

Fase 1

👤 Grupo 3
💻 Sistemas de Bases de Datos 2
🏛️ Universidad San Carlos de Guatemala
📅 Primer Semestre 2025

👥 Integrantes del Grupo:

Nombre	Carné
👤 Jose Andres Hinestrosa Garcia	202100316
👤 Joab Israel Ajsivinac Ajsivinac	202200135

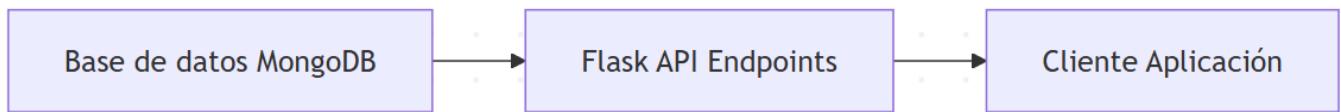
1. Resumen del Sistema

Esta API proporciona acceso a estadísticas completas de la NBA, incluyendo victorias y derrotas de equipos, puntuaciones de jugadores y análisis de equipos "víctimas" (equipos a los que un equipo específico ha derrotado frecuentemente).

El sistema utiliza una arquitectura de dos bases de datos:

- **Base de datos fuente (SQLite)**: Contiene los datos originales de la NBA.
- **Base de datos destino (MongoDB)**: Almacena los datos procesados para consultas optimizadas.

2. Arquitectura



3. Configuración y Conexión

Base URL

```
https://127.0.0.1:5000
```

Estructura de Archivos

- **app.py**: Servidor Flask con endpoints de la API
- **db_manager.py**: Gestor de conexiones y consultas a bases de datos

Bases de Datos

- **SQLite**: ruta

- **MongoDB:** Servidor local en `mongodb://localhost:27017/`
 - Base de datos: `nba_database`
 - Colecciones: `team_wins`, `team_losses`, `player_scores`, `victims`, `games_details`

4. Código Principal

4.1. Estadísticas Completas de Equipos

`/team_stats [POST]`

Procesa y transfiere estadísticas completas de equipos desde SQLite a MongoDB, incluyendo información general, detalles administrativos, redes sociales y análisis de equipos "víctimas".

Descripción: Este endpoint extrae información completa de equipos de la NBA, combinando datos básicos con estadísticas de victorias y derrotas, información administrativa y datos de redes sociales. Además, incluye un análisis de los equipos "víctimas" (equipos contra los que han obtenido más victorias).

Proceso:

1. Obtiene datos combinados de equipos mediante la función `get_team_stats`
2. Calcula estadísticas adicionales como el porcentaje de victorias
3. Formatea los datos de equipos "víctimas" en una estructura adecuada
4. Estructura toda la información en formato JSON jerárquico
5. Almacena los datos en la colección `teams` de MongoDB

Estructura de datos procesados:

```
{
  "team_id": "BOS",
  "team_name": "Boston Celtics",
  "nickname": "Celtics",
  "abbreviation": "BOS",
  "stats": {
    "wins": 56,
    "losses": 26,
    "total_games": 82,
    "win_percentage": 68.29
  },
  "details": {
    "year Founded": 1946,
    "arena": "TD Garden",
    "arena_capacity": 18624,
    "head_coach": "Joe Mazzulla",
    "owner": "Boston Basketball Partners L.L.C. (Wyc Grousbeck)",
    "general_manager": "Brad Stevens",
    "dleague_affiliation": "Maine Celtics"
  },
  "social_media": {
    "facebook": "bostonceltics",
    "instagram": "celtics",
    "twitter": "celtics"
  }
}
```

```
},
"victims": [
  {
    "equipo_victima": "Charlotte Hornets",
    "victorias": 12
  },
  {
    "equipo_victima": "Toronto Raptors",
    "victorias": 10
  }
]
```

Respuesta exitosa:

```
{
  "message": "Datos de equipos cargados exitosamente en la colección teams",
  "count": 30
}
```

Respuesta de error:

```
{
  "error": "No se encontraron datos de equipos"
}
```

Ejemplo de uso:

```
curl -X POST https://127.0.0.1:5000/team_stats
```

The screenshot shows the HTTPie interface with a collection named "Incognito sp...". It contains several requests and a draft section. A specific POST request to "http://127.0.0.1:5000/api/load_games" is selected, showing its parameters, headers, and body. The response is a JSON object with a "message" key containing the value "Player scores data loaded successfully".

The screenshot shows the MongoDB Compass interface connected to "BD2". The "team_stats" collection is selected. A single document is expanded to show its fields and values. The document includes fields like nickname, abbreviation, stats (win, losses, total_games), win_percentage, details (year_founded, arena, arena_capacity, head_coach, owner, general_manager, dleague_affiliation), social_media (facebook, instagram, twitter), and victims (an array of objects detailing game results). The document structure is as follows:

```

4 nickname: "Celtics"
5 abbreviation: "BOS"
6 stats: Object
7 win: 3626
8 losses: 2581
9 total_games: 6207
10 win_percentage: 58.42
11 details: Object
12 year Founded: 1946
13 arena: "TD Garden"
14 arena Capacity: 18624
15 head_coach: "Joe Mazzulla"
16 owner: "Wyc Grousbeck"
17 general_manager: "Brad Stevens"
18 dleague_affiliation: "Maine Celtics"
19 social_media: Object
20 facebook: "https://www.facebook.com/bostonceltics"
21 instagram: "https://instagram.com/celtics"
22 twitter: "https://twitter.com/celtics"
23 victims: Array (72)
24 0: Object
25 equipo_victima: "New York Knicks"
26 victorias: 308
27 1: Object
28 equipo_victima: "Philadelphia 76ers"
29 victorias: 213
30 2: Object
31 equipo_victima: "Detroit Pistons"
32 victorias: 206
33 3: Object
34 equipo_victima: "Atlanta Hawks"

```

4.1 Función get_games_data

Esta función obtiene datos completos de juegos de la NBA de forma optimizada desde la base de datos SQLite.

```

def get_games_data(cursor):
    """Consulta para obtener los datos completos de juegos de forma optimizada."""
    # Consulta principal para datos de juegos
    main_query = """
        SELECT
            g.game_id, g.game_date, g.season_id, g.season_type,
            g.team_id_home, g.team_abbreviation_home, g.team_name_home,
            g.matchup_home, g.wl_home, g pts_home, g.plus_minus_home,
    
```

```

g.fgm_home, g.fga_home, g.fg_pct_home, g.fg3m_home, g.fg3a_home,
g.fg3_pct_home,
    g.ftm_home, g.fta_home, g.ft_pct_home, g.oreb_home, g.dreb_home, g.reb_home,
    g.ast_home, g.stl_home, g.blk_home, g.tov_home, g.pf_home,
    g.team_id_away, g.team_abbreviation_away, g.team_name_away,
    g.matchup_away, g.wl_away, g.pts_away, g.plus_minus_away,
    g.fgm_away, g.fga_away, g.fg_pct_away, g.fg3m_away, g.fg3a_away,
g.fg3_pct_away,
    g.ftm_away, g.fta_away, g.ft_pct_away, g.oreb_away, g.dreb_away, g.reb_away,
    g.ast_away, g.stl_away, g.blk_away, g.tov_away, g.pf_away,
    CASE WHEN g.wl_home = 'W' THEN g.team_name_home ELSE g.team_name_away END as
winner
FROM game g
"""

cursor.execute(main_query)
columns = [col[0] for col in cursor.description]
games = []
game_ids = []

# Crear diccionario de juegos primero
games_dict = {}
for row in cursor.fetchall():
    game_data = dict(zip(columns, row))
    game_id = game_data['game_id']
    games_dict[game_id] = game_data
    game_ids.append(game_id)

# Función auxiliar para obtener datos relacionados en lotes pequeños
def get_related_data(table_name):
    if not game_ids:
        return {}

    result = {}
    columns = None
    # Procesar en lotes de 500 para evitar "too many SQL variables"
    batch_size = 500

    for i in range(0, len(game_ids), batch_size):
        batch = game_ids[i:i+batch_size]
        placeholders = ','.join(['?'] * len(batch))
        query = f"SELECT * FROM {table_name} WHERE game_id IN
({placeholders})"
        cursor.execute(query, batch)

        if not cursor.description:
            continue

        if columns is None:
            columns = [col[0] for col in cursor.description]

        for row in cursor.fetchall():
            data = dict(zip(columns, row))
            result[data['game_id']] = data

```

```

    return result

# Obtener datos relacionados en lote
related_tables = ['line_score', 'game_summary', 'game_info', 'other_stats']
related_data = {table: get_related_data(table) for table in related_tables}

# Completar y estructurar los datos de juegos con la información relacionada
for game_id, game_data in games_dict.items():
    # Extraer el año de la temporada desde game_summary si está disponible
    season_year = None
    if game_id in related_data['game_summary'] and 'season' in related_data['game_summary'][game_id]:
        season_year = related_data['game_summary'][game_id]['season']

    structured_game = {
        'id': game_id,
        'date': game_data['game_date'],
        'season': {
            'id': game_data['season_id'],
            'type': game_data['season_type'],
            'year': season_year
        },
        'teams': {
            'home': {
                'id': game_data['team_id_home'],
                'abbreviation': game_data['team_abbreviation_home'],
                'name': game_data['team_name_home'],
                'matchup': game_data['matchup_home'],
                'result': game_data['wl_home'],
                'stats': {
                    'points': game_data['pts_home'],
                    'plus_minus': game_data['plus_minus_home'],
                    'shooting': {
                        'field_goals': {
                            'made': game_data['fgm_home'],
                            'attempted': game_data['fga_home'],
                            'percentage': game_data['fg_pct_home']
                        },
                        'three_points': {
                            'made': game_data['fg3m_home'],
                            'attempted': game_data['fg3a_home'],
                            'percentage': game_data['fg3_pct_home']
                        },
                        'free_throws': {
                            'made': game_data['ftm_home'],
                            'attempted': game_data['fta_home'],
                            'percentage': game_data['ft_pct_home']
                        }
                    },
                    'rebounds': {
                        'offensive': game_data['oreb_home'],
                        'defensive': game_data['dreb_home'],
                        'total': game_data['reb_home']
                    }
                }
            }
        }
    }

```

```

        },
        'assists': game_data['ast_home'],
        'steals': game_data['stl_home'],
        'blocks': game_data['blk_home'],
        'turnovers': game_data['tov_home'],
        'fouls': game_data['pf_home']
    }
},
'away': {
    'id': game_data['team_id_away'],
    'abbreviation': game_data['team_abbreviation_away'],
    'name': game_data['team_name_away'],
    'matchup': game_data['matchup_away'],
    'result': game_data['wl_away'],
    'stats': {
        'points': game_data['pts_away'],
        'plus_minus': game_data['plus_minus_away'],
        'shooting': {
            'field_goals': {
                'made': game_data['fgm_away'],
                'attempted': game_data['fga_away'],
                'percentage': game_data['fg_pct_away']
            },
            'three_points': {
                'made': game_data['fg3m_away'],
                'attempted': game_data['fg3a_away'],
                'percentage': game_data['fg3_pct_away']
            },
            'free_throws': {
                'made': game_data['ftm_away'],
                'attempted': game_data['fta_away'],
                'percentage': game_data['ft_pct_away']
            }
        },
        'rebounds': {
            'offensive': game_data['oreb_away'],
            'defensive': game_data['dreb_away'],
            'total': game_data['reb_away']
        },
        'assists': game_data['ast_away'],
        'steals': game_data['stl_away'],
        'blocks': game_data['blk_away'],
        'turnovers': game_data['tov_away'],
        'fouls': game_data['pf_away']
    }
},
'winner': game_data['winner'],
'details': {}
}

# Agregar los datos relacionados a la estructura
for table in related_tables:
    if game_id in related_data[table]:

```

```

        table_data = related_data[table][game_id]
        # Eliminar el game_id para evitar redundancia
        if 'game_id' in table_data:
            del table_data['game_id']
        structured_game['details'][table] = table_data

    games.append(structured_game)

return games

```

The screenshot shows the HTTPie interface. On the left, there's a sidebar with various collections and requests. In the main area, a POST request is being made to `http://127.0.0.1:5000/api/load_games`. The response is a JSON object with a count of 65642 and a message indicating successful game data loading.

```

Request POST Response 200
HTTP/1.1 200 OK (5 headers)
{
  "count": 65642,
  "message": "Game data loaded successfully"
}

```

The screenshot shows the MongoDB Compass interface. It's connected to the `nba_database` and is viewing the `games_details` collection. A single document is expanded to show its structure, which includes fields like `_id`, `date`, `season`, `teams`, `stats`, `details`, `line_score`, `game_summary`, and `game_info`.

4.2 Endpoint para cargar datos de juegos

```

@app.route('/api/load_games', methods=['POST'])
def load_full_games():
    try:
        # Obtener los datos de juegos usando la función del db_manager
        results = get_games_data(app.config['SQLITE_CURSOR'])

        if not results:
            return jsonify({'error': 'No games data found'}), 404

    except Exception as e:
        return jsonify({'error': str(e)}), 500

    try:
        # Insertar los resultados en MongoDB
        games_collection = app.config['MONGO_DB']['games_details']

        # Opcional: limpiar datos existentes si se solicita
        force = request.args.get('force', '').lower() == 'true'
        if force:
            games_collection.delete_many({})

        # Insertar datos usando tu función existente
        insert_data_to_mongo(games_collection, results)

    except Exception as e:
        return jsonify({'error': str(e)}), 500

    return jsonify({'message': 'Game data loaded successfully', 'count': len(results)}), 200

```

5. Endpoints de la API

5.1. Estadísticas de Victorias de Equipos

```

[{"team_id": "1610612747", "team_name": "Los Angeles Lakers", "wins": 3641}, {"team_id": "1610612738", "team_name": "Boston Celtics", "wins": 3626}, {"team_id": "1610612755", "team_name": "Philadelphia 76ers", "wins": 3000}, {"team_id": "1610612752", "team_name": "New York Knicks", "wins": 2863}, {"team_id": "1610612744", "team_name": "Warriors", "wins": 2443}

```

The screenshot shows the Thunder Client interface with a successful GET request to `/team_wins`. The response body contains a JSON array of five NBA team documents, each with `team_id`, `team_name`, and `wins` fields. The interface also displays a terminal window showing log entries related to the application's startup and configuration.

/team_wins [GET]

Obtiene las estadísticas de victorias de todos los equipos desde MongoDB.

Respuesta:

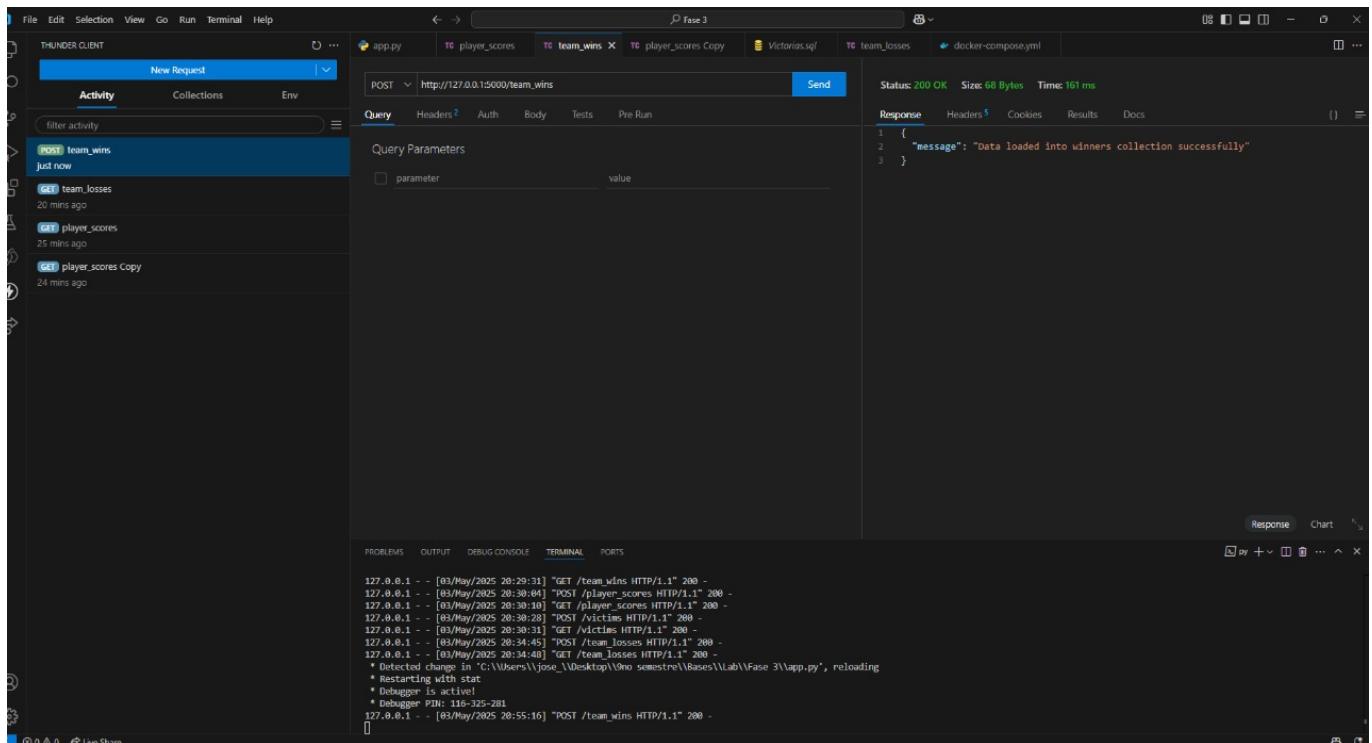
```
[
  {
    "team_id": "BOS",
    "team_name": "Boston Celtics",
    "nickname": "Celtics",
    "wins": 56
  },
  {
    "team_id": "LAL",
    "team_name": "Los Angeles Lakers",
    "nickname": "Lakers",
    "wins": 52
  }
]
```

Ejemplo de uso:

```
curl -X GET https://127.0.0.1:5000/team_wins
```

/team_wins [POST]

Procesa y transfiere datos de victorias desde SQLite a MongoDB.



Consulta SQL utilizada:

```

SELECT
    t.id AS id,
    t.full_name AS nombre_equipo,
    t.nickname AS nick_name,
    COUNT(*) AS wins
FROM (
    SELECT
        CASE
            WHEN pts_home > pts_away THEN team_id_home
            WHEN pts_away > pts_home THEN team_id_away
            ELSE NULL
        END AS team_name
    FROM game
) victorias
INNER JOIN team t ON t.id = victorias.team_name
GROUP BY t.id, t.full_name, t.nickname
ORDER BY wins DESC;

```

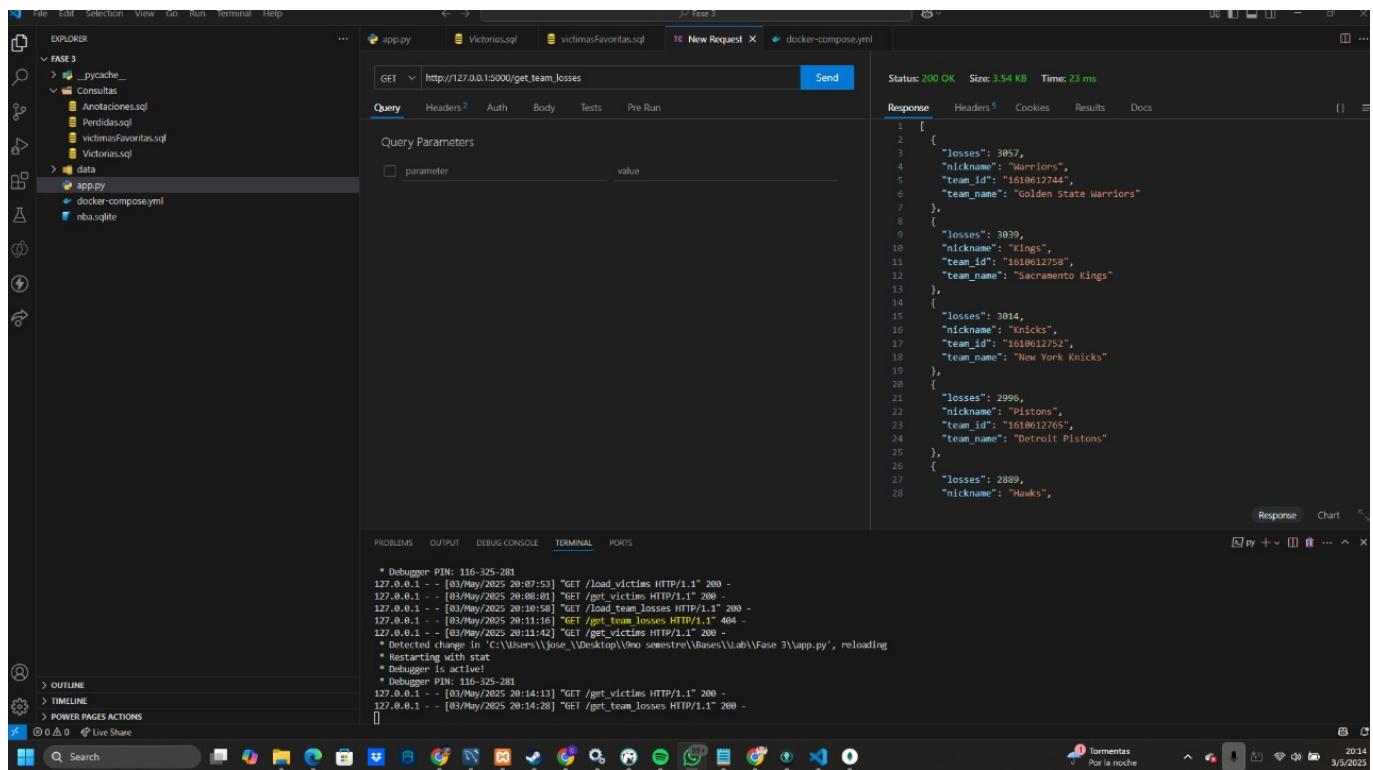
Respuesta exitosa:

```
{
    "message": "Data loaded into winners collection successfully"
}
```

Ejemplo de uso:

```
curl -X POST https://127.0.0.1:5000/team_wins
```

5.2. Estadísticas de Derrotas de Equipos



/team_losses [GET]

Obtiene las estadísticas de derrotas de todos los equipos desde MongoDB.

Respuesta:

```
[  
 {  
   "team_id": "BOS",  
   "team_name": "Boston Celtics",  
   "nickname": "Celtics",  
   "losses": 26  
 },  
 {  
   "team_id": "NYK",  
   "team_name": "New York Knicks",  
   "nickname": "Knicks",  
   "losses": 32  
 }]
```

Ejemplo de uso:

```
curl -X GET https://127.0.0.1:5000/team_losses
```

The screenshot shows the Postman interface. In the top navigation bar, there are tabs for 'app.py', 'Victorias.sql', 'victimasFavoritas.sql', 'New Request' (which is active), and 'docker-compose.yml'. Below the tabs, there's a search bar with 'http://127.0.0.1:5000/load_team_losses' and a 'Send' button. The main area has tabs for 'Query', 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run'. Under 'Query', there's a table for 'Query Parameters' with one entry: 'parameter' and 'value'. On the right side, under 'Response', it shows 'Status: 200 OK', 'Size: 67 Bytes', and 'Time: 113 ms'. The response body is a JSON object:

```
1  {
2      "message": "Data loaded into losers collection successfully"
3  }
```

/team_losses [POST]

Procesa y transfiere datos de derrotas desde SQLite a MongoDB.

Consulta SQL utilizada:

```
SELECT
    t.id AS id,
    t.full_name AS nombre_equipo,
    t.nickname AS nick_name,
    COUNT(*) AS losses
FROM (
    SELECT
        CASE
            WHEN pts_home < pts_away THEN team_id_home
            WHEN pts_away < pts_home THEN team_id_away
            ELSE NULL
        END AS team_name
    FROM game
) victorias
INNER JOIN team t ON t.id = victorias.team_name
GROUP BY t.id, t.full_name, t.nickname
ORDER BY losses DESC;
```

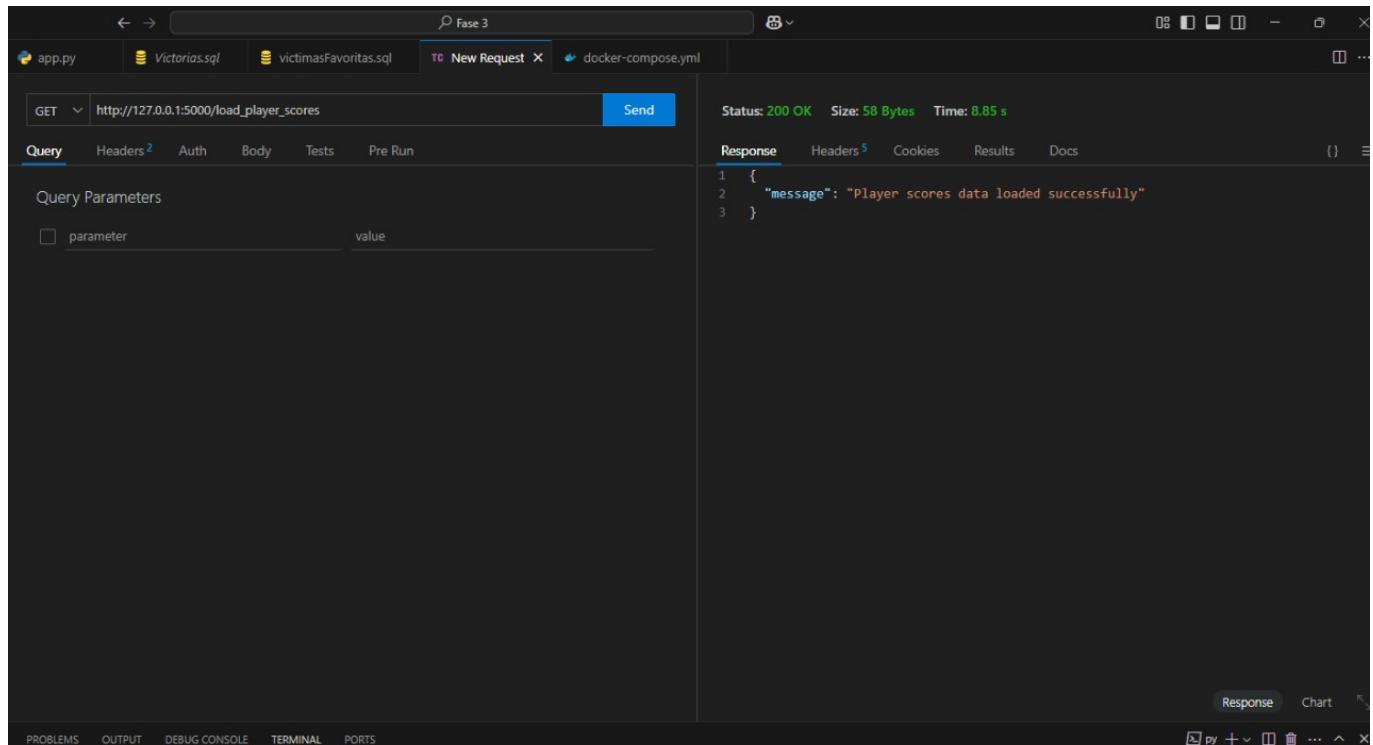
Respuesta exitosa:

```
{
    "message": "Data loaded into losers collection successfully"
}
```

Ejemplo de uso:

```
curl -X POST https://127.0.0.1:5000/team_losses
```

5.3. Estadísticas de Jugadores



The screenshot shows the Postman interface with a successful API call. The request URL is `http://127.0.0.1:5000/load_player_scores`. The response status is `200 OK`, size is `58 Bytes`, and time is `8.85 s`. The response body is a JSON object with a single key `message` containing the value `"Player scores data loaded successfully"`.

```
rv = self.handle_user_exception(e)
File "C:\Users\jose\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 917, in full_dispatch_request
    rv = self.dispatch_request()
File "C:\Users\jose\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 902, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
File "C:\Users\jose\Desktop\9no semestre\Bases\Lab\Fase 3\app.py", line 263, in get_team_wins
    'nickname': item['nickname'],
    ~~~~~^~~~~~
KeyError: 'nickname'
127.0.0.1 - - [03/May/2025 20:16:22] "GET /load_team_wins HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2025 20:17:50] "GET /load_player_scores HTTP/1.1" 200 -

```

/player_scores [GET]

Obtiene las estadísticas de puntuación de todos los jugadores desde MongoDB.

Respuesta:

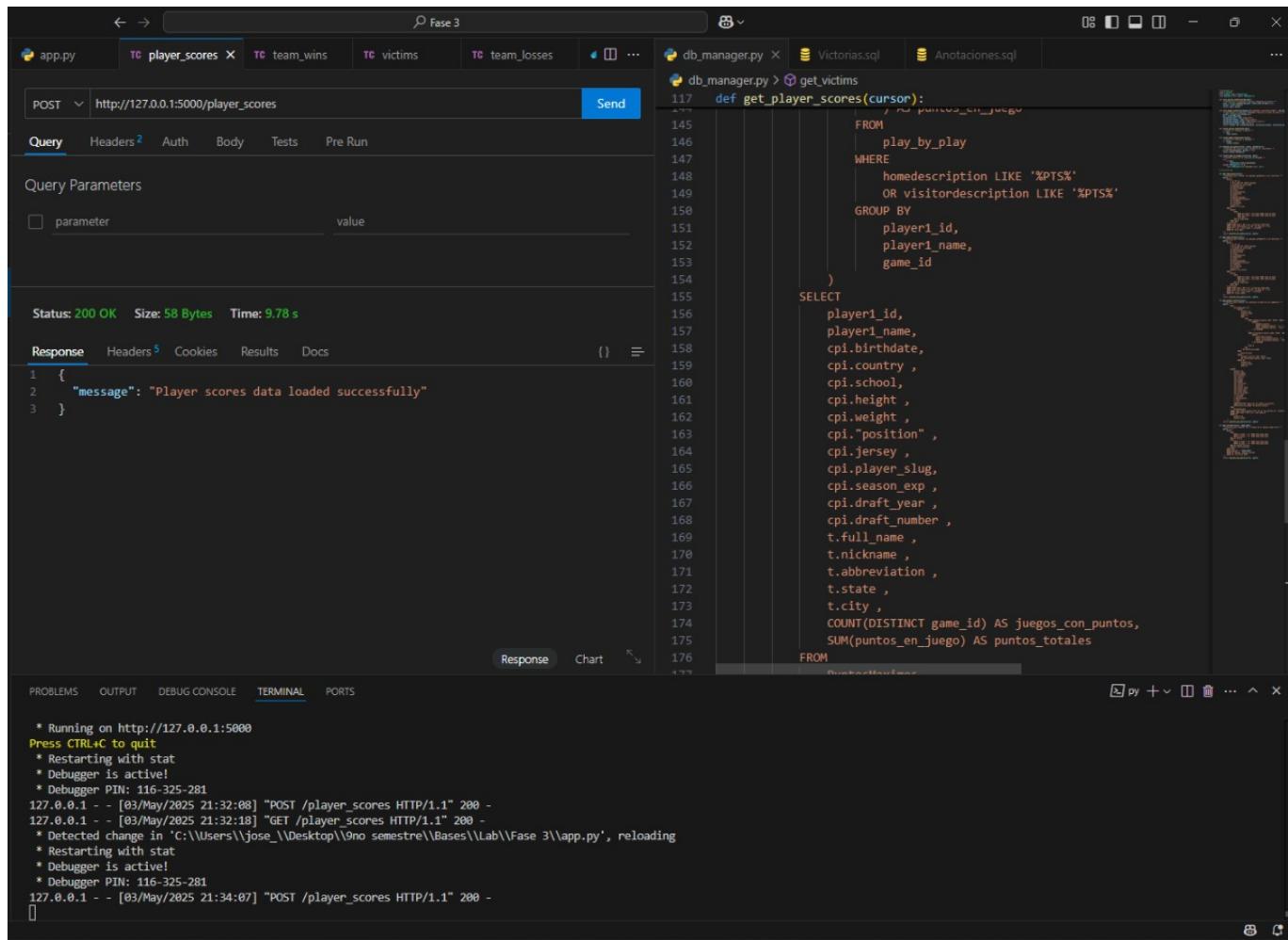
```
[  
  {  
    "player_id": "jamesle01",  
    "player_name": "LeBron James",  
    "games_with_points": 82,  
    "total_points": 2251  
  },  
  {  
    "player_id": "duranke01",  
    "player_name": "Kevin Durant",  
    "games_with_points": 78,
```

```

        "total_points": 2118
    }
]
```

Ejemplo de uso:

```
curl -X GET https://127.0.0.1:5000/player_scores
```



The screenshot shows a code editor interface with several tabs open. On the left, there's a tab for 'app.py' which contains a POST method for '/player_scores'. The body of the POST method is a JSON object with a single key 'message': "Player scores data loaded successfully". To the right of this, there's a tab for 'db_manager.py' containing an SQL query named 'get_player_scores'. This query joins multiple tables: 'play_by_play', 'player1', 'cpi', and 't'. It uses various functions like COUNT(DISTINCT) and SUM to calculate statistics such as 'juegos_con_puntos' and 'puntos_totales'. Below the code editor, a terminal window shows the Flask application running on port 127.0.0.1:5000, with logs indicating successful POST requests to '/player_scores'.

```

POST http://127.0.0.1:5000/player_scores
Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters
parameter value

Status: 200 OK Size: 58 Bytes Time: 9.78 s

Response Headers Cookies Results Docs
1 {
2   "message": "Player scores data loaded successfully"
3 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 116-325-281
127.0.0.1 - - [03/May/2025 21:32:08] "POST /player_scores HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2025 21:32:18] "GET /player_scores HTTP/1.1" 200 -
* Detected change in 'C:\\Users\\jose_\\Desktop\\9no semestre\\Bases\\Lab\\Fase 3\\app.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger PIN: 116-325-281
127.0.0.1 - - [03/May/2025 21:34:07] "POST /player_scores HTTP/1.1" 200 -

```

/player_scores [POST]

Procesa y transfiere datos de puntuaciones de jugadores desde SQLite a MongoDB.

Consulta SQL utilizada: La consulta analiza los registros de play-by-play para extraer la información de puntos por jugador por juego y calcular estadísticas agregadas.

Respuesta exitosa:

```
{
  "message": "Player scores data loaded successfully"
}
```

5.4. Datos de Juegos Completos

/api/load_games [POST]

Procesa y transfiere datos detallados de juegos desde SQLite a MongoDB.

Descripción: Este endpoint extrae información completa de juegos de la NBA, incluyendo estadísticas de equipos locales y visitantes, resultados de partidos, y datos relacionados como line_score, game_summary, game_info y other_stats.

Proceso:

1. Consulta principal a la tabla `game` para obtener datos básicos
2. Procesamiento en lotes de 500 juegos para evitar la limitación "too many SQL variables"
3. Recopilación de datos relacionados desde tablas auxiliares
4. Estructuración de datos en formato JSON jerárquico
5. Almacenamiento en la colección `games_details` de MongoDB

Parámetros:

- `force` (opcional): Si se establece como "true", elimina registros existentes antes de insertar

Respuesta exitosa:

```
{  
  "message": "Game data loaded successfully",  
  "count": 1230  
}
```

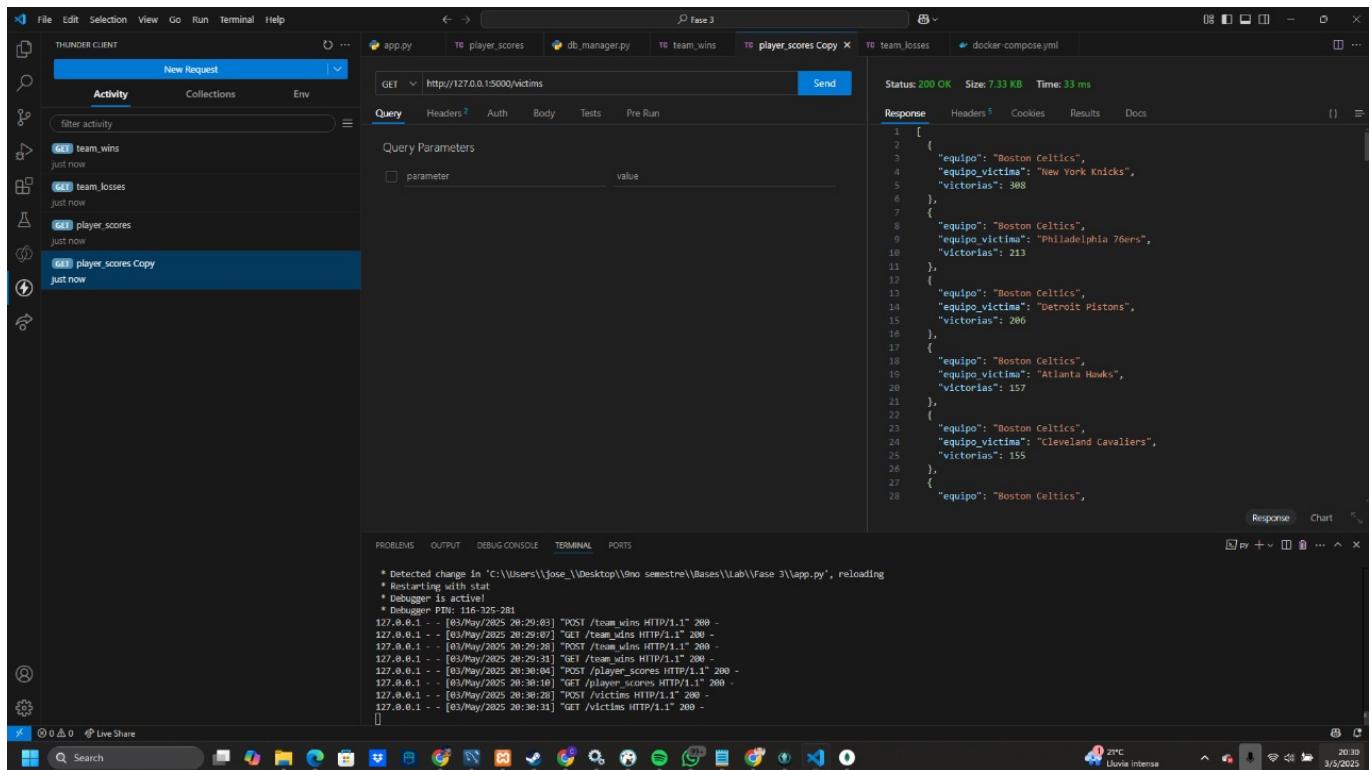
Ejemplo de uso:

```
curl -X POST https://127.0.0.1:5000/api/load_games?force=true
```

Ejemplo de uso sin forzar la eliminación de datos existentes:

```
curl -X POST https://127.0.0.1:5000/api/load_games
```

4.4. Análisis de Equipos "Víctimas"



/victims [GET]

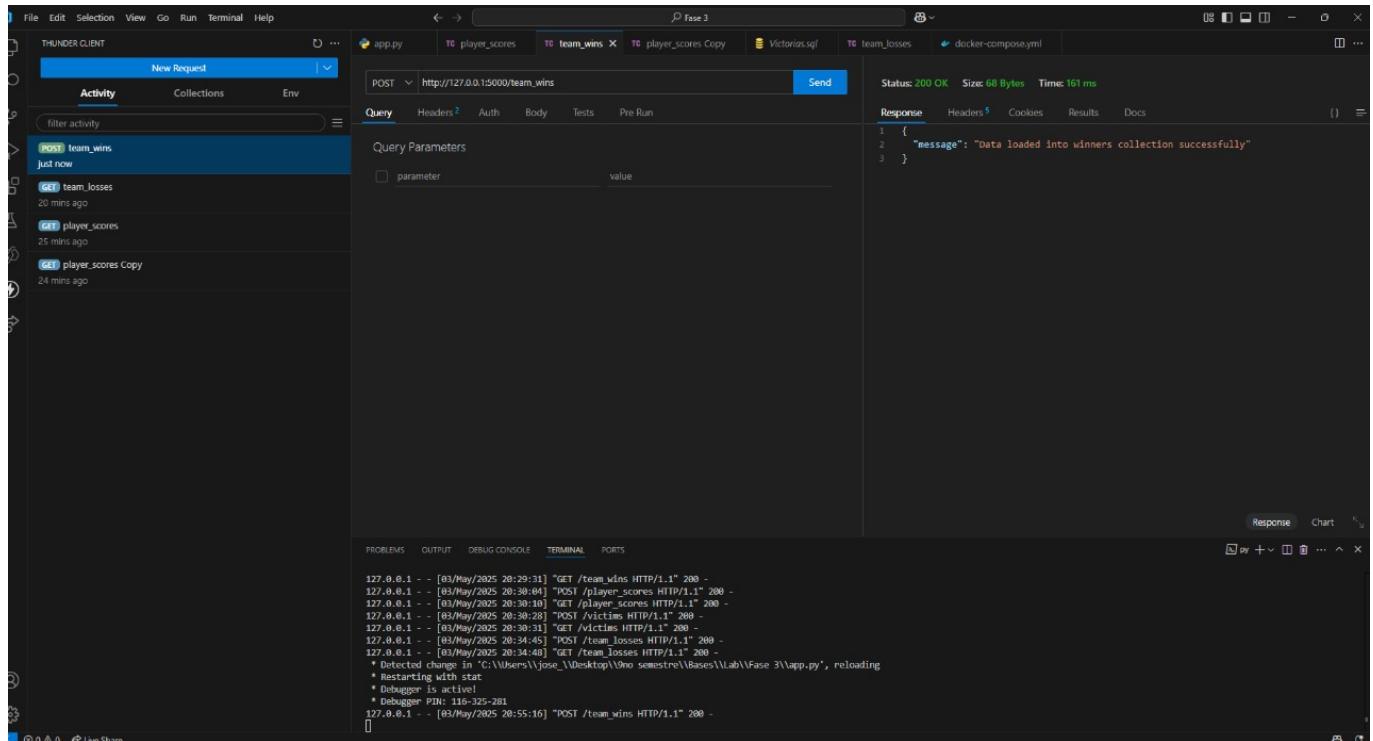
Obtiene el análisis de "víctimas" (equipos contra los que un equipo específico ha ganado) desde MongoDB.

Respuesta:

```
[
  {
    "equipo": "Boston Celtics",
    "equipo_victima": "Los Angeles Lakers",
    "victorias": 12
  },
  {
    "equipo": "Boston Celtics",
    "equipo_victima": "Chicago Bulls",
    "victorias": 8
  }
]
```

Ejemplo de uso:

```
curl -X GET https://127.0.0.1:5000/victims
```



/victims [POST]

Procesa y transfiere datos de "víctimas" para un equipo específico desde SQLite a MongoDB.

Parámetros de consulta:

- **team_name** (opcional): Nombre del equipo para analizar (por defecto: "Boston Celtics")

Consulta SQL utilizada:

```

SELECT
CASE
    WHEN wl_home = 'W' THEN team_name_home
    WHEN wl_away = 'W' THEN team_name_away
END AS equipo,
CASE
    WHEN wl_home = 'W' THEN team_name_away
    WHEN wl_away = 'W' THEN team_name_home
END AS equipo_victima,
COUNT(*) AS victorias
FROM game
WHERE equipo = '[team_name]'
GROUP BY equipo, equipo_victima
ORDER BY victorias DESC;

```

Respuesta exitosa:

```
{
  "message": "Data loaded into victims collection successfully"
}
```

Ejemplo de uso:

```
curl -X POST "https://127.0.0.1:5000/victims?team_name=Los%20Angeles%20Lakers"
```

5. Modelos de Datos

5.1. Colección team_wins

Campo	Tipo	Descripción
team_id	string	Identificador único del equipo
team_name	string	Nombre completo del equipo
nickname	string	Apodo o nombre corto del equipo
wins	integer	Total de victorias registradas

5.2. Colección team_losses

Campo	Tipo	Descripción
team_id	string	Identificador único del equipo
team_name	string	Nombre completo del equipo
nickname	string	Apodo o nombre corto del equipo
losses	integer	Total de derrotas registradas

5.3. Colección player_scores

Campo	Tipo	Descripción
player_id	string	Identificador único del jugador
player_name	string	Nombre completo del jugador
games_with_points	integer	Número de juegos en los que el jugador anotó puntos
total_points	integer	Total de puntos anotados

5.4. Colección victims

Campo	Tipo	Descripción
equipo	string	Nombre del equipo ganador
equipo_victima	string	Nombre del equipo derrotado ("victima")
victorias	integer	Número de veces que el equipo ha derrotado a esta "victima"

6. Códigos de Respuesta

Código	Estado	Descripción
200	OK	La solicitud ha tenido éxito
404	Not Found	No se encontraron datos para la consulta
500	Internal Server Error	Error del servidor al procesar la solicitud

7. Flujo de Trabajo Recomendado

7.1. Inicialización de Datos

- Ejecutar los endpoints POST para cargar datos en MongoDB:

```
curl -X POST https://127.0.0.1:5000/team_wins  
curl -X POST https://127.0.0.1:5000/team_losses  
curl -X POST https://127.0.0.1:5000/player_scores  
curl -X POST "https://127.0.0.1:5000/victims?team_name=Boston%20Celtics"
```

7.2. Consulta de Datos

- Utilizar los endpoints GET para obtener información:

```
curl -X GET https://127.0.0.1:5000/team_wins  
curl -X GET https://127.0.0.1:5000/team_losses  
curl -X GET https://127.0.0.1:5000/player_scores  
curl -X GET https://127.0.0.1:5000/victims
```

8. Consideraciones Técnicas

8.1. Conexiones a Bases de Datos

- Las conexiones a SQLite y MongoDB se inicializan al inicio de cada solicitud.
- Las conexiones se cierran automáticamente al finalizar cada solicitud.
- Se implementa la gestión adecuada de errores para problemas de conexión.

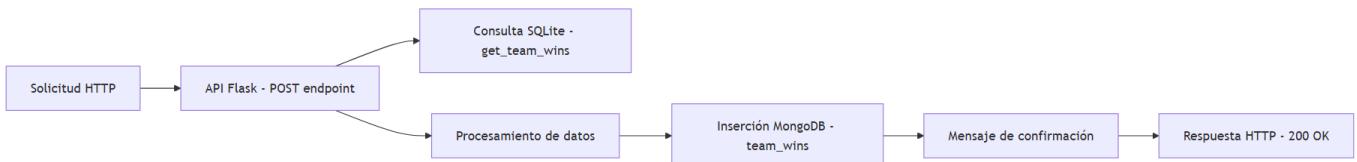
8.2. Consultas SQL

- Las consultas SQL están optimizadas para el rendimiento.
- Se utilizan subconsultas y funciones de agregación para procesar datos complejos.
- Todas las consultas son parametrizadas para prevenir inyecciones SQL.

8.3. Operaciones MongoDB

- Se utilizan operaciones bulk para inserciones masivas de datos.
- Las colecciones están estructuradas para optimizar las consultas habituales.

9. Diagrama de Flujo de Datos



10. Historial de Cambios

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar titled 'Activity' with a log of recent operations like 'POST team_wins' and 'POST team_losses'. The main area is titled 'MongoDB Compass - Fase3/nba_database.team_losses' and shows the 'team_losses' collection. It lists two documents:

```
_id: ObjectId('610ddadbb6c611eb9a0ebf03')
team_id: "1610612744"
team_name: "Golden State Warriors"
nickname: "Warriors"
abbreviation: "GSW"
year_founded: 1946
arena: "Chase Center"
arena_capacity: null
head_coach: "Steve Kerr"
owner: "Joe Lacob"
general_manager: "Bob Myers"
league: "Santa Cruz Warriors"
facebook: "https://www.facebook.com/warriors"
instagram: "https://Instagram.com/warriors"
twitter: "https://Twitter.com/warriors"
losses: "GSW"

_id: ObjectId('610ddadbb6c611eb9a0ebf04')
team_id: "1610612758"
team_name: "Sacramento Kings"
nickname: "Kings"
abbreviation: "SAC"
year_founded: 1948
arena: "Golden 1 Center"
arena_capacity: 17500
head_coach: "Mike Brown"
owner: "Viviek Ranadive"
general_manager: "Monte McNair"
league: "Stockton Kings"
```

At the bottom, a status bar shows command history and a message about a detected change in the database.

- se agrega mas información team_losses

The screenshot shows a POST request to 'http://127.0.0.1:5000/team_losses' in Postman. The response status is 500 INTERNAL SERVER ERROR. The response body is:

```
1  {
2     "error": "no such column: losses"
3 }
```

On the right, the code editor shows the Python script 'db_manager.py' with the function 'get_team_losses(cursor)'. The code is a complex SQL query using CASE statements to calculate team names based on win/loss records.

```
def get_team_losses(cursor):
    """
    Consulta para obtener los equipos perdedores y sus derrotas."""
    query = """
        SELECT
            t.id AS id,
            t.full_name AS nombre_equipo,
            t.nickname AS nick_name,
            td.abbreviation,
            td.yearfounded,
            td.arena,
            td.arenacapacity,
            td.headcoach,
            td.owner,
            td.generalmanager,
            td.leagueaffiliation,
            td.facebook,
            td.instagram,
            td.twitter,
            COUNT(*) AS wins
        FROM (
            SELECT
                CASE
                    WHEN pts_home < pts_away THEN team_id_home
                    WHEN pts_away < pts_home THEN team_id_away
                    ELSE NULL
                END AS team_name
            FROM game
        ) victorias
        INNER JOIN team t ON t.id = victorias.team_name
        INNER JOIN team details td ON td.team_id = t.id
    """

    cursor.execute(query)
    results = cursor.fetchall()
    return results
```

- se agrega mas información al endpoint de team_losses

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar titled "THUNDER CLIENT" with a "New Request" button and a "filter activity" search bar. Below it, a list of recent requests includes:

- POST** team_wins just now
- GET** team_losses 32 mins ago
- GET** player_scores 37 mins ago
- GET** victims 37 mins ago

The main area is titled "MongoDB Compass - Fase3/nba_database.team_wins". It shows a tree view of connections and collections. Under "nba_database", the "team_wins" collection is selected, displaying 27 documents. A specific document is expanded, showing fields like _id, team_id, team_name, nickname, abbreviation, year_founded, arena, arena_capacity, head_coach, owner, general_manager, dleague, facebook, instagram, and twitter. The document ID is `_id: ObjectId('6816d1a0927cb7a9716ac2b')`. The document content is as follows:

```
_id: ObjectId('6816d1a0927cb7a9716ac2b')
team_id: "1610612747"
team_name: "Los Angeles Lakers"
nickname: "Lakers"
abbreviation: "LAL"
year_founded: 1948
arena: "Crypto.com Arena"
arena_capacity: 19660
head_coach: "Darvin Ham"
owner: "Jeanie Buss"
general_manager: "Rob Pelinka"
dleague: "South Bay Lakers"
facebook: "https://www.facebook.com/losangeleslakers"
instagram: "https://Instagram.com/lakers"
twitter: "https://Twitter.com/Lakers"
wins: 3641
```

A warning message at the bottom states: "WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead."

At the bottom, a terminal window shows the command line:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
  Debugger PIN: 110-325-281
127.0.0.1 - [03/May/2025 21:07:43] "GET /team_wins HTTP/1.1" 500 -
127.0.0.1 - [03/May/2025 21:08:10] "POST /team_wins HTTP/1.1" 200 -
```

- se agrega mas información team_wins