleccion.count_documents({})
    print(f"Cantidad de Clientes: {cantidad}")

    # muestro los clientes de la coleccion usando
    # print y pretty print que muestra objetos json
    # de una forma mas amigable
    # ver https://docs.python.org/3/library/pprint.html
    for cli in clis:
        print(cli)

    # agrego un nuevo cliente a la coleccion
    codigo = int(input('Codigo:'))
    razon = input('Razon Social:')
    saldo = 0
    nuevo_cli = { "codigo" : codigo, "razon" : razon, "saldo" : saldo }
    nuevo_cli_id = coleccion.insert_one(nuevo_cli).inserted_id
    print(f"id cliente insertado: {nuevo_cli_id}")

except ConnectionFailure:
    print("Error en conexion a servidor mongoDB!")
```

podemos obtener una salida como esta:

```
Conectado a mongoDB server!
Conectado a base de datos tp6!
Accedi a coleccion cliente!
Creacion de indice por campo codigo  codigo_1
Cantidad de Clientes: 6
{'_id':  ObjectId('63ff605d15a167254dc5d035'),  'codigo':  1234,  'razon':
'cliente1234', 'saldo': 0}
{'_id':  ObjectId('63ff613f63e58eb8700f1201'),  'codigo':  456,  'razon':
'cliente456', 'saldo': 0}
{'_id':  ObjectId('63ff61f5f062fa1f2039f273'),  'codigo':  2020,  'razon':
'cliente2020', 'saldo': 0}
{'_id':  ObjectId('63ff62059e692b9546b21f1d'),  'codigo':  2121,  'razon':
'cliente2121', 'saldo': 0}
{'_id':  ObjectId('63ff648d9178f33ec0909b4c'),  'codigo':  3131,  'razon':
'cliente3131', 'saldo': 0}
{'_id':  ObjectId('63ff7f8a0a2aea01d82f0ba3'),  'codigo':  2727,  'razon':
'cliente2727', 'saldo': 0}
Codigo:1234
```

```
Razon Social:juan
Traceback (most recent call last):
  File "d:\grchere\UNLu\varios\arg.programa.4\tp6\tp6ej3.py", line 46, in
<module>
    nuevo_cli_id = coleccion.insert_one(nuevo_cli).inserted_id
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\collection.py", line 628, in insert_one
self._insert_one(
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\collection.py", line 569, in _insert_one
    self.__database.client._retryable_write(acknowledged, _insert_command,
session)
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\mongo_client.py", line 1476, in _retryable_write
    return self._retry_with_session(retryable, func, s, None)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\mongo_client.py", line 1349, in _retry_with_session
    return self._retry_internal(retryable, func, session, bulk)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\_csot.py", line 105, in csot_wrapper
    return func(self, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\mongo_client.py", line 1390, in _retry_internal
    return func(session, sock_info, retryable)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\collection.py", line 567, in _insert_command
    _check_write_command_response(result)
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\helpers.py", line 217, in _check_write_command_response
    _raise_last_write_error(write_errors)
  File "C:\Users\grche\AppData\Local\Programs\Python\Python311\Lib\site-
packages\pymongo\helpers.py", line 189, in _raise_last_write_error
    raise DuplicateKeyError(error.get("errmsg"), 11000, error)
pymongo.errors.DuplicateKeyError: E11000 duplicate key error collection:
tp6.cliente index: codigo_1 dup key: { codigo: 1234 },
full error: {'index': 0, 'code': 11000, 'errmsg': 'E11000 duplicate key
error collection: tp6.cliente index: codigo_1 dup key: { codigo: 1234 }',
'keyPattern': {'codigo': 1}, 'keyValue': {'codigo': 1234}}
PS D:\grchere\UNLu\varios\arg.programa.4\tp6>
```

Esto sucede cuando intentamos duplicar la clave sobre la cual hemos creado un índice *unique*.

4) Si bien la API[i] de mongoDB es muy simple, no obstante, podemos agrupar en una librería varias operaciones: conectar, insertar, borrar, buscar, indexar, etc. Creamos el programa libmongo.py para agrupar allí las funciones:

```python
import pymongo
import pprint
from pymongo import MongoClient, errors
from pymongo.errors import *

def mgconectar(servidor,puerto,baseDeDato,coleccion):
    client = None
    db = None
    col = None
    ret = False
    try:
        # si el server o el puerto es incorrecto no lanza excepcion
        client = MongoClient(servidor, puerto)
        # aqui si lanza excepcion si hay algo mal
        client.admin.command('ismaster')
        # ahora si puedo decir que estoy conectado!
        print(f"Conectado a mongoDB server {servidor}:{puerto}!")

        # conecto a la base de datos tp6 dentro del servidor
        db = client[baseDeDato]
        print(f"Conectado a base de datos {baseDeDato}!")

        # obtengo referencia a coleccion cliente
        col = db[coleccion]
        print(f"Accedi a coleccion {coleccion}!")

        #retorno general de la funcion
        ret = True

    except: # capturo todos los errores!
        print("Error en conexion a servidor mongoDB!")
        ret = False

    finally:
        return client,db,col,ret

#devuelve el object_id insertado
def mginsertar(coleccion,docjson):
    objid = None
    try:
        objid = coleccion.insert_one(docjson).inserted_id
        print(f"Documento {objid} insertado Ok!")

    except: # capturo todos los errores!
        print("Documento no insertado!")
```

```
    finally:
        return objid

def mginsertarMuchos(coleccion,docsjson):
    for docjson in docsjson:
        mginsertar(coleccion,docjson)

#devuelve cursor con la coleccion de documentos encontrados
def mgbuscar(coleccion,queryDocjson=None):
    cursor = None
    try:
        if queryDocjson:
            cursor = coleccion.find(queryDocjson)
        else:
            cursor = coleccion.find()
        print('Busqueda Ok!')

    except: # capturo todos los errores!
        print("Error en Busqueda!")

    finally:
        return cursor

def mgindexar(coleccion,arregloClaves,unico=True):
    ret = None
    try:
        if unico:
            ret = coleccion.create_index(arregloClaves,unique=True)
        else:
            ret = coleccion.create_index(arregloClaves,unique=False)
        print(f"Indexacion por {arregloClaves} Ok!")

    except: # capturo todos los errores!
        print("Error en Indexacion!")

    finally:
        return ret

def mgborrar(coleccion,docjson):
    try:
        coleccion.delete_one(docjson)
        print(f"Documento {docjson} borrado Ok!")

    except: # capturo todos los errores!
        print("Documento no borrado!")
```

```
def mgborrarMuchos(coleccion,queryDocsjson):
    docsjson = mgbuscar(coleccion,queryDocsjson)
    for docjson in docsjson:
        mgborrar(coleccion,docjson)

def mgborrarTodos(coleccion):
    mgborrarMuchos(coleccion,{})
```

también se pudo haber usado insert_many() y delete_many() pero decidi hacerlo basado en insert_one() y delete_one(), pueden implementar por su cuenta esta otra forma de hacerlo.

uso de la función mgconectar:

```
client, db, coleccion, ret = mgconectar('localhost',27017,'tp6','cliente')
```

si ret devuelve True significa que todo anduvo Ok, caso contrario, hugo algún error.
Uso de la función mgindexar:

```
            campo = input('Nombre Campo/Propiedad:')
             mgindexar(coleccion,[(campo, pymongo.ASCENDING)],True)
```

uso de la función mgbuscar para buscar TODOS los documentos de la colección:

```
            docs = mgbuscar(coleccion)
```

para buscar un documento determinado:

```
            codigo = ingresoNum("Codigo:")
            query_cli = { "codigo" : codigo }
            docs = mgbuscar(coleccion,query_cli)
            for doc in docs:
                pprint.pprint(doc)
```

etc, etc

5) Copiamos el programa libmenu.py del TP V y le agregamos una nueva función (ingresoFloat) para el ingreso de numero de punto flotante para poder ingresar el saldo del cliente:

```
def menu(titulo,opciones):
    print("------------------")
    print(titulo)
    n=1
    for opcion in opciones:
        print(f"{n:3} {opcion}")
        n=n+1
    print("  0 Salir!")
    opcion=-1
    while opcion < 0 or opcion >= n:
```

```
        opcion = ingresoNum("Ingrese su Opcion:")

    return opcion

def ingresoNum(prompt):
    while True:
        try:
            opcion = int(input(prompt))
            return opcion
        except Exception as error:
            opcion = -1

def ingresoFloat(prompt):
    while True:
        try:
            opcion = float(input(prompt))
            return opcion
        except Exception as error:
            opcion = -1
```

6) Creamos el programa tp6ej6.py para usar la librería de menú (libmenu.py) y la librería mongoDB (libmongo.py). Este programa nos permite demostrar el uso de funciones y como jugar con la colección cliente de la base de datos tp6:

```
import libmongo
import libmenu
from libmongo import *
from libmenu import *

client, db, coleccion, ret = mgconectar('localhost',27017,'tp6','cliente')
if ret:
    opcion = -1
    while opcion != 0:
        opcion = menu("MENU PRINCIPAL",["Indexar Coleccion","Mostrar toda la
Coleccion", "Insertar Documento","Borrar Documento","Borrar toda la
Coleccion","Buscar Documento"])
        if opcion == 1:
            campo = input('Nombre Campo/Propiedad:')
            mgindexar(coleccion,[(campo, pymongo.ASCENDING)],True)
        elif opcion == 2:
            docs = mgbuscar(coleccion)
            cantidad = coleccion.count_documents({})
            print(f"Cantidad de Documentos: {cantidad}")
            for doc in docs:
```

```
                pprint.pprint(doc)
        elif opcion == 3:
            codigo = ingresoNum("Codigo:")
            razon = input("Razon:")
            saldo = ingresoFloat("Saldo:")
            nuevo_cli = { "codigo" : codigo, "razon" : razon, "saldo" : saldo
}
            nuevo_cli_id = mginsertar(coleccion,nuevo_cli)
            print(f"id cliente insertado: {nuevo_cli_id}")
        elif opcion == 4:
            codigo = ingresoNum("Codigo:")
            query_cli = { "codigo" : codigo }
            mgborrar(coleccion,query_cli)
        elif opcion == 5:
            mgborrarTodos(coleccion)
        elif opcion == 6:
            codigo = ingresoNum("Codigo:")
            query_cli = { "codigo" : codigo }
            docs = mgbuscar(coleccion,query_cli)
            cantidad = coleccion.count_documents(query_cli)
            print(f"Cantidad de Documentos: {cantidad}")
            for doc in docs:
                pprint.pprint(doc)
print('Fin!')
```

Desde el punto de vista teórico, obsérvese como tp6ej6.py se independiza del driver mongoDB (no hace falta importar pymongo) gracias al uso de la capa de software libmongo, si mañana queremos usar este programa con otro tipo de base de dato, será cuestión de cambiar a libmongo por lo que corresponda, si no cambiamos la api, este programa podría seguir funcionando con muy pocos cambios. Esto es modularidad.

7) Creamos el programa tp6ej7.py para usar la librería de menú (libmenu.py) y la librería mongoDB (libmongo.py). Este programa nos permitirá implementar las siguientes opciones de menu:

1 Insertar muchos documentos
2 Buscar por razón social
3 Buscar por un rango de códigos
4 Mostrar colección ordenada por razón social
5 Buscar por saldo
6 Cambiar un Documento
0 Sallir

1 Insertar muchos documentos: el usuario ingresará los datos de un cliente a insertar, se realizará la inserción y se le preguntará al usuario si desea seguir ingresando clientes, si responde que si, se continúa con el ingreso, sino se sale de esta opción

2 Buscar por razón social: se ingresa por teclado parte de la razón social, el programa muestra en pantalla todos los clientes cuya razón social coincide, comienza con lo tipeado por el usuario

3 Buscar por un rango de códigos: se ingresa por teclado un código (desde) y luego otro código (hasta), se muestran por pantalla, todos los clientes cuyo código sea mayor o igual que desde y menor o igual que hasta

4 Mostrar colección ordenada por razón social: sin comentarios

5 Buscar por saldo: el usuario ingresa por teclado un saldo y el programa muestra todos los clientes cuyo saldo sea mayor o igual que el ingresado por el usuario, ordenado en forma descendente a partir del saldo más grande hasta el saldo más pequeño.

6 Cambiar un Documento: el usuario ingresa un codigo, el programa muestra el documento correspondiente a ese código (si no existe, muestra mensaje de error), permite que el usuario tipee una nueva razón social y un nuevo saldo y actualiza estos datos en la colección cliente. Modifique libmongo.py para que ahora también utilice a la función update_one() de pymongo y cree una nueva función para utilizarla desde tp6ej7.py

[i] API (Application Program Interface) interfaz de aplicación, conjunto de funciones con las cuales puedo interactuar con un software determinado.