GoLang - Leaderboard Task

The task is to create a leaderboard API for games developed by Bidstack, as well as a client to test the API. The communication with the API has to be made via **gRPC** and should only be accessible with this protocol. The API server has to be written in **GoLang**.

In order to access the API, an authorization token should be provided. The token can either be hardcoded or stored inside of a database. The leaderboard should be stored inside of any kind of database - it is up to you to choose which one to use.

The client can be either an abstraction layer above the gRPC communication (meaning that the client would receive a JSON request, pass the data to API server and return the response as JSON) or a set of functions that would call the API. In case of choosing the second way (set of functions), the client should call the API and print results and relevant debug information on the screen.

The API must be capable of doing the following things:

- Authentication
- Storing player's score
- Getting the leaderboard

If possible, a Dockerfile should be provided so that the work can be tested. A readme file should also be present, instructing how to set up and test the project, as well as any other information that could be needed to test the project (e.g., authentication key).

The work should be submitted inside a Git repository.

Authentication

Must be implemented as a middleware that checks for authorization token before all API requests. If no token is provided or the token is invalid, the API should return an appropriate message/error code. The token can either be hardcoded or stored inside of a database.

Storing player's score

In order to store a player's score, a stream connection has to be opened to the API server. This means that all the requests and responses will be performed within one persistent stream.

The request should consist of player name and score (integer). The API should store the score of this player. If such player does not exist in the database, it has to be created. If a player with such name already exists, the score for this player should only be updated **if the score** passed to the API **is larger** than the one stored in the database.

The response should return player's position within the leaderboard.

Getting the leaderboard

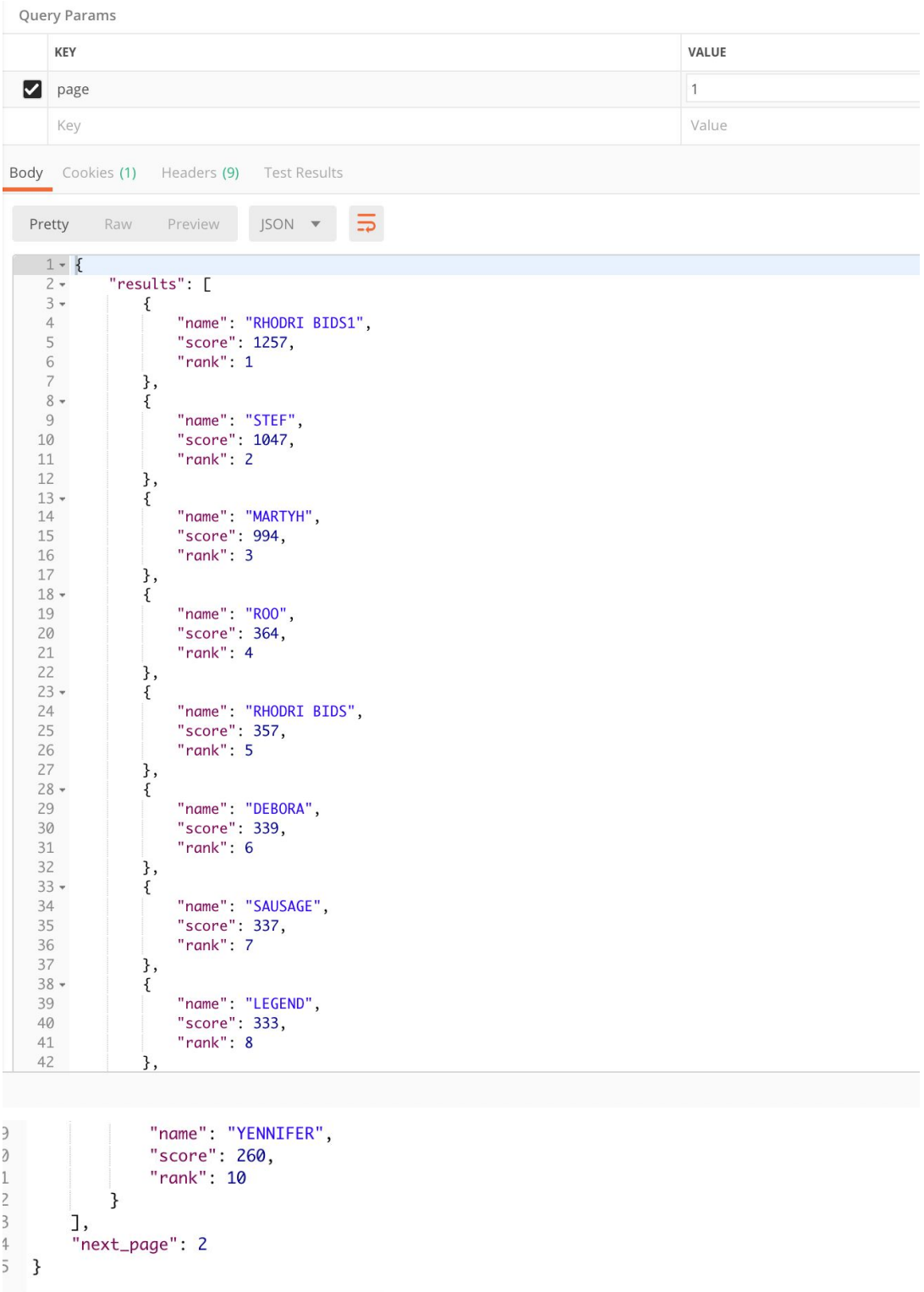
It should be possible to fetch the leaderboard using a regular gRPC request. The request should consist of player's name (optional) and page number and option to see the all

time/monthly leaderboard. Each request to the leaderboard should return one page of results. One page should consist of 10 results (though that should be configurable).
If the name of the player is passed and the player is not in this list of results (and their result is not in any of the previous pages), a list of players around the current player should be returned. Each object of a player has to contain their name, score and position (rank).
The number of the next page should also be returned in the response. If there is no next page (you have reached the end), the number should be 0.
If no page number is passed, the first page should be returned. If the page number is invalid, an appropriate error should be returned.
There should also be an option to see monthly and all time leaderboard. For example, if an appropriate parameter is passed, the request should return results from all time the database (and the API) has existed. If no such parameter is passed, only results submitted this month should be returned. There should be seeders (or similar ways) that would preload some results for past months into the database so that this can be tested.

Please see some examples on the next page.

Examples for fetching the leaderboard
Please note that the examples are shown in JSON, but the API has to be done with gRPC.

Passed page number 1, got the first page with results. The number of the next page is also provided.



Passed player name that is in 3rd page:

KEY	VALUE
page	1
name	MORITZ
Key	Value


```
    },
    {
        "name": "YENNIFER",
        "score": 260,
        "rank": 10
    }
],
"around_me": [
    {
        "name": "ZANNA",
        "score": 108,
        "rank": 28
    },
    {
        "name": "MORITZ",
        "score": 107,
        "rank": 29
    },
    {
        "name": "DAZWOLF306",
        "score": 107,
        "rank": 30
    },
    {
        "name": "NIBBA",
        "score": 106,
        "rank": 31
    },
    {
        "name": "QQ",
        "score": 102,
        "rank": 32
    }
],
"next_page": 2
```

Loading 3rd page (with the name still present):


```
{
    "results": [
        {
            "name": "ARTUR",
            "score": 116,
            "rank": 21
        },
        {
            "name": "FRAN",
            "score": 113,
            "rank": 22
        },
        {
            "name": "MIZENATOR",
            "score": 112,
            "rank": 23
        },
        {
            "name": "FISHISTWONK",
            "score": 111,
            "rank": 24
        },
        {
            "name": "DIMA",
            "score": 111,
            "rank": 25
        },
        {
            "name": "SOFIA",
            "score": 109,
            "rank": 26
        },
        {
            "name": "BIGD",
            "score": 109,
            "rank": 27
        },
        {
            "name": "ZANNA",
            "score": 108,
            "rank": 28
        },
        {
            "name": "MORITZ",
            "score": 107,
            "rank": 29
        },
        {
            "name": "DAZWOLF306",
            "score": 107,
            "rank": 30
        }
    ],
    "next_page": 4
}
```