gjg-backend-challenge

Release 1.0.0

Deniz Gokcin

Aug 02, 2020

CHAPTER ONE

GJG-BACKEND-CHALLENGE

A REST API endpoint, that manages a game which uses a leaderboard with players submitting new scores from around the world.

1.1 Requirements

- docker
- docker-compose

1.2 Used Containers

```
flask: Contains the Flask application and uWSGI application server.
nginx: Contains the Nginx web server.
redis: Stores information about users & handles leaderboard interactions.
```

- The containers can be found under my docker-hub account

1.3 Building

```
docker-compose up -d
```

1.4 Deployment

- The API is deployed to my Docker Swarm running on my Digital Ocean Droplets. The application is distributed on 3 nodes. The main page can be reached from this link.

```
docker stack deploy -c docker-compose-swarm.yml gjg
```

1.5 Testing the endpoints

- To test the endpoints, you need to add users to the leaderboard. You can achieve this by posting sample-data.json to http://178.62.26.184/user/create. You can also add individual users using the same endpoint.
- You can get the leaderboard from http://178.62.26.184/leaderboard
- You can update a users score by posting to http://178.62.26.184/score/submit following the syntax in this document

1.6 Notes:

- GitHub Actions are used for automatically running pytests and deploying to DockerHub. ### Future Work:
- Although there are multiple worker nodes, the response time could be improved if more powerful droplets are used.

CHAPTER

TWO

HANDLERS PACKAGE

2.1 Submodules

2.2 handlers.leaderboard module

`handlers.leaderboard.`**`generate_leaderboard`**`(`*r*`)`

Generates the global leaderboard. Due to the use of sorted sets, as the data structure for the leaderboard, the time complezity of obtaining the leaderboard takes O(log(N)+M) with N being the number of elements in the sorted set and M the number of elements returned.

Parameters: r (RedisClient): Redis Client

Returns: leaderboard (list): The leaderboard as a list of dicts

`handlers.leaderboard.`**`generate_leaderboard_by_country`**`(`*r*, *iso*`)`

Generates the leaderboard and filters it by iso code.

Parameters: r (RedisClient): Redis Client iso (str): Cointry iso code

Returns: leaderboard (list): The leaderboard as a list of dicts

2.3 handlers.score module

`handlers.score.`**`update_user_score`**`(`*r*, *user_id*, *score_worth*`)`

Increments the score of a given player by score_worth. Time complexity of incrementing the score: O(log(N)) where N is the number of elements in the sorted set. Time complexity of updating user profile: O(1).

Parameters: r (RedisClient): Redis Client user_id (guid): guid score_worth (float): Score to increment

2.4 handlers.users module

`handlers.users.`**`get_rank_of_user`**`(`*r*, *guid*`)`

Returns the rank of a specific user in O(log(N)), due to the use of a sorted set.

Parameters: r (RedisClient): Redis Client user_id (guid): guid

Returns: rank (int): The rank of the given user

`handlers.users.`**`get_user_profile`**`(`*r*, *guid*`)`

Returns detailed information about a given user.

Parameters: r (RedisClient): Redis Client guid (guid): guid

Returns: user (dict): The user object as a dict.

`handlers.users.`**`register_user`**(*r*, *user_id*, *display_name*, *points*, *rank*, *country*)

Stores the json fields of user data in a redis hash. Stores the country iso code of a user in a redis set Adds the user to the leaderboard using player:<guid> as the key and the points as the value.

Parameters: r (RedisClient): Redis Client user_id (guid): guid display_name (str): Display Name points (float): Initial points rank (int): Initial rank, will be overriden once added to the leaderboard. country (str): Country iso code

2.5 Module contents