RM Scores Analysis – API Implementation

Name: Yatharth Singh Vikal

Date: 14-01-2026

1. Objective

The objective of this task is to calculate and expose Relationship Manager (RM) performance scores based on recorded evaluation data. The final output provides the top RMs ranked by average score, along with their best and worst scores, grouped by zone and region.

2. Data Sources

The solution uses the following PostgreSQL tables:

- **recorded_info**
  Contains RM evaluation data stored in JSON format (score_json).
- **user_master**
  Contains RM metadata including zone and region.

The tables are joined using the common key rm_id.

3. Approach

- PostgreSQL is used for data aggregation and JSON parsing.
- The final_score value is extracted from the JSON field using jsonb operators.
- Aggregate functions (AVG, MAX, MIN) are applied per RM.
- Results are grouped by zone, region, and RM ID.
- A FastAPI application exposes the result through a REST API endpoint.
- Database credentials are securely handled using environment variables.

4. SQL Query Logic

The following logic is applied in the SQL query:

- Extract final_score from the JSON structure in score_json
- Convert the extracted value to numeric
- Compute:
    - Average RM score
    - Best (maximum) score
    - Worst (minimum) score

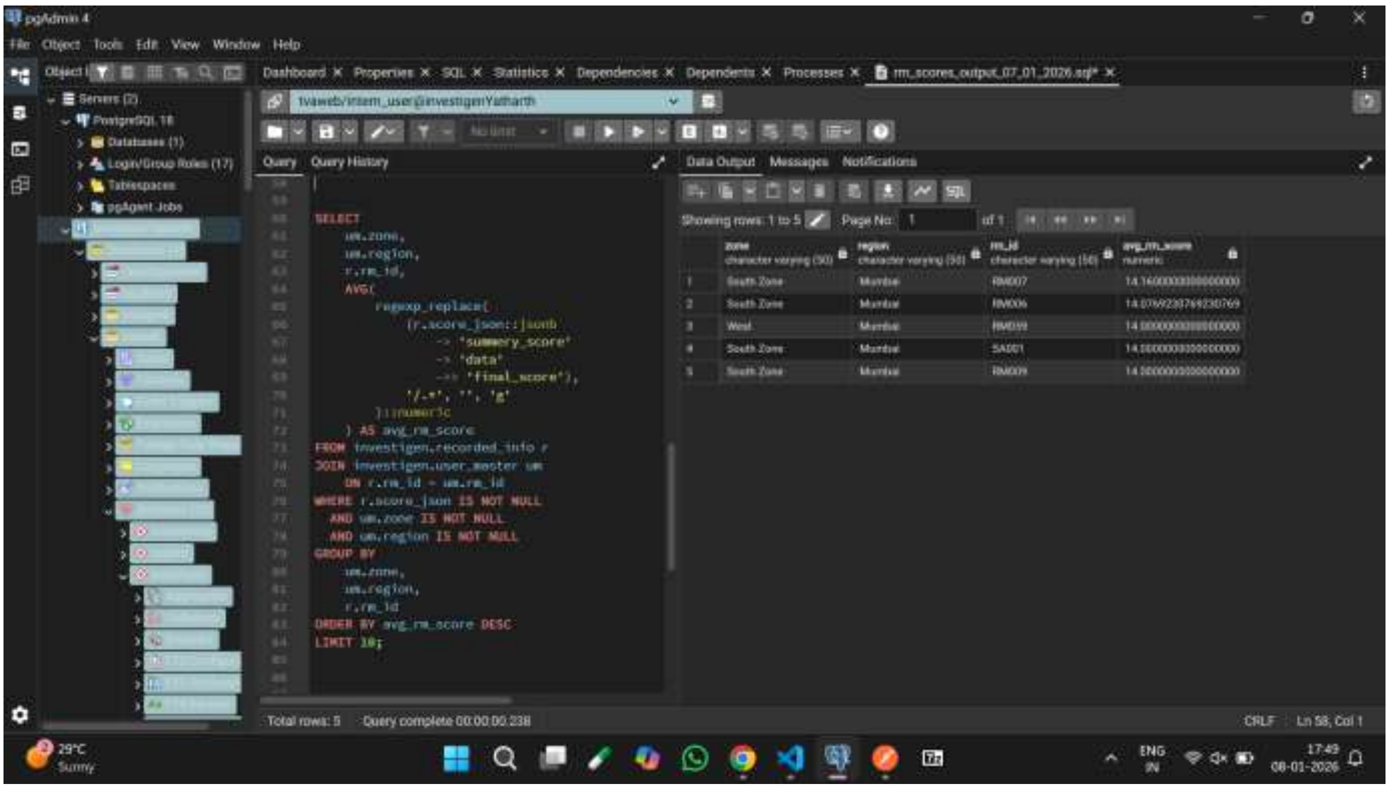
(Final SQL query is provided below)

SQL Query :

```
SELECT
   um.zone,
   um.region,
   r.rm_id,
   AVG(
      (r.score_json::jsonb
         -> 'summery_score'
         -> 'data'
         ->> 'final_score'
      )::numeric
   ) AS avg_rm_score
FROM investigen.recorded_info r
JOIN investigen.user_master um
   ON r.rm_id = um.rm_id
WHERE r.score_json IS NOT NULL
```

```
  AND um.zone IS NOT NULL
  AND um.region IS NOT NULL
GROUP BY
    um.zone,
    um.region,
    r.rm_id
ORDER BY avg_rm_score DESC
LIMIT 10;
```

<u>SQL Query screenshot (in pgAdmin4):</u>



```
SELECT
    um.zone,
    um.region,
    r.rm_id,
    AVG(
        regexp_replace(
            (r.score_json::jsonb
                -> 'summery_score'
                -> 'data'
                ->> 'final_score'),
            '/.*', '', 'g'
        )::numeric
    ) AS avg_rm_score
FROM investigen.recorded_info r
JOIN investigen.user_master um
    ON r.rm_id = um.rm_id
WHERE r.score_json IS NOT NULL
  AND um.zone IS NOT NULL
  AND um.region IS NOT NULL
GROUP BY
    um.zone,
    um.region,
    r.rm_id
ORDER BY avg_rm_score DESC
LIMIT 10;
```

Showing rows: 1 to 5

	zone character varying (50)	region character varying (50)	rm_id character varying (50)	avg_rm_score numeric
1	South Zone	Mumbai	RM007	14.1600000000000000
2	South Zone	Mumbai	RM006	14.0769230769230769
3	West	Mumbai	RM039	14.0000000000000000
4	South Zone	Mumbai	SA001	14.0000000000000000
5	South Zone	Mumbai	RM009	14.0000000000000000

Total rows: 5   Query complete 00:00:00.238

5. API Implementation

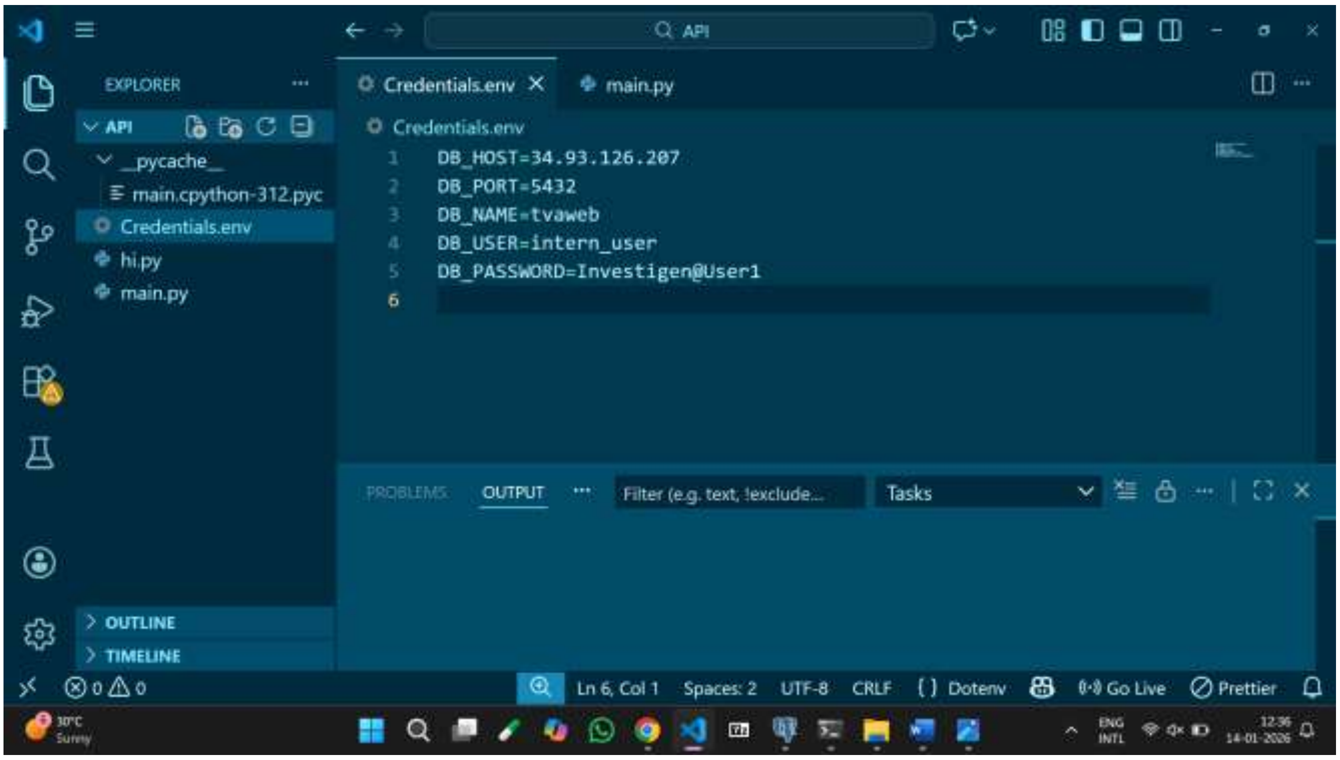
- Framework: **FastAPI**
- Database Driver: **psycopg2**
- Output Format: **JSON**

- **Endpoint: GET /rm-scores**

The API establishes a database connection, executes the query, and returns the aggregated RM score data.

6. Security Considerations

- Database credentials are not hardcoded.
- Credentials are loaded from environment variables using a .env file.
- This approach ensures sensitive information is not exposed in the source code or repository.



```
DB_HOST=34.93.126.207
DB_PORT=5432
DB_NAME=tvaweb
DB_USER=intern_user
DB_PASSWORD=Investigen@User1
```

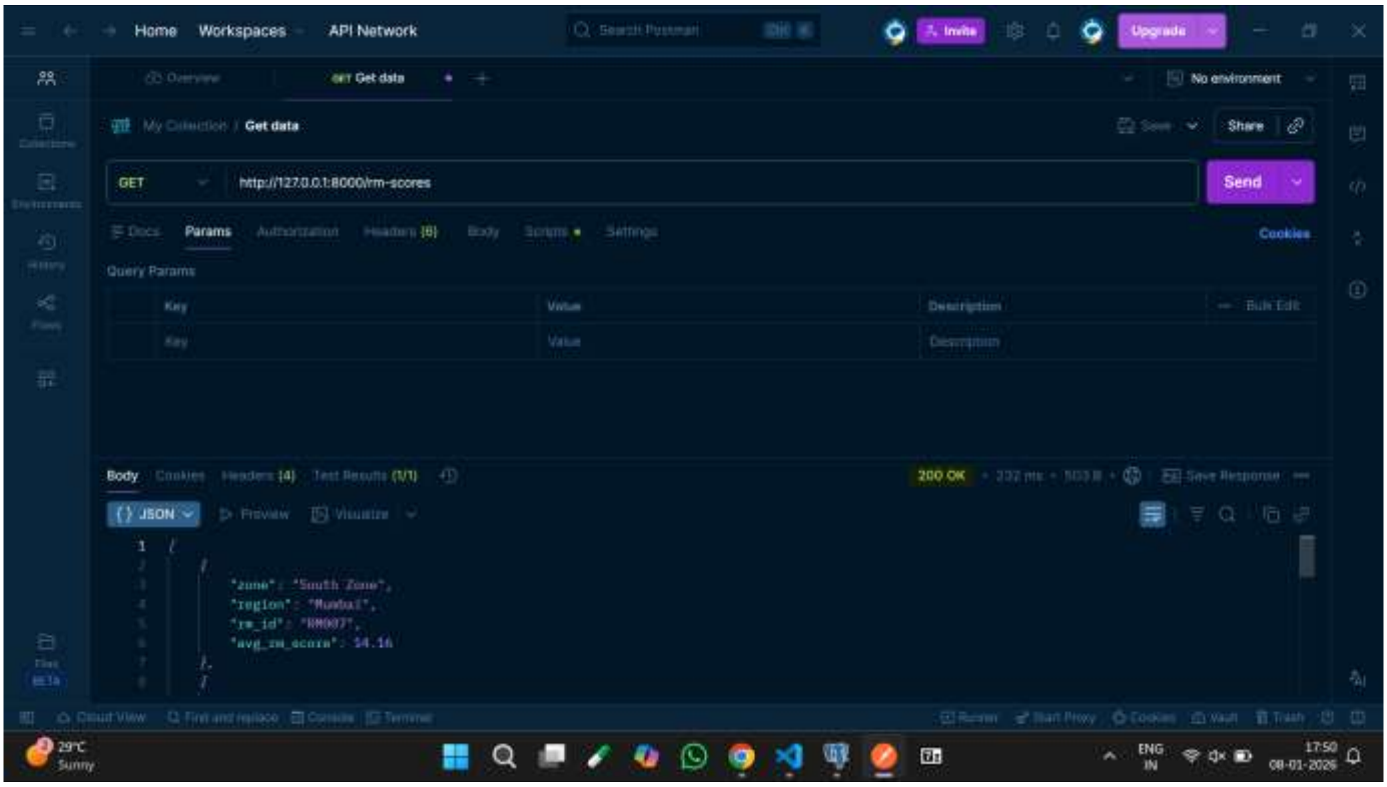
7. Output and Results

The API returns the following fields:

- Zone

- Region
- RM ID
- Average RM Score.

<u>**API Response screenshot :**</u>



http://127.0.0.1:8000/rm-scores

```
[
  {
    "zone": "South Zone",
    "region": "Mumbai",
    "rm_id": "RM007",
    "avg_rm_score": 54.16
  },
]
```

8. How to Run the Application:

1. Set database credentials using environment variables.
2. Install required dependencies.
3. Start the server:

   uvicorn main:app --reload

4. Access the API at:

   URL : http://127.0.0.1:8000/rm-scores

9. Conclusion

The task was successfully completed by aggregating RM performance scores from the database and exposing the results through a RESTful API. The solution is secure, reproducible, and adheres to standard backend development practices.