



Deusto

Facultad de Ingeniería
Ingeniaritza Fakultatea

ARQUITECTURA TECNOLÓGICA PARA BIG DATA

PRÁCTICA 2 – BD NoSQL

Autor

Asier Bujedo Álvarez

Docente

Ana Isabel Torre Bastida

Fecha

2 de diciembre de 2024

Repositorio

<https://github.com/AsierBujedo/ATBD-PR02>

Índice de contenido

1.	Base de datos seleccionada y justificación	1
2.	Definición del esquema y las sentencias de creación	3
1.1	Sentencias de creación	5
1.2	Explicación y ejecución de scripts	5
1.2.1	Ejecución.....	6
3.	Inserción de 100 registros	7
4.	Modificación de dos registros.....	8
5.	Consultas específicas	9
5.1	Consulta por una jugadora específica	9
5.1.1	Filtrando por el año de comienzo en el fútbol mayor de 2020	9
5.1.2	Filtrando por un equipo que empiece por “Manchester...”	10
5.2	Consulta por un país concreto donde juega una jugadora.....	11
5.3	Tabla de tiempos de ejecución	12
6.	Código.....	13
6.1	Main.py.....	13
6.2	modifyRegistries.py	16
6.3	queries.py.....	17
7.	Bibliografía	19

Índice de ilustraciones

Ilustración 1 - Formato del dato en JSON.....	4
Ilustración 2 - Creación de índices de mongo	5
Ilustración 3 - Transformación de datos	7
Ilustración 4 - Registro de muestra.....	7
Ilustración 5 - Registros modificados	8
Ilustración 6 - Algoritmo de medición de tiempos	9
Ilustración 7 – Muestra - Resultado de la ejecución 5.1.1.....	9
Ilustración 8 – Resultado de la ejecución 5.1.2	10
Ilustración 9 – Muestra - Resultado de la ejecución 5.2	11
Ilustración 10 - Tabla de tiempos de ejecución	12

1. BASE DE DATOS SELECCIONADA Y JUSTIFICACIÓN

Se ha decidido escoger MongoDB porque los datos del dataset seleccionado encajan perfectamente con su modelo documental. MongoDB almacena la información como documentos en formato JSON, ofreciendo un esquema que no requiere de estructuras predefinidas (como en las bases de datos relacionales). Esto permite manejar datos de distinto tipo y adaptarse a cambios en su estructura sin necesidad de realizar modificaciones en la base de datos. En este caso, los datos pueden agruparse de manera lógica en documentos JSON. El dataset incluiría datos como las estadísticas de las jugadoras, equipos o características individuales de las mismas.

Cassandra, por su parte, está diseñada para casos que requieren de una alta escalabilidad y disponibilidad en la gestión de datos distribuidos. Su arquitectura descentralizada y su enfoque en particionado y replicación la hacen mejor base de datos para operaciones masivas de escritura o consultas para claves específicas. Sin embargo, este dataset no exige ese tipo de capacidad, ya que su principal necesidad es manejar datos que describen entidades con atributos detallados sobre las jugadoras, no grandes volúmenes de operaciones concurrentes. Cassandra es menos eficiente para representar relaciones jerárquicas o agrupar información relacionada dentro de una misma unidad de datos, algo que MongoDB resuelve de manera natural.

En comparación con Neo4j, que se especializa en modelar relaciones mediante grafos, MongoDB es más adecuada para este caso porque el dataset no está orientado al análisis de conexiones entre entidades. Aunque Neo4j sería una elección excelente para explorar relaciones como transferencias de jugadoras entre equipos o redes sociales (como se habló en clase), este dataset está creado para la consulta de datos descriptivos de las jugadoras, como, entre otras cosas, la nacionalidad, las estadísticas o los equipos a los que pertenecen, por lo que MongoDB, se convierte en la base de datos idónea.

Finalmente, InfluxDB, como base de datos para series temporales, está diseñada para gestionar datos dependientes del tiempo. Aunque podría ser útil en escenarios donde se analice el rendimiento de las jugadoras a lo largo del tiempo, no es el enfoque principal de este dataset. MongoDB, permite manejar este tipo de información en caso de ser necesario, pero además proporciona una solución más amplia para los datos documentales de este caso de uso.

2. DEFINICIÓN DEL ESQUEMA Y LAS SENTENCIAS DE CREACIÓN

Dado que MongoDB es una base de datos NoSQL orientada a documentos, el esquema se representa como una estructura lógica en formato JSON.

La sección “PersonalInfo” contiene información sobre cada jugadora, estructurada en subcomponentes. Entre ellos se encuentran el país de origen en el campo “Nation”, y “Position” se modela como una lista para reflejar la posibilidad de que una jugadora tenga múltiples posiciones que puede desempeñar a lo largo de su carrera. Además, se incluye un sub-objeto “SquadInfo”, que almacena el nombre del equipo (“Squad”) y un identificador único del mismo (“SquadID”), lo que permite organizar y consultar fácilmente los datos según el equipo al que pertenece la jugadora.

También dentro de “PersonalInfo” se encuentran detalles específicos como la edad de la jugadora (“Age”), el año de nacimiento (“Born”), y el año estimado de inicio de su carrera profesional (“EstimatedStartYear”), que ayudan a contextualizar su trayectoria. Por último, el identificador único de la jugadora se almacena en “PlayerID”, garantizando que cada registro pueda ser identificado de manera inequívoca.

Por otro lado, las estadísticas de rendimiento, como goles, asistencias y minutos jugados, se agrupan en “PerformanceStats”, mientras que las relacionadas con la defensa, como recuperaciones y duelos aéreos, están en “DefensiveStats”. Dentro de este último, el sub-objeto “AerialDuels” detalla la cantidad de duelos aéreos ganados, perdidos y la tasa de éxito, ofreciendo un análisis más específico de este aspecto del juego.

Por último, las estadísticas disciplinarias, como tarjetas amarillas y rojas, se agrupan en “DisciplinaryStats”, y eventos especiales, como penales ganados o goles en propia puerta, se encuentran en “SpecialEvents”. Finalmente, métricas avanzadas, como el porcentaje de minutos jugados respecto al máximo posible, están en “AdvancedStats”.

```
{
  "Player": "string",
  "PersonalInfo": {
    "Nation": "string",
    "Position": ["string"],
    "SquadInfo": {
      "Squad": "string",
      "SquadID": "string"
    },
    "PlayerID": "string",
    "Age": "integer",
    "Born": "integer",
    "EstimatedStartYear": "integer"
  },
  "PerformanceStats": {
    "MatchesPlayed": "integer",
    "Starts": "integer",
    "MinutesPlayed": "integer",
    "Goals": "integer",
    "NonPenaltyGoals": "integer",
    "PenaltiesScored": "integer",
    "PenaltiesAttempted": "integer",
    "Assists": "integer",
    "AssistPerMatch": "float",
    "SubstituteAppearances": "integer"
  },
  "DefensiveStats": {
    "TacklesWon": "integer",
    "Recoveries": "integer",
    "AerialDuels": {
      "Won": "integer",
      "Lost": "integer",
      "SuccessRate": "float"
    }
  },
  "DisciplinaryStats": {
    "YellowCards": "integer",
    "SecondYellowCards": "integer",
    "RedCards": "integer"
  },
  "SpecialEvents": {
    "Crosses": "integer",
    "OwnGoals": "integer",
    "Penalties": {
      "Won": "integer",
      "Conceded": "integer"
    }
  },
  "AdvancedStats": {
    "MinutesPlayedRatio": "float"
  }
}
```

Ilustración 1 - Formato del dato en JSON

Además, debe destacarse que “EstimatedStartYear” es un atributo calculado. Debido a la alta dificultad de encontrar un dataset que cumpla todos los requisitos del enunciado, se ha optado por realizar un pequeño cálculo aproximatorio con aleatorización que asigna un valor a dicho campo. De esta forma, se pretende cumplir con los requisitos y realizar los posteriores ejercicios de esta práctica. El script de asignación aproximatorio con aleatorización de adjunta en los anexos de este documento.

1.1 SENTENCIAS DE CREACIÓN

MongoDB no impone un esquema rígido, pero el uso de índices es una buena práctica para mejorar el rendimiento de las consultas, por lo que se ha optado en utilizarlos. Los índices permiten a MongoDB buscar documentos dentro de una colección de forma más eficiente, especialmente en campos que se consultan frecuentemente.

En este caso, se han añadido índices en los campos Player, Nation, y Squad para facilitar consultas rápidas en esos atributos.

```
from pymongo import MongoClient

client = MongoClient('mongodb://<UNAME>:<UPASS>@<SERVER>:<PORT>/')
db = client['football']
collection = db['players']

collection.create_index("Player")
# Búsqueda rápida por jugador
collection.create_index("PlayerInfo.Nation")
# Búsqueda rápida por nacionalidad
collection.create_index("PlayerInfo.SquadInfo.Squad")
# Búsqueda rápida por equipo
```

Ilustración 2 - Creación de índices de mongo

1.2 EXPLICACIÓN Y EJECUCIÓN DE SCRIPTS

El script “main.py” adapta datos desde un archivo CSV para insertarlos en una base de datos MongoDB, estructurándolos en formato documental mediante Python. Utiliza pandas para transformar las filas del dataset en documentos JSON, organizando atributos en subdocumentos para optimizar consultas y agrupaciones. Además, conecta con MongoDB utilizando credenciales seguras desde un archivo .env (que deberá

modificarse para cada caso), elimina datos redundantes en la colección existente, y crea índices en campos clave como Player, Squad y EstimatedStartYear para mejorar el rendimiento. Los datos transformados se insertan en la base de datos, con un caso especial donde se filtra por jugadores cuyo equipo contiene "Manchester", agregándolos como registros adicionales.

Por otro lado, "transform.py" prepara el dataset original para su uso en MongoDB. Calcula un campo adicional llamado "Estimated_Start_Year", que estima el año de inicio de la carrera de cada jugadora basándose en su edad y otros atributos. El cálculo utiliza funciones personalizadas que consideran el número de partidos como titular y el año de nacimiento de la jugadora. Se ha utilizado este método debido a la alta dificultad de encontrar un dataset que cumpla todos los requisitos exigidos por el enunciado.

Por último, "modifyRegistries.py" y "queries.py" contienen las consultas y el cálculo de los tiempos requeridos por el enunciado.

1.2.1 Ejecución

1. Creación de un entorno virtual en Python e instalar las dependencias.
 - a. `$ python -m venv env`
 - b. (Windows) `$ env\Scripts\activate`
 - c. (Linux) `$ source env/bin/activate`
 - d. `$ pip install -r requirements.txt`
2. Modificar el fichero .env para configurar las 4 variables de acceso a mongo.
3. Ejecutar los ficheros en este orden:
 - a. `$ python main.py`
 - b. `$ python modifyRegisters.py`
 - c. `$ python queries.py`

***Nota:** Los logs se almacenan en un fichero autogenerado en la ruta desde donde se ha ejecutado cada programa.

3. INSERCIÓN DE 100 REGISTROS

MongoDB dispone de una función para insertar varios registros al mismo tiempo, este es el método que se utilizará para completar este apartado de la práctica.

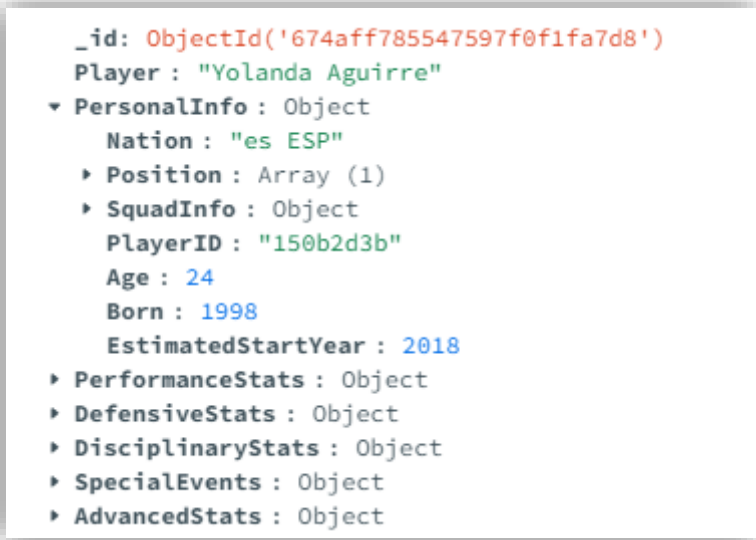
Haciendo uso de pandas y de pymongo, se procederá a la lectura del csv y a crear un dataframe para posteriormente, enviarlo a MongoDB.

```
csv_file = './data/dataset.csv'  
data = pd.read_csv(csv_file)  
data_subset = data.head(100)  
documents = [transform_row(row) for _, row in data_subset.iterrows()]
```

Ilustración 3 - Transformación de datos

El método head() se utiliza directamente para la obtención de X registros del dataframe “data”, de esta forma se evita, por ejemplo, la utilización de bucles y/o listas y gastar recursos computacionales de forma innecesaria. Por último, insert_many() envía de golpe los 100 registros escogidos mediante head() a MongoDB.

```
result = collection.insert_many(documents)
```



```
_id: ObjectId('674aff785547597f0f1fa7d8')  
Player : "Yolanda Aguirre"  
▼ PersonalInfo : Object  
  Nation : "es ESP"  
  ▶ Position : Array (1)  
  ▶ SquadInfo : Object  
    PlayerID : "150b2d3b"  
    Age : 24  
    Born : 1998  
    EstimatedStartYear : 2018  
  ▶ PerformanceStats : Object  
  ▶ DefensiveStats : Object  
  ▶ DisciplinaryStats : Object  
  ▶ SpecialEvents : Object  
  ▶ AdvancedStats : Object
```

Ilustración 4 - Registro de muestra


4. MODIFICACIÓN DE DOS REGISTROS

Los dos registros se han modificado directamente en la colección players de MongoDB utilizando el script modifyRegistries.py. Este script busca cada registro por el nombre de la jugadora (Teresa Abilleira y Jessica Aby) usando un filtro exacto en el campo Player. Luego, actualiza el valor del nombre a mayúsculas empleando la operación \$set. Cada modificación se ejecuta por separado y se mide el tiempo de ejecución para evaluar el rendimiento de las actualizaciones.

```
name2 = "Jessica Aby"
start_time = time.time()
collection.update_one(
    {"Player": name2},
    {"$set": {"Player": name2.upper()}}
)
end_time = time.time()
execution_time2 = end_time - start_time
```

Otra opción, sería ejecutar la consulta directamente en la terminal de MongoDB mediante la siguiente sentencia:

```
db.players.updateOne(
    { "Player": "Jessica Aby" },
    { $set: { "Player": "Jessica Aby".toUpperCase() } }
);
```



```
_id: ObjectId('6744385b2df3d50464ba9827')
Player : "TERESA ABILLEIRA"
Nation : "es ESP"
> Position : Array (1)
Squad : "Real Madrid"
Age : 22
Born : 2000
EstimatedStartYear : 2020
> Stats : Object
> OtherStats : Object

_id: ObjectId('6744385b2df3d50464ba9828')
Player : "JESSICA ABY"
Nation : "ci CIV"
> Position : Array (2)
Squad : "Alavés"
Age : 24
Born : 1998
EstimatedStartYear : 2021
> Stats : Object
> OtherStats : Object
```

Ilustración 5 - Registros modificados

5. CONSULTAS ESPECÍFICAS

Todas las consultas tienen un método en común mediante el cual se mide el tiempo en la que tarda cada una en ejecutarse. Es un método que recibe y ejecuta la query para posteriormente, devolver el resultado en formato JSON imprimiéndolo por pantalla.

```
def measure_query_time(query):  
    start_time = time.time()  
    result = list(collection.find(query))  
    end_time = time.time()  
    execution_time = end_time - start_time  
    pretty_result = json.dumps(result, indent=4, default=str)  
    return execution_time, pretty_result
```

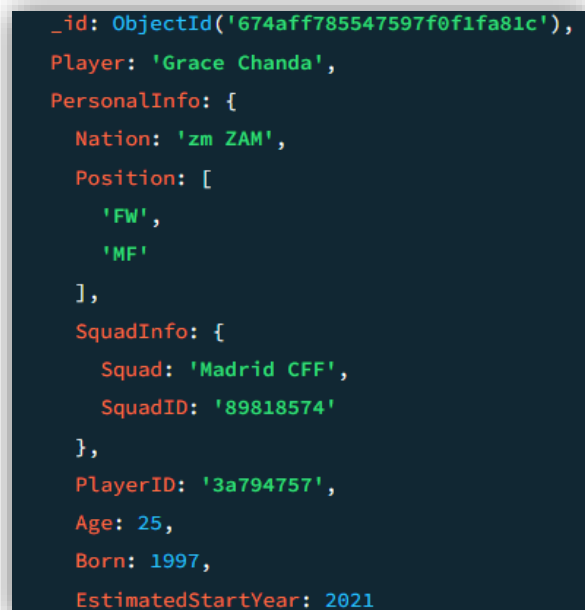
Ilustración 6 - Algoritmo de medición de tiempos

5.1 CONSULTA POR UNA JUGADORA ESPECÍFICA

5.1.1 Filtrando por el año de comienzo en el fútbol mayor de 2020

Haciendo uso del campo “EstimatedStartYear” y la función \$gt (greater than) de mongo, se obtendrán todos los registros cuyo campo “EstimatedStartYear” tenga un valor superior al especificado a la función \$gt.

```
query_ai = {"EstimatedStartYear": {"$gt": 2020}}  
time_ai, result_ai = measure_query_time(query_ai)
```



```
{  
  "_id": ObjectId('674aff785547597f0f1fa81c'),  
  "Player": "Grace Chanda",  
  "PersonalInfo": {  
    "Nation": "zm ZAM",  
    "Position": [  
      "FW",  
      "MF"  
    ],  
  },  
  "SquadInfo": {  
    "Squad": "Madrid CFF",  
    "SquadID": "89818574"  
  },  
  "PlayerID": "3a794757",  
  "Age": 25,  
  "Born": 1997,  
  "EstimatedStartYear": 2021  
}
```

Ilustración 7 – Muestra - Resultado de la ejecución 5.1.1


5.1.2 Filtrando por un equipo que empiece por “Manchester...”

Haciendo uso del campo “Squad”, expresiones regulares y la función \$options de mongo, se obtendrán todos los registros cuyo campo “Squad” empiece (\$options: ‘i’) por “Manchester” (\$regex: “^Manchester”)

```
query_a11 = {"Squad": {"$regex": "^Manchester", "$options": "i"}}  
time_a11, result_a11 = measure_query_time(query_a11)
```

Otra opción, sería ejecutar la consulta directamente en la terminal de MongoDB mediante la siguiente sentencia:

```
db.players.find({"PersonalInfo.SquadInfo.Squad": {"$regex": "^Manchester", "$options": "i"}})
```



```
_id: ObjectId('674aff785547597f0f1fa838'),  
Player: 'Laia Aleixandri',  
PersonalInfo: {  
  Nation: 'es ESP',  
  Position: [  
    'DF',  
    'MF'  
  ],  
  SquadInfo: {  
    Squad: 'Manchester City',  
    SquadID: '9ce68f8a'  
  },  
  PlayerID: '31691fc9',  
  Age: 22,  
  Born: 2000,  
  EstimatedStartYear: 2019
```

Ilustración 8 – Resultado de la ejecución 5.1.2

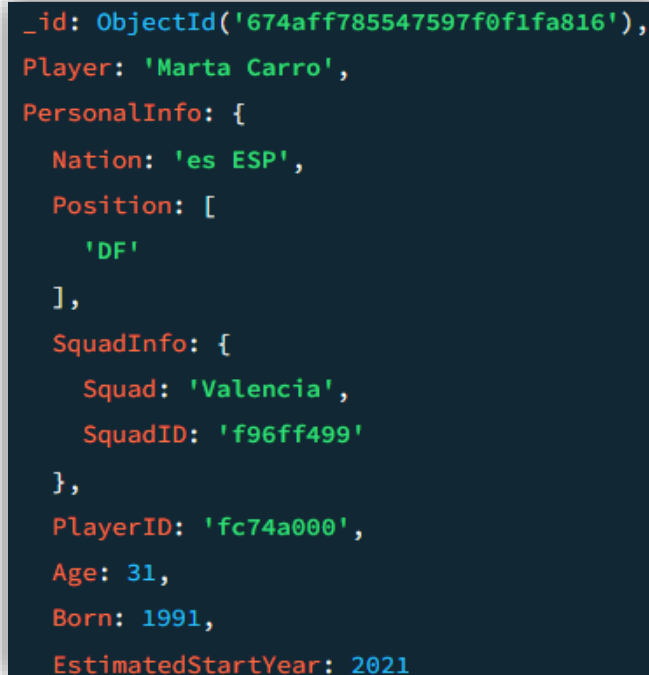
5.2 CONSULTA POR UN PAÍS CONCRETO DONDE JUEGA UNA JUGADORA

Por último, para filtrar por país, en este caso, España (“es ESP”), solamente habrá que obtener los registros cuyo campo “Nation” sea “es ESP”.

```
query_b = {"Nation": "es ESP"}  
time_b, result_b = measure_query_time(query_b)
```

Otra opción, sería ejecutar la consulta directamente en la terminal de MongoDB mediante la siguiente sentencia:

```
db.players.find({"PersonalInfo.Nation": "es ESP"})
```



```
_id: ObjectId('674aff785547597f0f1fa816'),  
Player: 'Marta Carro',  
PersonalInfo: {  
  Nation: 'es ESP',  
  Position: [  
    'DF'  
  ],  
  SquadInfo: {  
    Squad: 'Valencia',  
    SquadID: 'f96ff499'  
  },  
  PlayerID: 'fc74a000',  
  Age: 31,  
  Born: 1991,  
  EstimatedStartYear: 2021
```

Ilustración 9 – Muestra - Resultado de la ejecución 5.2

5.3 TABLA DE TIEMPOS DE EJECUCIÓN

Ejecución	Tiempo
Año de comienzo mayor que 2020	0.005805 (s)
El equipo comienza por Manchester	0.000690 (s)
Por un país concreto	0.000701 (s)

Ilustración 10 - Tabla de tiempos de ejecución

Para las consultas realizadas sobre mongo, se ha medido el tiempo de ejecución de cada una para evaluar el rendimiento y extraer conclusiones sobre posibles optimizaciones en el esquema o índices existentes. La consulta que filtra jugadoras por el año de inicio de su carrera muestra un tiempo más alto debido al gran volumen de resultados devueltos, lo que resalta la necesidad de mantener el índice en “EstimatedStartYear”. Las consultas relacionadas con la búsqueda por equipo y país presentan tiempos mucho más bajos, indicando que los índices existentes en Squad y Nation son bastante efectivos para cada uno de estos casos.

6. CÓDIGO

6.1 MAIN.PY

```
import os
import pandas as pd
import logging as logger
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv

log = logger.getLogger("pr02")
logger.basicConfig(filename='pr02.log', level=logger.INFO)

load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
UNAME = quote_plus(os.getenv("USR"))
UPASS = quote_plus(os.getenv("PWD"))

def transform_row(row):
    aerial_duels_won = row["AerialDuels_Won"] if
pd.notnull(row["AerialDuels_Won"]) else 0
    aerial_duels_lost = row["AerialDuels_Lost"] if
pd.notnull(row["AerialDuels_Lost"]) else 0
    aerial_duels_total = aerial_duels_won + aerial_duels_lost
    aerial_duels_success_rate = (aerial_duels_won /
aerial_duels_total) * 100 if aerial_duels_total > 0 else 0

    minutes_played = row["Min"] if pd.notnull(row["Min"]) else 0
    matches_played = row["MP"] if pd.notnull(row["MP"]) else 0
    max_minutes_possible = matches_played * 90
    minutes_played_ratio = (minutes_played / max_minutes_possible) *
100 if max_minutes_possible > 0 else 0

    assists = row.get("Ast", 0) if pd.notnull(row.get("Ast", None))
else 0
    assist_per_match = (assists / matches_played) if matches_played >
0 else 0

    return {
        "Player": row["Player"],
        "PersonalInfo": {
            "Nation": row["Nation"],
            "Position": row["Pos"].split(",") if
pd.notnull(row["Pos"]) else [],
            "SquadInfo": {
                "Squad": row["Squad"],
                "SquadID": row["Squad_id"],
            },
            "PlayerID": row["Player_id"],
            "Age": int(row["Age"].split("-")[0]) if
pd.notnull(row["Age"]) else None,
            "Born": int(row["Born"]) if pd.notnull(row["Born"]) else
None,
```

```

    "EstimatedStartYear":
    int(row["Estimated_Start_Year"]) if
    pd.notnull(row["Estimated_Start_Year"]) else None,
    },
    "PerformanceStats": {
        "MatchesPlayed": matches_played,
        "Starts": row["Starts"] if pd.notnull(row["Starts"]) else
0,
        "MinutesPlayed": minutes_played,
        "Goals": row["Gls"] if pd.notnull(row["Gls"]) else 0,
        "NonPenaltyGoals": row["G-PK"] if pd.notnull(row["G-PK"])
else 0,
        "PenaltiesScored": row["PK"] if pd.notnull(row["PK"]) else
0,
        "PenaltiesAttempted": row["PKatt"] if
pd.notnull(row["PKatt"]) else 0,
        "Assists": assists,
        "AssistPerMatch": round(assist_per_match, 2),
        "SubstituteAppearances": row["Subs"] if
pd.notnull(row["Subs"]) else 0,
    },
    "DefensiveStats": {
        "TacklesWon": row["TklW"] if pd.notnull(row["TklW"]) else
0,
        "Recoveries": row["Recov"] if pd.notnull(row["Recov"])
else 0,
        "AerialDuels": {
            "Won": aerial_duels_won,
            "Lost": aerial_duels_lost,
            "SuccessRate": round(aerial_duels_success_rate, 2),
        },
    },
    "DisciplinaryStats": {
        "YellowCards": row["CrDY"] if pd.notnull(row["CrDY"]) else
0,
        "SecondYellowCards": row["2CrDY"] if
pd.notnull(row["2CrDY"]) else 0,
        "RedCards": row["CrDR"] if pd.notnull(row["CrDR"]) else 0,
    },
    "SpecialEvents": {
        "Crosses": row["Crs"] if pd.notnull(row["Crs"]) else 0,
        "OwnGoals": row["OG"] if pd.notnull(row["OG"]) else 0,
        "Penalties": {
            "Won": row["PKwon"] if pd.notnull(row["PKwon"]) else
0,
            "Conceded": row["PKcon"] if pd.notnull(row["PKcon"])
else 0,
        },
    },
    "AdvancedStats": {
        "MinutesPlayedRatio": round(minutes_played_ratio, 2),
    },
}

csv_file = './data/dataset.csv'
data = pd.read_csv(csv_file)
data_subset = data.head(100)
documents = [transform_row(row) for _, row in data_subset.iterrows()]

try:

```

```
client = MongoClient(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/')
db = client['football']
collection = db['players']

collection.create_index("PersonalInfo.EstimatedStartYear")
collection.create_index("Player")
collection.create_index("PersonalInfo.SquadInfo.Squad")
collection.create_index("PersonalInfo.Nation")

collection.drop()
log.info(f"Collection {collection.name} dropped")
result = collection.insert_many(documents)
log.info(f"Inserted {len(result.inserted_ids)} record(s) to
mongo")

# Inserts a player from Manchester City
manchester_players = data[data['Squad'].str.contains('Manchester',
na=False, case=False)]
if not manchester_players.empty:
    first_manchester_player_row = manchester_players.iloc[0]
    first_manchester_player =
transform_row(first_manchester_player_row.to_dict())
    collection.insert_one(first_manchester_player)
    log.info(f"Inserted player
{first_manchester_player['PersonalInfo']['Player']} from Manchester.")
else:
    log.info("No players found with Squad containing
'Manchester'.")

except Exception as e:
    log.error("Could not connect to mongo: " + str(e))
```

6.2 MODIFYREGISTRIES.PY

```
import os
import time
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv
import logging as log

load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
UNAME = quote_plus(os.getenv("USR"))
UPASS = quote_plus(os.getenv("PWD"))

try:
    client =
MongoClient(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/')
    db = client['football']
    collection = db['players']

    # UPDATE PLAYER Teresa Abilleira

    name1 = "Teresa Abilleira"
    start_time = time.time()
    collection.update_one(
        {"Player": name1},
        {"$set": {"Player": name1.upper()}}
    )
    end_time = time.time()
    execution_time1 = end_time - start_time
    print(f"Actualización de {name1} ejecutada en
{execution_time1:.6f} segundos.")
    input("Presiona enter para ejecutar la segunda modificación")

    # UPDATE PLAYER Jessica Aby

    name2 = "Jessica Aby"
    start_time = time.time()
    collection.update_one(
        {"Player": name2},
        {"$set": {"Player": name2.upper()}}
    )
    end_time = time.time()
    execution_time2 = end_time - start_time
    print(f"Actualización de {name2} ejecutada en
{execution_time2:.6f} segundos.")

except Exception as e:
    log.error("Could not modify registries: " + str(e))
    print("Error durante las actualizaciones:", str(e))
```

6.3 QUERIES.PY

```
import time, os, json
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv

# Cargar variables de entorno
load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
USERNAME = quote_plus(os.getenv("USR"))
UPASS = quote_plus(os.getenv("PWD"))

# Conectar a MongoDB
client = MongoClient(f'mongodb://{USERNAME}:{UPASS}@{SERVER}:{PORT}/')
db = client['football']
collection = db['players']

# Función para medir el tiempo de ejecución de una consulta
def measure_query_time(collection, query):
    start_time = time.time()
    result = list(collection.find(query))
    end_time = time.time()
    elapsed_time = end_time - start_time
    return elapsed_time, result

# Función para ejecutar y almacenar los resultados de una consulta
def execute_and_store_results(query_name, collection, query):
    elapsed_time, result = measure_query_time(collection, query)
    print(f"Resultado {query_name}:")
    for doc in result:
        print(doc)
    print(f"Consulta {query_name} ejecutada en {elapsed_time:.6f} segundos")
    return {
        "query_name": query_name,
        "execution_time": elapsed_time,
        "results": result
    }

# Ejecutar consultas y almacenar resultados para almacenarlos en JSON
results_data = []

# Consulta a.i
query_ai = {"PersonalInfo.EstimatedStartYear": {"$gt": 2020}}
result_ai = execute_and_store_results("a.i", collection, query_ai)
results_data.append(result_ai)
input("Presiona enter para ejecutar la consulta a.ii")

# Consulta a.ii
query_aii = {"PersonalInfo.SquadInfo.Squad": {"$regex": "^Manchester",
"$options": "i"}}
result_aii = execute_and_store_results("a.ii", collection, query_aii)
```

```
results_data.append(result_ain)
input("Presiona enter para ejecutar la consulta b")

# Consulta b
query_b = {"PersonalInfo.Nation": "es ESP"}
result_b = execute_and_store_results("b", collection, query_b)
results_data.append(result_b)

# Guardar resultados de cada consulta en archivos JSON separados
for result in results_data:
    query_name = result["query_name"]
    output_file = f"results/query_results_{query_name}.json"
    with open(output_file, "w", encoding="utf-8") as json_file:
        json.dump(result, json_file, default=str, indent=4)
    print(f"Resultados de la consulta {query_name} guardados en
{output_file}")
```

7. BIBLIOGRAFÍA

- [1] Se ha utilizado ChatGPT para mejorar la estructura de los párrafos y la calidad del texto en general. No se ha usado para la generación de texto.

Prompt: Este párrafo puede necesitar de correcciones en las estructuras de las frases y correcciones en la gramática y en la fluidez, realiza los cambios oportunos: {PÁRRAFO}

- [2] Se ha utilizado ChatGPT para la generación de la función de `transform_row()` de `main.py` que maquetaba los datos, aunque no para la estructuración de los datos, que se ha realizado manualmente. También para averiguar el significado de algunas variables del dataset.

Prompt: Dado la siguiente estructura JSON, se pide generar una función en Python que transforme los datos procedentes de un data frame de pandas a la estructura provista para su posterior envío a MongoDB: {JSON}

Prompt: En este contexto, ¿qué significa esta variable del dataset? {DATASET}

- [3] Para la justificación de la base de datos, se ha utilizado el material provisto en ALUD.